Análise e implementação de algoritmos para localização e mapeamento de robôs móveis baseada em computação reconfigurável

Marcelo Carvalho Sacchetin

Orientador:
Prof. Dr. Eduardo Marques

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC, USP, para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Dezembro de 2005

Análise e implementação de algoritmos para localização e mapeamento de robôs móveis baseada em computação reconfigurável

Marcelo Carvalho Sacchetin

Dedicatória

Dedico este trabalho a meu pai e minha mãe por tudo.

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais por todo apoio em todos os sentidos;

Ao meu orientador Prof. Dr. Eduardo Marques que sempre estave disposto e presente para ajudar em tudo que foi preciso;

À minha namorada Priscila por sempre me incentivar e por todos os momentos de alegria que vivemos juntos;

Aos meus irmãos Fernando e Marina por toda compreensão;

Ao meu tio "Vê"por estar sempre ao meu lado;

Ao colega Denis Wolf que mesmo distante foi responsável por ajudas imprescindíveis para o desenvolvimento do trabalho;

Aos colegas do LCR pelos bons momentos de trabalho e descontração;

Aos meus companheiros de moradia Júlio, Léo e Lionis por todos os momentos de diversão na nossa memorável República Enxofre;

A Deus pela minha vida e pela oportunidade de poder trabalhar no que eu mais gosto e em um local tão agradável;

E, por fim, ao CNPq pelo apoio financeiro.

.

"Não cruzarás o mesmo rio duas vezes, porque outras são as águas que correm nele."
(Heráclito de Éfeso)

SACCHETIN, M. C. *Análise e implementação de algoritmos para localização e mapeamento de robôs móveis baseada em computação reconfigurável*, São Carlos, 2005. - Instituto de Ciências Matemáticas e de Computação - ICMC, USP.

Resumo

Localização e Mapeamento são problemas fundamentais da robótica que vêm sendo estudados exaustivamente pela comunidade científica para a navegação de robôs móveis. A maior parte das pesquisas estão concentradas em implementações para computadores pessoais, mas pouco se tem feito na área de computação embarcada. Este trabalho mostra a análise e implementação em FPGA de um algoritmo de localização para ambientes dinâmicos composto por um filtro de partículas, e também de um algoritmo de mapeamento baseado na técnica de *scan matching*. Os algoritmos originais desenvolvidos em linguagem de programação C foram analisados e modificados para uma abordagem embarcada (*embedded*) em robôs reconfiguráveis utilizando-se o processador Nios II da Altera. Os algoritmos são comparados quanto ao desempenho, no intuito de servir como referência no futuro desenvolvimento da ferramenta de *codesign* automático ARCHITECT+.

SACCHETIN, M. C. *Análise e implementação de algoritmos para localização e mapeamento de robôs móveis baseada em computação reconfigurável*, São Carlos, 2005. - Instituto de Ciências Matemáticas e de Computação - ICMC, USP.

Abstract

Localization and Mapping are fundamental robot navigation problems wich currently has been exaustily studied by scientific comunity. Most of research is concentrated on implementation for personal computers, and the robot navigation is done on static environment. But, these algorithms can not be directly applied for embedded solutions on dynamic environments. This work shows an analysis and implementation on FPGA of a localization algorithm for dynamic environments composed of a particle filter, and by an mapping algorthm known as scan matching. The original algorithm devoloped on C programming language for PCs are analised and modified for an embedded approach to mobile robots using Altera Nios II processor. Both C and embedded algorithms are compared within performance, to serve as reference on a future developement of automatic codesign tool ARCHITECT+.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo

1

Introdução

1.1 Contextualização

Localização e Mapeamento são problemas fundamentais para a navegação de robôs móveis que vêm sendo estudados exaustivamente na literatura de robótica recente. Em parelelo, outra área de pesquisa que tem obtido grande atenção pela comunidade científica nos últimos anos é a de sistemas embutidos. Ao contrário de processadores de propósito geral como os encontrados em computadores pessoais, sistemas embutidos são projetados para executar tarefas específicas com extrema eficiência [14].

A união dessas duas importantes e emergentes áreas de pesquisa (robótica móvel e sistemas embutidos) tem se mostrado extremamente promissora [15], consistindo o objetivo principal desse trabalho, pois, implementações de algoritmos em liguagem de alto nível serão portadas para robótica móvel embarcada. Soluções já existentes em C, incluindo algoritmos de localização [13] e mapeamento [16] são modificados para executar em FPGA em ambientes dinâmicos.

Este trabalho também utilizará, além do algoritmo desenvolvido e gentilmente cedido por Denis F. Wolf, arquivos de log de experimentos realizados por este mesmo autor na *University of Southern California*.

Os dados contidos no arquivo de log são fornecidos ao filtro de partículas durante a execução. O filtro de partícula que é responsável pela localização, juntamente com o scan matching, responsável pelo mapeamento, são executados em FPGA.

1.2 Motivação e Objetivos

Nos últimos anos o interesse por robôs móveis tem crescido muito e grandes avanços foram obtidos nessa área. Contudo, a tarefa de programar um robô ainda se mostra uma tarefa extremamente árdua tanto em termos da complexidade quanto do tempo desprendido. A ferramenta ARCHITECT+ [15] [18] vem sendo desenvolvida com o propósito de tentar amenizar essas dificuldades, através de uma geração automática de hardware especializado para robôs móveis a partir de algoritmos descritos em linguagem de programação de alto nível.

Neste trabalho será implementado em hardware através de computação reconfigurável os algoritmos de localização e mapeamento que poderá servir como base para um futuro estudo de caso para a ferramenta ARCHITECT+. Para tal tarefa, inicialmente os códigos originais serão analisados através de ferramentas específicas para medição do desempenho da execução dos programas. Identificando-se as porções de execução mais custosas, pretende-se implementá-las em hardware visando ganho em desempenho.

1.3 Estrutura

Este documento apresenta a seguinte organização:

- O **capítulo 2** apresenta uma revisão geral sobre robótica, seus diferentes paradigmas e arquiteturas, além de uma contextualização do mercado atual de robôs móveis.
- No **capítulo 3** é feita uma introdução às questões relacionadas ao problema de localização e mapeamento de robôs.
- No **capítulo 4** são apresentados algoritmos de localização, mapeamento e o SLAM (*Simultaneous Localization and Mapping*).
- O **capítulo 5** discorre sobre os principais aspectos da computação reconfigurável levando em consideração aspectos de hardware e software.
- No **capítulo 6** são levantadas questões relacionadas à análise de desempenho de sistemas de computação e uma breve descrição da ferramenta gprof [19].
- No **capítulo 7** é analisado a implementação do algoritmo de localização utilizado. É feita uma descrição completa, além de análises de tempo de execução, de fluxo de execução, e de estrutura (variávies, funções, dados de entrada e saída);
- O **capítulo 8** apresenta todos aspectos da implementação em FPGA dos algoritmos de localização e mapeamento descritos no capítulo 7.
- A conclusão e porposta de trabalhos futuros são apresentadas no capítulo 9

- Finalmente, são apresentadas as **Referências Bibliográficas**.

Capítulo

2

Robótica Móvel

Esse capítulo tem por objetivo esclarecer alguns conceitos básicos sobre robótica abordando questões, tais como os paradigmas de robótica, arquiteturas de robôs e navegação de robôs. Também são abordadas algumas questões relativas ao mercado atual de robôs móveis.

2.1 Considerações Iniciais

Robôs são dispositivos mecânicos versáteis (por exemplo, braços mecânicos articulados, veículos terrestres, aéreos ou submarinos, etc.) equipados com sensores e atuadores sob o controle de um sistema computacional. Eles realizam tarefas executando movimentos num espaço físico. Este espaço é povoado por vários objetos e está sujeito às leis da natureza [20].

Numa primeira etapa, houve um grande desenvolvimento na área de robótica industrial. A implantação no chão de fábrica de robôs industriais trouxe benefícios com a melhoria da eficácia e da qualidade, redução da mão-de-obra, maior confiabilidade e redução de custos. Vantagens adicionais incluem a capacidade de realizar tarefas para as quais os humanos teriam grande dificuldade e a remoção de humanos de tarefas em ambientes perigosos. Esses robôs eram pouco sofisticados e na maioria das vezes atuavam em ambientes estáticos.

Em uma nova etapa de evolução. pesquisadores em robótica têm concentrado esforços na construção de robôs móveis, introduzindo as capacidades de mobilidade e autonomia para reagir adequadamente a ambientes dinâmicos [15], o que abre uma vasto campo de novas aplicações e consequentemente novos desafios.

São encontradas aplicações para robótica móvel nas mais diversas áreas, tais como

a utilização de veículos autoguiados em estoques de produtos manufaturados, veículos planetários utilizados para exploração de outros mundos, robôs para vigilância e segurança, robôs enfermeiros, robôs utilizados como guias em museus, dentre outras [21].

2.2 Mercado de Robôs Móveis

Paralelo ao crescimento das aplicações, segue o crescimento do mercado de robôs móveis. Estes robôs utilizados para serviços privados são denominados *Personal Robots* (PR) [3]. O mercado dsse tipo de robô se apresenta em ascenção, pois o custo de cada unidade representa apenas uma fração do preço de um robô industrial. Dessa forma, o comércio em massa dos PRs só tende a crescer, como por exemplo no caso do aspirador de pó lançado em 2001, que em apenas dois anos após o seu lançamento vendeu mais de 570.000 unidades.

Esses levantamentos têm sido realizados por órgãos sérios tais como a *United Nations Economic Commission for Europe* (UNECE) e a *International Federation of Robotics* (IFR) e apontam um grande crescimento previsto para os próximos anos, conforme pode ser observado na figura 2.1.

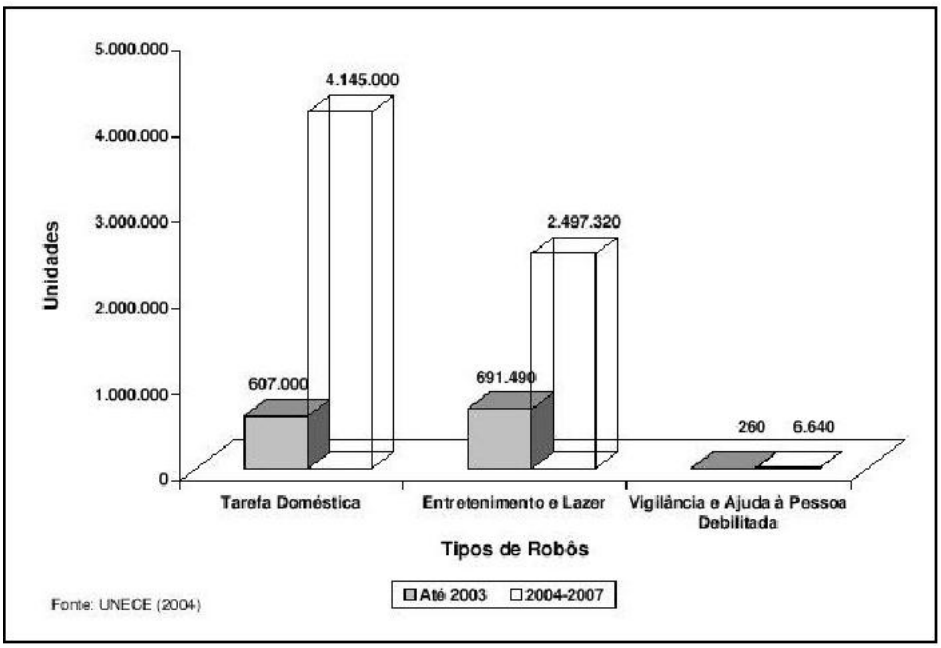

Figura 2.1: Estimativa de crescimento de mercado de PRs até 2007 [2]

As espectativas são boas até para robôs com pouca inteligência, como no caso do robô aspirador de pó que navega aletóriamente pelo ambiente. Mesmo com as limitações de inteligência, onde durante a operação certos locais podem ficar mais limpos do que outros, esse robô já vendeu até hoje mais de 1 milhão de unidades.

Dentro desse contexto, pode-se estipular para o mercado de futuro de robôs móveis pessoais que:

- É necessário que o custo do robô seja baixo, pois os compradores são representados por pessoas físicas;
- É necessário que os robôs tenham um maior nível de inteligência e autonomia para conseguirem realizar as tarefas com maior qualidade.

Atualmente, vários trabalhos para o barateamento dos custos vêm sendo realizados, como por exemplo a utilização de câmeras como sensores para que o robô possa capturar os dados do mundo externo. Dentre as soluções atuais podemos citar a utilização de multi-câmeras para navegação [3] (figura 2.2), além de sistemas de visão omnidirecional através de espelhos hiperbólicos [4] e [22] (figura 2.3).

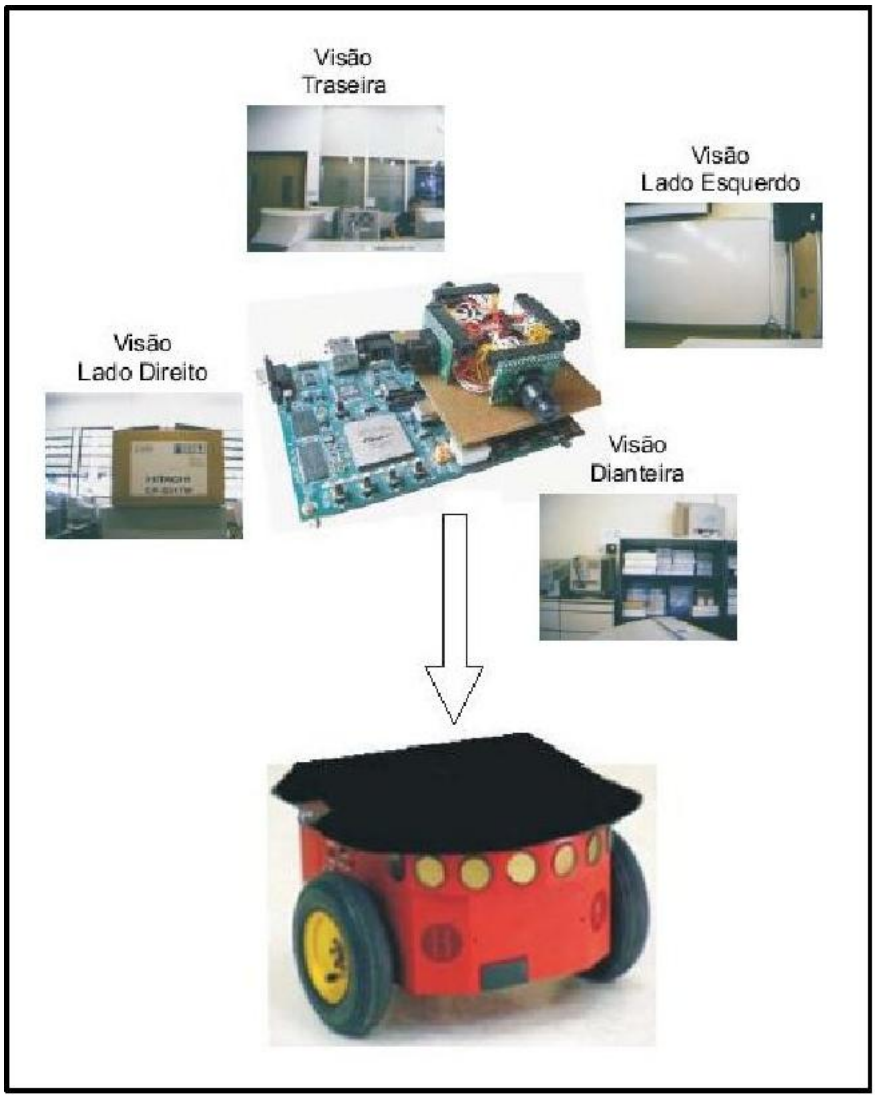

Figura 2.2: Sistema multi-câmeras embarcado na base de um robô Pioneer 3DX [3]

Em relação a criação de robôs com maior inteligência, ocorrem alguns problemas: diversos algoritmos sofisticados e eficientes para esse tipo de exigência são desenvolvidos em linguagens de programação de alto nível para executarem em computadores pessoais de mesa (*desktops*) com um grande poder de processamento comparado a sistemas de

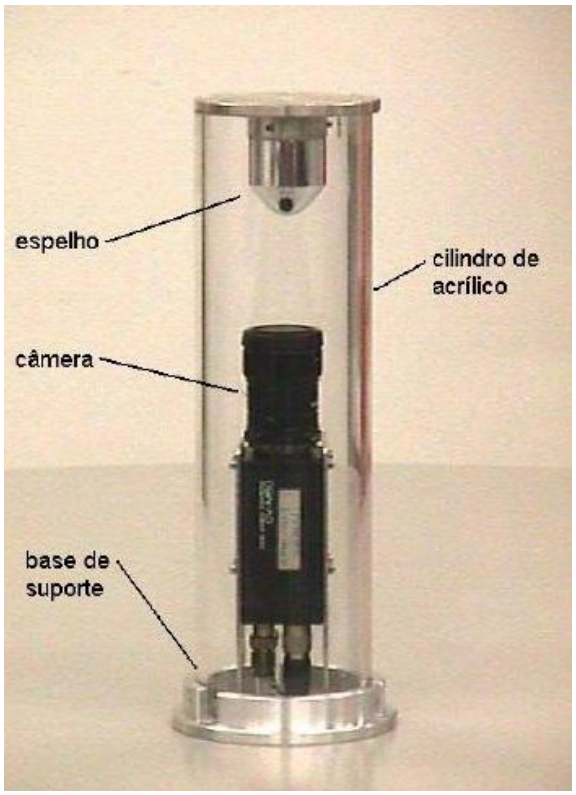

Figura 2.3: Sistema de visão omnidirecional utilizando uma câmera e um espelho hiperbólico [4]

pequeno porte, o que dificulta a sua utilização em PRs; a maioria dos algoritmos existentes conta com ambientes estruturados, o que não se aplica ao caso de robôs móveis que devem interagir fortemente com omeio externo. A figura 2.4 representa a relação entre nível de autonomia do robô e estruturação do ambiente.

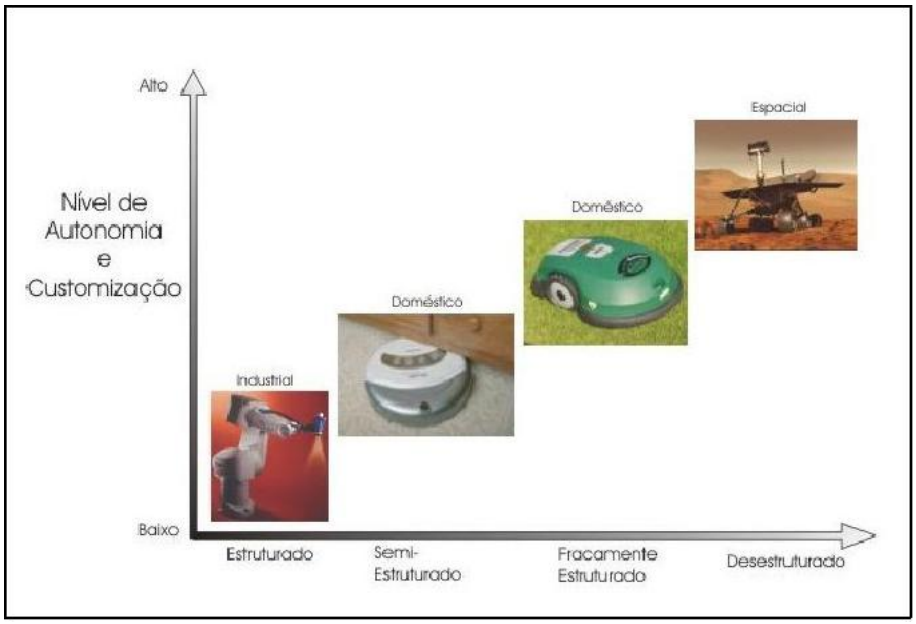

Figura 2.4: Relação entre nível de autonomia do robô e estruturação do ambiente [3]

Seguindo a linha de pensamento de que para o futuro do mercado de robôs móvies,

eles devem apresentar baixo custo, e possuir inteligência suficiente para serem capazes de navegar em ambientes pouco estruturados, este trabalho, conforme descrito no capítulo 8, busca emabarcar uma solução em linguagem de alto nível para localização e mapeamento de robôs.

2.3 Paradigmas da Robótica

Um paradigma é uma filosofia ou conjunto de concepções e/ou técnicas que caracterizam uma abordagem a uma classe de problemas. Um paradigma robótico o caracteriza a forma em que a "inteligência"pode ser estruturada em robôs. Três primitivas básicas e a forma de sua organização, diferenciam os paradigmas da robótica, são elas: SENTIR, PLANEJAR e AGIR. A SENTIR diz respeito às tarefas relacionadas com a recuperação de informações do ambiente em que o robô está inserido, por meio de seus sensores [23]. A PLANEJAR está relacionada ao processo de receber informações e a partir delas, produzir tarefas a serem conduzidas pelo robô. Por fim, AGIR produz comandos diretamente aos atuadores do robô (motores ou quaisquer dispositivos que modifiquem ativamente o ambiente).

Até o momento, existem três paradigmas robóticos, o Hierárquico, o Reativo e o Híbrido Reativo / Deliberativo. Os paradigmas divergem sobre dois aspectos:

1. Pela forma como eles relacionam as três primitivas da robótica: sentir, planejar e agir. Se uma função recebe dados dos sensores do robô e transforma esses dados em informações úteis para outras funções do robô, essa função é classificada na categoria de sentir. Se uma função recebe informações (quer seja direto dos sensores quer seja de outras funções) e produz uma ou mais tarefas que o robô deve realizar, essa função é classificada na categoria de planejar. Se uma função produz dados para os atuadores do robô, essa função é classificada na categoria de agir.

2. Pela forma como os dados sensoriais são processados e distribuídos pelo sistema. Os dados podem ser processados localmente por cada função ou podem ser processados por uma única função que produz um modelo global do mundo (ambiente onde o robô está atuando) e distribui essa informação para as demais funções.

A seguir, cada paradigma será descrito de forma sucinta [24].

2.3.1 Paradigma Hierárquico

O paradigma hierárquico foi o primeiro paradigma robótico. Neste, as primitivas robóticas estão organizadas da seguinte maneira. Primeiro é realizada a primitiva SENTIR, que obtém dados dos sensores e as integra em um modelo de mundo global. Em seguida, a primitiva PLANEJAR é executada, a qual irá interpretar o modelo de mundo global e

fornecer tarefas à primitiva AGIR que irá comandar os módulos de controle de baixo nível do robô. Essa organização, que pode ser observada na figura 2.5 , é cíclica, ou seja, após agir, o robô reiniciará o ciclo, sentindo, planejando e então agindo novamente.

Figura 2.5: Paradigma Hierárquico [5]

O principal problema desse paradigma é o tempo de resposta. Os processos de criação de modelos de mundo complexos e de planejamento são computacionalmente caros, e como nesse paradigma as primitivas estão organizadas seqüencialmente, o robô deve esperar o término de todo o ciclo para "sentir"o ambiente e identificar se suas ações foram realizadas corretamente assim como as mudanças causadas por outros agentes.

Um dos primeiros robôs móveis da IA foi Shakey [25]. Ele segue o paradigma hierárquico. Ele representa bem este paradigma pois tem a percepção do mundo totalmente separada da ação. Ele usa um planejador chamado STRIPS, o qual utiliza a análise meios-fins para inferir ações tomando como entrada um modelo simbólico do mundo obtido anteriormente ao planejamento.

2.3.2 Paradigma Deliberativo

O paradigma deliberativo é o mais antigos dos paradigmas. Ele dominou as implementações de robôs de 1967 até 1980 (antes do surgimento do paradigma reativo).

O modelo deliberativo é seqüencial por natureza. Inicialmente, o robô sente o mundo a sua volta e constrói um mapa global do ambiente. Depois disso, já sem receber dados dos seus sensores, o robô planeja a melhor forma de obter seus objetivos. Finalmente, o robô age de forma a seguir o plano traçado. Essa seqüência é seguida de forma cíclica durante toda a execução do algoritmo do robô [26].

Como pode ser observado na seqüência de ações do robô, o paradigma deliberativo recebe todos os dados dos sensores em uma etapa e gera a representação global do ambiente que é utilizada durante as fases de planejamento e atuação.

A principal vantagem desse paradigma é que ele é fácil de ser codificado, pois existe uma ordem clara entre as tarefas a serem executadas. O principal problema desse paradigma é que ele se baseia na suposição do mundo fechado, ou seja, ele supõe que

todas as informações que o robô precisa saber estão contidas no seu modelo de mundo. Isso é ruim por dois motivos:

1. É muito difícil especificar e armazenar todas as informações sobre o ambiente e, se houver o esquecimento de algum detalhe, o robô pode ser surpreendido e não funcionar corretamente.

2. Para realizar até mesmo tarefas simples, o robô precisa lidar com um conjunto de dados bastante grande no qual a maioria das informações são irrelevantes, o que torna o robô mais lento que o necessário.

Uma desvantagem adicional desse sistema é que o robô leva muito tempo para reagir a mudanças no seu ambiente. Se, por exemplo, um objeto se move rapidamente em direção ao robô e ele estiver em fase de planejamento (que geralmente é demorada), o robô não será capaz de desviar do objeto para evitar a colisão.

2.3.3 Paradigma Reativo

O paradigma reativo surgiu da insatisfação dos pesquisadores com o paradigma deliberativo e com o aumento do fluxo de idéias etimológicas [24]. O paradigma reativo recebe os dados dos sensores e já age, sem nenhum planejamento, ou seja, o ciclo sentir-planejar-agir fica reduzido a sentir-agir.

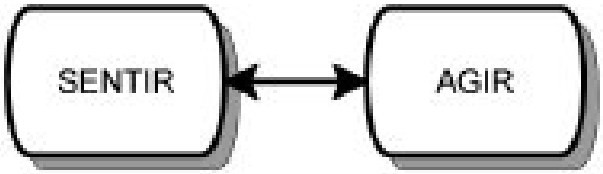

Figura 2.6: Paradigma Reativo [5]

Todo paradigma reativo é baseado no conceito de comportamentos. Um comportamento é uma mapeamento direto dos dados dos sensores em padrões de ações, as quais são utilizadas para se obter um determinado objetivo. Geralmente, um robô possui vários comportamentos. Esses comportamentos são organizados em camadas de forma que comportamentos em camadas superiores possam inibir comportamentos em camadas inferiores [5]. Um exemplo disso é um robô que possui o comportamento de ficar andando aleatoriamente por uma sala em uma camada, e numa camada superior possui o comportamento de seguir paredes. Enquanto o robô não identificar uma parede ele irá andar aleatoriamente. Assim que ele identificar uma parede ele irá seguí-la; e, portanto, irá parar de andar aleatoriamente. Desta forma, o comportamento de seguir paredes inibiu o comportamento de andar aleatoriamente.

A forma como um robô reativo funciona depende de todos os comportamentos que esse robô possui (isso é chamado de comportamento emergente). Quase sempre um robô com mais comportamentos é considerado mais inteligente do que outro. Contudo, na verdade, os robôs reativos não são inteligentes: eles apenas são estimulados e respondem a esse estímulo. A inteligência dos robôs reativos está nos olhos do observador [27]. Quem irá dizer que um robô que é capaz de buscar um objeto e trazê-lo até você não possui nenhum tipo de inteligência?

Nas primeiras implementações de robôs reativos foi usada a idéia de um sensor e um comportamento [24]. Entretanto, com o passar do tempo e com os resultados de inúmeras pesquisas, ficou comprovado que para comportamentos mais avançados era melhor utilizar os dados de vários sensores, pois isso implicava em um aumento da precisão das informações recebidas pelo comportamento.

Uma característica interessante dos robôs reativos é que eles não possuem nenhuma representação do ambiente; ou seja, "o mundo é a sua melhor representação"[27]. Se por um lado isso é uma vantagem sobre o paradigma deliberativo, que na maioria das vezes necessita de quantidades enormes de dispositivos de armazenamento de dados para manter suas representações do ambiente, por outro lado também é uma desvantagem. Como um robô reativo possui o comportamento de seguir uma parede sabe que ele não está andando em círculos? A resposta para essa pergunta é simples: ele não sabe! [24]

2.3.4 Paradigma Híbrido Reativo/Deliberativo

Apesar do sucesso alcançado com o paradigma reativo demonstrado por implementações de comportamentos, como desviar de obstáculos e seguir metas simples usando percepção orientada à ação, muitos roboticistas perceberam que o paradigma reativo não resolveria todos os problemas. Muitas tarefas requerem que os robôs tenham uma certa memória de uma representação de mundo, como por exemplo mapeamento e a localização, tarefas cruciais para que um robô possa navegar de forma autônoma por longas distâncias e grandes períodos de tempo[18].

O paradigma híbrido é capaz de unir os dois paradigmas anteriores de forma a não tornar os robôs lentos como no paradigma hierárquico, e ainda assim realizar planejamentos que possuem alto custo computacional.

Como pode ser visto na figura 2.7 , a primitiva PLANEJAR está presente, porém não mais ligada seqüencialmente com as outras. Neste paradigma, a organização é do tipo PLANEJAR, SENTIR-AGIR. Isso significa que o planejamento é feito em um passo e então sentir e agir são realizados juntos. Portanto, o planejamento não faz parte do ciclo de percepção-ação, interagindo nessa organização somente quando tem algum resultado relevante de seus planejamentos.

O planejador pode ser usado, dentre outras coisas, para definir a tarefa que a parte

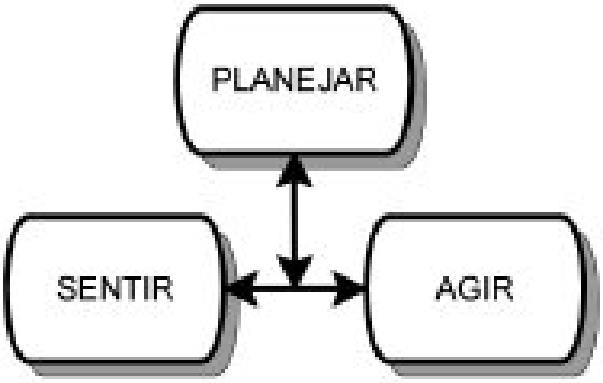

Figura 2.7: Paradigma Híbrido [5]

"reativa"irá realizar e para escolher quais são os comportamentos que devem estar ativos para a próxima tarefa.

2.4 Arquiteturas de Robôs

Arquiteturas de robôs são sistemas computacionais e especificações das características desejadas em um robô, os quais provêm linguagens e ferramentas para a construção de sistemas robóticos [28].

A escolha de uma arquitetura para o robô afeta todo o projeto e desenvolvimento do mesmo [29]. Assim, surgiu a seguinte pergunta: como avaliar a utilidade de uma determinada arquitetura? Arkin propôs os seguintes critérios [28]:

- *Robustez:* é a capacidade da arquitetura em prover tolerância a falhas.
- *Recursos de Desenvolvimento:* quais as ferramentas e os ambientes de desenvolvimento que a arquitetura provê para o desenvolvedor de sistemas robóticos.
- *Desempenho:* esse critério diz respeito ao tempo de execução do sistema, seu consumo de energia e, principalmente, se a tarefa para a qual o robô foi projetado foi realizada com sucesso.
- *Suporte à Paralelismo:* a maioria dos comportamentos de um robô são inerentemente paralelos, pois o robô necessita adquirir dados dos seus sensores, planejar uma forma de obter seu objetivo, evitar colidir com objetos pessoas e tudo isso deve ser feito ao mesmo tempo. Portanto, é importante considerar qual o suporte que a arquitetura fornece para a execução paralela de tarefas
- *Portabilidade de Hardware:* esse critério se refere à capacidade da arquitetura em se adaptar aos atuadores e sensores do robô e de a mesma ser mapeada diretamente em hardware.

- *Portabilidade de Ambiente e Utilidade:* testa se a arquitetura possui a capacidade de se adaptar a novos ambientes e a diferentes propósitos de utilização.
- *Suporte à Modularidade:* diz respeito ao suporte que a arquitetura provê para a reutilização dos componentes (via de regra componentes de software) do projeto atual em futuros projetos.

As arquiteturas de robôs podem ser classificadas em três categorias:

1. *Hierárquicas:* adota o estilo *top-down*. Esse estilo enfatiza a supremacia do controle de alto nível e tenta inibir as comunicações de baixo nível. As primeiras arquiteturas utilizavam esse estilo, mas hoje em dia ela é difícil de ser empregada. ela é empregada em robôs deliberativos.
2. *Comportamentais:* adota o estilo *bottom-up*. Esse estilo faz uso de módulos chamados comportamentos. Esses módulos são executados paralelamente e enfatizam a interação através da comunicação e com o ambiente. É empregada em robôs reativos.
3. *Híbridas:* mistura os dois estilos anteriores. Tenta facilitar a comunicação entre o controle eficiente de baixo nível (reativo) com controle de planejamento de alto nível (deliberativo). é empregada em robôs híbridos.

O maior problema enfrentado pelos desenvolvedores de sistemas robóticos contemporâneos é o fato de que não existe nenhuma arquitetura preponderante em nenhuma das três categorias: na maioria dos casos, a arquitetura é criada para o robô que está sendo desenvolvido [30]. Isso possui sérias implicações, tais como:

- A dificuldade de reutilização de código gerado durante o projeto de um robô para o projeto de outros robôs;
- A dificuldade de validação de um sistema robótico;
- A dificuldade nas especificação e desenvolvimento do sistema robótico.

O principal motivo pelo qual ainda não surgiu nenhuma arquitetura que seja considerada preponderante é o fato de que, na maioria das vezes, cada robô possui características e necessidades muito peculiares [18].

2.5 Robôs Probabilísticos

Robôs são inerentemente imprecisos quanto ao estado de seu ambiente. Essas imprecisões são conseqüências de limitações dos sensores, ruídos nos sensores e do fato de que a maioria dos ambientes são imprevisíveis. Na hora de se obter dados dos sensores, robôs

probabilísticos [31] [7] [17], computam distribuições probabilísticas sobre o que deve estar acontecendo no seu ambiente ao invés de gerar apenas um único resultado determinístico. Desta forma, eles podem se recuperar mais suavemente de possíveis erros, podem lidar com ambigüidades e podem integrar dados de vários sensores de forma mais consistente.

Qual é a vantagem de se usar robôs probabilísticos? A vantagem é que um robô que carrega uma noção de suas incertezas (de sua ignorância) e age de acordo, obterá melhores resultados do que um que não conhece suas próprias limitações.

As duas principais áreas em que a computação probabilística tem sido utilizada com sucesso são: percepção e controle. A percepção de um robô sobre seu ambiente é conhecida como o problema mais fundamental para prover um robô móvel com capacidades autônomas. Particularmente complexo, é o problema da localização global no qual um robô não sabe a sua posição inicial e tem que se localizar.

Sob uma ótica probabilística, o problema da localização é um problema de estimativa de uma densidade probabilística, onde o robô procura estimar uma distribuição posterior sobre o domínio de suas possíveis localizações condicionadas aos dados disponíveis.

O principal objetivo da robótica é construir robôs que ajam corretamente. Como mencionado anteriormente, um robô que tem noção de suas incertezas e levam em consideração essas incertezas na hora de tomar suas decisões tenderão a tomar decisões melhores. Os dois principais conceitos envolvidos nos algoritmos probabilísticos são a regra de Bayes e a Suposição de Markov [7].

A abordagem probabilística pode ser ilustrada através do seguinte exemplo [6]: localização de robôs móveis (detalhado no capítulo 3). A localização é o problema de se estimar as coordenadas do robô utilizando referências externas aos dados dos sensores usando um mapa do ambiente. A figura 2.8 ilustra a abordagem probabilística para localização de robôs móveis. O problema de localização específico estudado nesse caso é conhecido como localização global, onde o robô é posicionado em algum lugar do ambiente e tem que se localizar sozinho. No paradigma probabilístico a estimação momentânea do robô (também chamada crença - *belief : bel(x)*) é representada por uma função de densidade de probabilidade através do espaço de todas as localizações. Isso é ilustrado no primeiro diagrama da figura 2.8, que mostra a distribuição uniforme que corresponde à incerteza máxima.

Suponha que o robô faça a primeira medição com o sensor e observe que ele está próximo a uma porta. A crença resultade ($bel(x)$) mostrada no segundo diagrama da figura 2.8, posiciona probabilidade alta nos lugares próximos às portas e baixa probabilidade para qualquer outro lugar. Note que essa distribuição possui três picos, cada um correspondendo a uma das (indistingüiveis) portas no ambiente. Além disso, a distribuição resultante assume alta probabilidade para três locais distintos, mostrando que a estrutura probabilística pode manusear múltiplas e conflitantes hipóteses que naturalmente surgem das situações ambíguas.

Qualquer local que não seja uma porta possui probabilidade diferente de zero. Isso é ocasionado pela incerteza inerente em coletar dados de sensores: Com uma probabilidade pequena, diferente de zero, o robô pode assumir que não está próximo a uma porta no dado momento. Agora, suponha que o robô se move. O terceiro diagrama na figura 2.8 mostra o efeito do movimento do robô e sua crença ($bel(x)$), assumindo que o robô se moveu como indicado. A crença é mudada em direção ao movimento. Ela é também suavizada, para adicionar a incerteza inerente na locomoção do robô.

Finalmente, o quarto diagrama na figura 2.8 mostra a crença depois de observar uma outra porta. Essa observação leva o algoritmo utilizado a colocar a maior massa de probabilidade em um local próximo à porta, e o robô está agora bem mais confiante de onde ele está.

2.6 Locomoção de Robôs Móveis

Tarefas sofisticadas requerem que um robô móvel construa mapas e use-os para se deslocar [32]. A navegação de robôs móveis é realizada basicamente por meio de três tarefas: localização, mapeamento e planejamento de trajetória, as quais são definidas a seguir.

2.6.1 Localização

Para operar de forma autônoma, robôs móveis precisam saber onde se encontram. Localização (comumente referida por estimativa de posição ou controle de posição) é o processo de determinar e manter o controle da posição de um robô móvel relativo a seu ambiente. Localização precisa é um requisito chave para a navegação em ambientes de larga escala, particularmente quando modelos globais são usados, como mapas, desenhos, descrições topológicas e modelos CAD [17].

Atualmente, os sistemas de robôs móveis de sucesso utilizam localização [32]. Ocasionalmente, localização tem sido referida como o problema fundamental para dar capacidades autônomas a robôs móveis [33].

O tópico de localização de robôs móveis será tratado com maior profundidade no **capítulo 3**.

2.6.2 Mapeamento

O mapeamento consiste na tarefa de criar um modelo do ambiente do robô por meio da integração de diversas leituras de seus sensores. O principal problema do mapeamento está no ruído e no erro introduzidos por seus sensores. Se o robô pudesse sempre saber corretamente sua localização e seus sensores fossem livres de ruído, a tarefa de mapeamento seria trivial. Porém, neste ponto, entramos em um "dead lock"pois para realizar

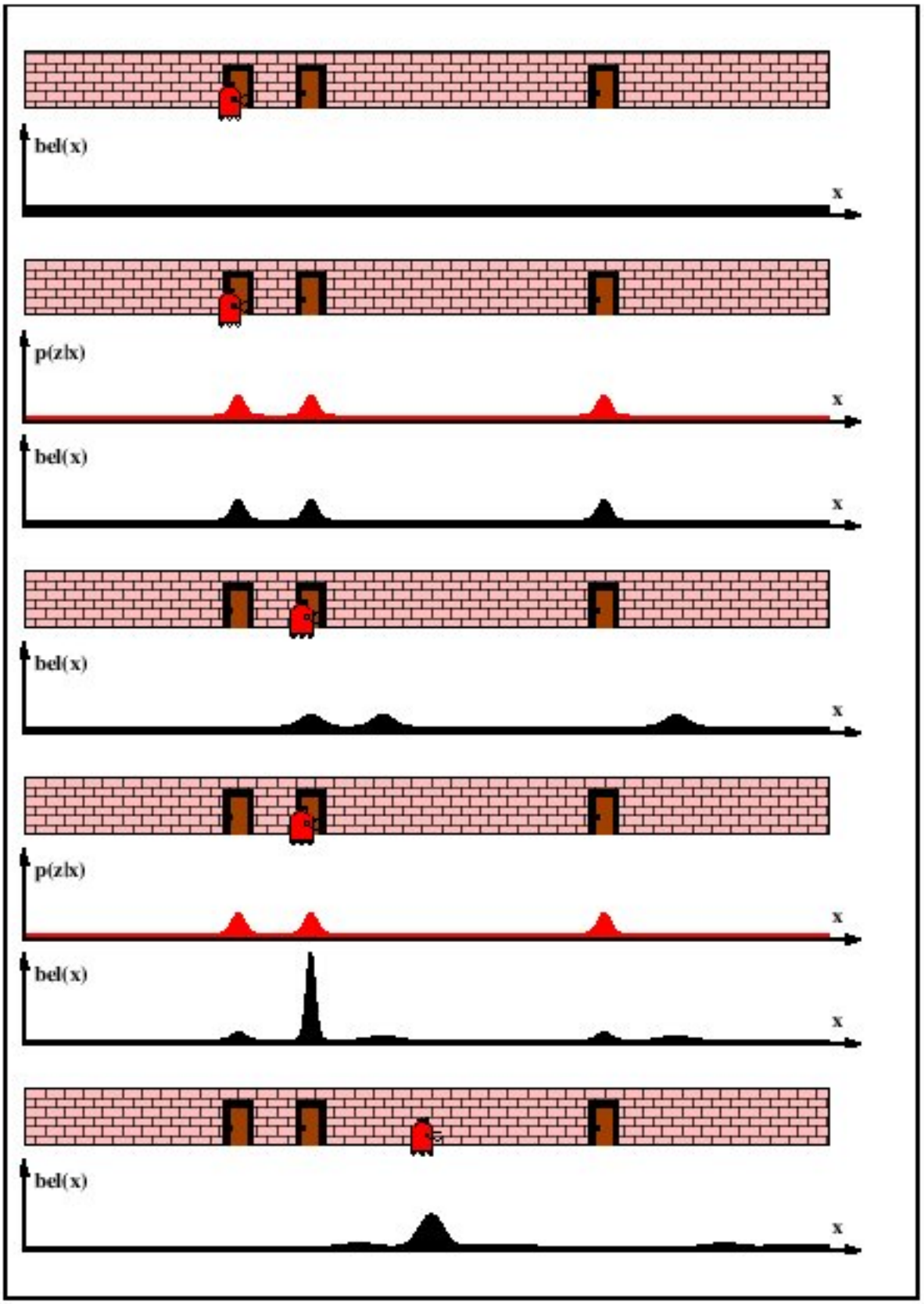

Figura 2.8: Idéia básica da localização de Markov: Um robô móvel durante a localização global [6]

localização é necessário um mapa, e para criar um mapa é necessário saber a posição do robô.

O processo de construção de mapas pode ser considerado como dois tópicos distintos. O robô tem que interpretar as leituras de seus sensores de forma a fazer deduções precisas sobre o estado de seu ambiente. Este é o problema de "criação de mapas". Ele tem que escolher pontos de vista de modo que as medidas dos sensores contenham informações úteis e novas. Este é o problema de "exploração"[34].

2.6.3 Navegação

Através da localização o robô é capaz de calcular sua poisção no ambiente, enquanto os algoritmo de mapeamento são responsáveis por criar um mapa desse ambiente, o que poissibilita criar rotas para locais que devam ser alcançados e estabelecer outras tarefas a serem realizadas no ambiente. Essa tarefa de planejamento de rota para que o robô possa se locomover pelo ambiente fica a cargo do algoritmo de navegação, dentre os quais podemos citar o VFH (Histograma de campos vetoriais) e o algoritmo de campos potenciais [21].

O algoritmo VFH mapeia localmente todos os obstáculos através de um mapa para que seja possível traçar uma rota até o objetivo final sem que haja nenhuma colisão. Os mapas locais são atualizados incorporando novas característica ocasionadas por mudanças no ambiente a cada execução do algoritmo que não se encerra até que o robô consiga chegar até o local estipulado como o alvo.

No caso dos algoritmos de campos potenciais os obstáculos e alvo final são representados como forças presentes no ambiente, de tal forma que os obstáculos repelem o robô em quanto o alvo o atrai em sua direção. Dessa forma, o robô é capaz de atingir o alvo desviando dos obstáculo pelo caminho [14].

2.7 Considerações Finais

Neste capítulo foi feita uma revisão de vários conceitos que são básicos para o entendimento do funcionamento de robôs. No capítulo seguinte serão descritos os problemas de localização e mapeamneto de robôs que servem como base conceitual para o estudo dos algoritmos implementados em FPGA.

Capítulo

3

Localização e Mapeamento para Robôs Móveis

3.1 Considerações Iniciais

Localização de robôs móveis, ou simplesmente localização, é uma tarefa que consiste em estimar a posição de um robô em seu ambiente por meio de informações de seus sensores. Para isso, o robô deve ter, a priori, um mapa do ambiente, o qual pode ser construído automaticamente (pelo robô) ou manualmente (por seres humanos, usando ferramentas de CAD , por exemplo).

O processo de mapeamento de ambientes pode ser dividido em duas tarefas: síntese do mapa e exploração. A síntese do mapa é uma tarefa passiva do ponto de vista do controle. Ele consiste em dadas as observações do ambiente, construir uma representação do mesmo. A exploração é um processo ativo que visa controlar o robô de forma que ele "visualize"com seus sensores todas as partes do ambiente [21].

Os mapas gerados por métodos de mapeamento automático possuem várias vantagens sobre a confecção manual e sobre a aquisição por modelos CAD ou plantas de projetos arquitetônicos:

1. **Mudanças no ambiente:** O ambiente do robô muda com o tempo. Deste modo, torna-se difícil manter um mapa preciso manualmente.

2. **Relacionando o Mapa aos Sensores:** É difícil para um usuário predizer quais características do mundo serão facilmente reconhecíveis pelos sensores do robô. Quando é utilizado um sensor de varredura de distâncias baseado em raio laser, dependendo da altura em que ele for instalado no robô, o mapa que representa as

características identificáveis por ele poderá ser diferente para cada altura. Um mapa provido pelo usuário pode então ser de pouco valor.

3. **Nível de Detalhe:** Para alguns propósitos, o nível de detalhe requerido pode ser mais alto do que o que pode ser obtido por meio de um projeto arquitetônico. Por exemplo, em um projeto arquitetônico pode estar especificado o local de uma mesa, já no mapa gerado automaticamente, se o sensor do robô está abaixo da altura da mesa, cada um dos pés da mesa serão representados.

Localização é aplicada em conjunto com muitos conjuntos de representação de mapas. Dentre os mapas baseados em localização, temos como exemplo os de *occupancy grid* (seção 3.4.3), com algumas instâncias mostradas na figura 3.1. Essa figura mostra um mapa 2D desenhado a mão, um diagrama de mapa topológico, e uma imagem de mosaico de um forro (que também pode ser utilizado como mapa). É imensa a quantidade de representação de mapas utilizadas nas pesquisas atuais.

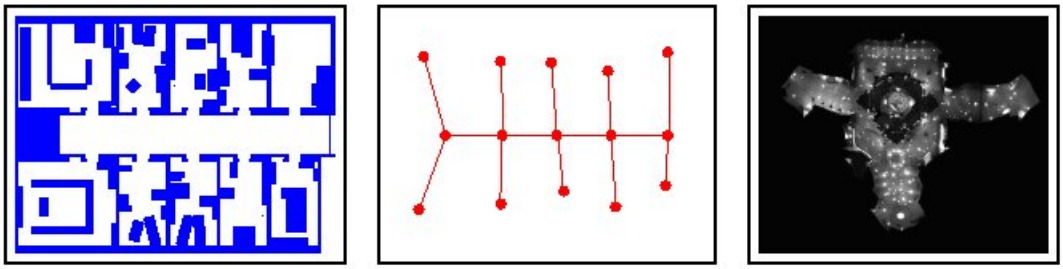

Figura 3.1: Exemplos de mapas utilizados para localização de robôs: *layout* 2D, diagrama de mapa topológico e imagem de um mosaico de um forro [6]

3.2 Taxonomia para os Problemas de Localização

Existem várias abordagens entre os métodos que resolvem o problema de localização. Uma possível categorização as divide em quatro dimensões [7]:

1. locais ou globais;
2. para ambientes estáticos ou dinâmicos;
3. ativas ou passivas;
4. um único robô ou múltiplos robôs.

A seguir, as diferenças entre elas serão brevemente descritas [6].

3.2.1 Métodos Locais e Globais

As abordagens locais para o problema de localização exigem que se tenha a posição inicial exata do robô a priori, pois elas somente são capazes de compensar os erros odométricos para manter coerente a informação da localização do robô. Ou seja, elas não conseguem recuperar a posição do robô se ela for perdida devido à falhas.

As abordagens globais são mais robustas. Elas podem estimar a posição de um robô mesmo sem nenhuma informação a priori sobre sua localização. Outra vantagem das abordagens globais é a sua capacidade de se recuperar de falhas. Algumas podem até mesmo tratar um tipo especial de falha conhecido como o "problema do robô seqüestrado", no qual o robô é retirado de seu local e posicionado em um outro sem que ele seja informado sobre esta manobra. Este problema é mais complexo do que a localização global sob total incerteza, pois neste caso, ao invés de não ter informações sobre sua posição, ele possui uma informação errônea.

Os problemas de localização podem ser caracterizados pelo tipo de conhecimento disponível inicialmente e durante o tempo de execução. Assim podemos classificá-los em três de acordo com o grau crescente de dificuldade:

- **Marca de posição (*Position tracking*):** A marca de posição assume como conhecida a localização inicial do robô. A localização do robô pode ser alcançada pela acomodação do ruído na movimentação do robô, pois o efeito dos ruídos são usualmente pequenos. Assim, a incerteza é aproximada por distribuições unimodais (e.g. Gaussianas), já que a marca de posição é um problema local, pois a incerteza é restrita a região próxima a posição real do robô.

- **Localização global:** Aqui a posição inicial do robô é desconhecida. O robô é inicialmente posicionado em algum lugar do ambiente. Abordagens globais não podem ser tratadas como limitadas em relação ao erro de posição. Portanto probabilidades unimodais são geralmente inapropriadas para estes casos.

- **Problema do robô seqüestrado:** É uma variante do problema de localização global, mas com o agravante que o robô é tirado de uma posição e colocado em outra, sem que ele seja informado disso.

3.2.2 Ambientes Estáticos e Dinâmicos

Um ambiente estático é aquele em que o único objeto que se move é o robô. Todos os outros objetos que os sensores do robô podem detectar estão representados no mapa. Em contraste, os ambientes dinâmicos podem possuir vários objetos móveis e não presentes no mapa.

A maioria dos métodos de localização pode lidar apenas com ambientes estáticos, ou ambientes com poucos elementos dinâmicos, os quais são tratados como ruído. Porém, existem abordagens, como a proposta por [35], que usam filtros para eliminar as leituras de sensores que possuam alta probabilidade de representar objetos dinâmicos. Dessa forma, pode-se usar métodos para ambientes estáticos, mesmo em locais altamente populosos como museus e bibliotecas.

- **Ambientes estáticos:** São ambientes em que a única variável presente é a posição do robô.
- **Ambientes dinâmicos:** São ambientes que possuem outros objetos que mudam de lugar, além do próprio robô.

3.2.3 Métodos Passivos e Ativos

A localização passiva trata exclusivamente da estimativa da posição do robô baseando-se em um fluxo de dados provenientes de sensores. Nesta abordagem, a movimentação do robô e a direção de seus sensores não podem ser controladas pela tarefa de localização. Em métodos ativos, a rotina de estimativa de posição tem o controle total ou parcial do robô, provendo uma oportunidade de melhorar a eficiência e a robustez da localização. As principais questões em aberto das abordagens ativas são "para onde se mover"e "para onde olhar"de forma a melhor localizar o robô [36].

- **Métodos passivos:** Na abordagem passiva, o módulo de localização apenas observa o robô operando. O robô é controlado por outros meios, e a locomoção do robô não é guiada para facilitar a localização.
- **Localização ativa:** Algoritmos de localização ativa controlam o robô para minimizar o erro de localização e/ou custos provenientes de uma movimentação limitada de um robô localizado em um ambiente perigoso.

Uma situação exemplo onde a localização ativa é mais adequada, é ilustrada na figura 3.2. Aqui o robô é posicionado em um corredor simétrico, e sua crença $bel(x)$ depois e navegar por um tempo está centrada em duas (simétricas) posições. A simetria local do ambiente torna impossível localizar o robô enquanto ele está no corredor. Apenas se ele entrar em um cômodo ele poderá eliminar a ambigüidade e determinar sua posição. Em situações como essas que a localização ativa fornece melhores resultados: Ao invés de esperar até que o robô entre em um cômodo por algum motivo, a localização ativa pode reconhecer o impasse e mandá-lo para lá diretamente.

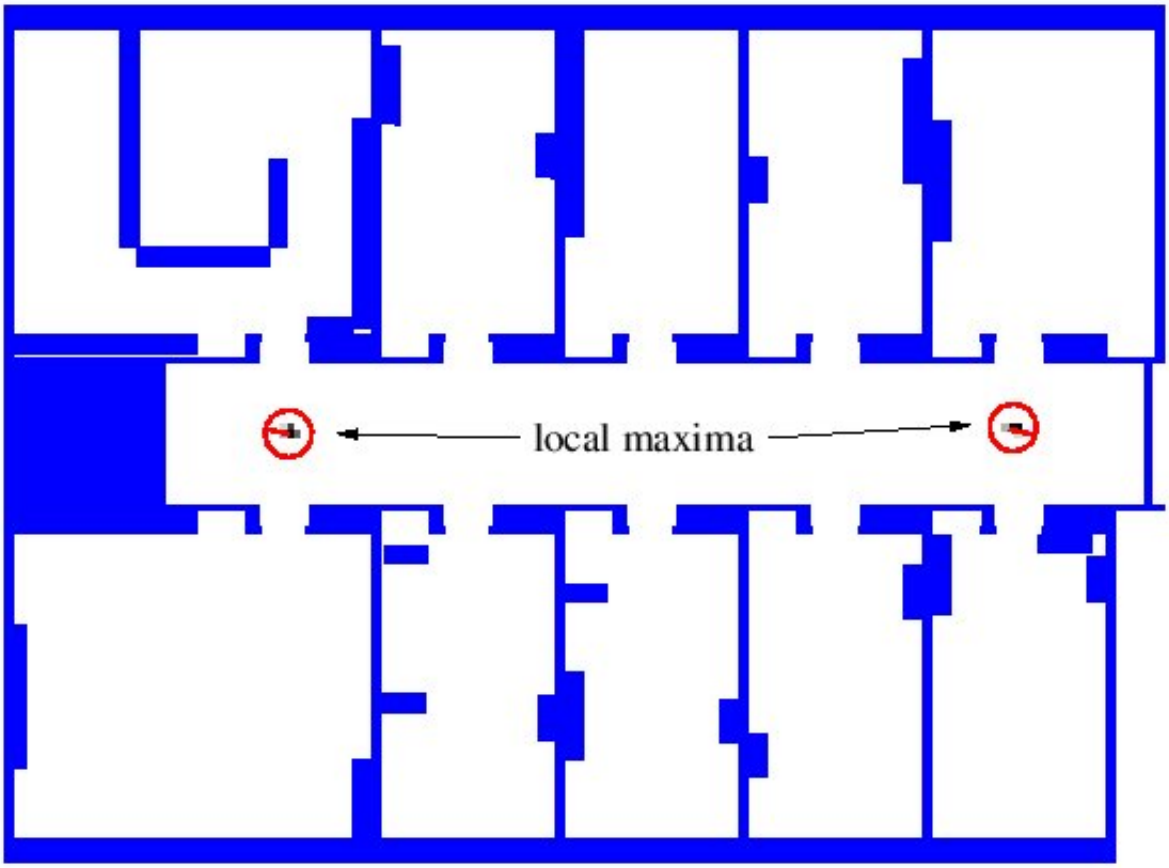

Figura 3.2: Corredor Simétrico. Exemplo de situação mais adequada para localização ativa [6]

3.2.4 Um Único Robô e Múltiplos Robôs

A abordagem mais comumente usada é a que envolve um único robô. Ela é conveniente, pois todos os dados são coletados em uma única plataforma e não existem problemáticas decorrentes de comunicação. Porém, em times de robôs, os métodos de localização que não ignoram o fato de existirem múltiplos robôs oferecem a vantagem de que o conhecimento de um robô pode ser usado para influenciar a crença de outro. Por exemplo, um robô X que sabe apenas que se encontra próximo a uma porta possui várias hipóteses sobre sua posição (cada hipótese corresponde a um lugar próximo a uma porta). Se X for detectado por outro robô Y que sabe exatamente sua própria posição, X poderá eliminar as falsas hipóteses baseando-se em informações fornecidas por Y [21].

- **Localização com um único robô:** Esse tipo de abordagem oferece a conveniência de que todos os dados são coletados em uma única plataforma de robô, e não há preocupação com comunicação

- **Localização com múltiplos robôs:** Primeiramente, cada robô se localiza individualmente, da mesma forma que ocorre em plataformas com um único robô. Se os robôs estiverem aptos a detectarem uns aos outros, isso ajudará na localização. A

localização com múltiplos robôs possui aspectos interessantes e mais complexos na representação das crenças e na natureza de comunicação entre elas.

3.3 Localização de Markov

Localização de Markov é um método probabilístico. Ele mantém múltiplas hipóteses sobre a posição do robô. Cada hipótese possui um peso associado que é um fator probabilístico numérico. A idéia principal é calcular uma distribuição de probabilidades sobre todos os possíveis locais do ambiente.

Nesta seção, será descrito o algoritmo básico desse método [35] que tem sido aplicado com muito sucesso em alguns trabalhos [31], [37].

Seja $l =< x, y, o >$ uma posição no espaço de estados, no qual x e y são coordenadas cartesianas do robô e O é a orientação do mesmo. A distribuição $Bel(l)$ sobre todos os locais l expressa a crença subjetiva que o robô tem em estar na posição l. Se a posição inicial é conhecida, $Bel(l)$ é centrada na posição correta, caso contrário, ela é uniformemente distribuída para refletir essa incerteza. Assim que o robô inicia sua operação, $Bel(l)$ é refinada de modo incremental.

A Localização de Markov aplica dois modelos probabilísticos diferentes para atualizar $Bel(l)$:

- um modelo de ação para incorporar os movimentos do robô;
- e um modelo perceptual para atualizar a crença baseando-se na entrada sensorial.

A movimentação do robô é modelada pela probabilidade condicional $p(l|l', a)$ que especifica a probabilidade de que uma ação a, quando executada em l' , leve o robô para l. $Bel(l)$ é então atualizada de acordo com a fórmula geral proveniente do domínio das cadeias de Markov, dada por:

$$Bel(l) \longleftarrow \sum_{l'} p(l|l', a) \cdot Bel(l') \tag{3.1}$$

O termo $p(l|l', a)$ representa um modelo da cinemática do robô. Na implementação proposta por [35], foi assumido que os erros de odometria são normalmente distribuídos.

As leituras dos sensores são integradas de acordo com uma fórmula de atualização Bayesiana. Seja s uma leitura de sensor e $p(s|l)$ a probabilidade de se perceber s dado que o robô está na posição l, então $Bel(l)$ é atualizada de acordo com a regra:

$$Bel(l) \longleftarrow ap(s|l)Bel(l) \tag{3.2}$$

na qual $alfa$ é um fator de normalização que garante que $Bel(l)$ totalize 1 sobre todo l.

Ambos os passos de atualização são aplicáveis apenas se o problema for Markoviano, ou seja, se as leituras de sensores passadas forem condicionalmente independentes de leituras futuras dada a localização do robô. Desse modo, a hipótese de Markov assume que o mundo é estático. Apesar de na prática essa abordagem ter sido aplicada até mesmo em ambientes que continham pessoas e que desta forma violavam a hipótese de Markov, os experimentos reportados em [35] mostram que ela não se adequa bem a ambientes densamente povoados.

Uma das formas de resolver esse problema consiste na aplicação de filtros que selecionam quais leituras dos sensores serão utilizadas efetivamente na localização. O intuito é eliminar as leituras influenciadas por objetos dinâmicos.

3.3.1 Localização de Monte Carlo

Existem diversas formas de se implementar a técnica de Localização de Markov. Elas diferem principalmente em três pontos:

- a representação da crença do robô;
- nos procedimentos de atualização da crença;
- e nos modelos probabilísticos de ação e percepção.

O algoritmo de Monte Carlo 3.3, representa a crença $Bel(s)$ por um conjunto de amostras s associadas a um fator numérico de importância p. Esse fator indica a probabilidade da amostra ser relevante para a determinação da posição do robô. A crença inicial é obtida gerando-se aleatoriamente N amostras da distribuição prévia $P(s_0)$, e atribuindo-se o fator de importância uniforme N^{-1} para cada amostra.

```
1:  S' = 0
2:  Psum = 0
3:  enquanto Psum < Pmax faça
4:          crie uma amostra aleatória < s, p > de S de acordo com p1, ..., pN
5:          gere um s' aleatório de acordo com P(s'|a, s,m)
6:          p' = P(o|s',m)
7:          adicione < s', p' > a S'
8:          psum = psum + p'
9:  fim-enquanto
10: normalize os fatores de importância p em S'
11: retorne S'
```

Figura 3.3: Algoritmo de Localização Monte Carlo [7]

Leituras de sensores o e ações a são processadas em pares. Monte Carlo constrói uma nova crença repetindo a seguinte seqüencia de "suposições": primeiro, um estado

aleatório s é gerado a partir da crença atual, sob consideração dos fatores de importância. Para aquele estado s, Monte Carlo supõe um novo estado s de acordo com o modelo de ação $P(s`|a, s, m)$ no qual m contém os dados do mapa. O fator de importância para esse novo estado s recebe um valor proporcional à consistência perceptual desse estado, como medido por $P(o|s, m)$. A nova amostra, juntamente com seu fator de importância, é memorizada e o laço básico é repetido. A geração de amostras é finalizada quando a soma total dos fatores de importância exceder um limite p_{max}, ou quando o próximo par $< o, a >$ chega. Finalmente, os fatores de importância são normalizados e o método de Monte Carlo retorna o novo conjunto de amostras definido como a crença corrente.

Uma desvantagem desse método é a de que ele possui uma tendência a descartar hipóteses corretas e às vezes não se recupera de falhas de localização global (não resolvendo o problema do robô seqüestrado). Felizmente, essa deficiência pode ser facilmente resolvida adicionando amostras aleatórias, matematicamente justificadas, assumindo que o robô pode ser seqüestrado com uma pequena probabilidade.

Em contrapartida, ele possui uma série de vantagens sobre os outros métodos, dentre elas [38]:

- Extremamente eficiente, pois a computação tem seu foco em regiões de alta probabilidade.
- Mais fácil de implementar do que as abordagens de Localização de Markov alternativas.
- Robusto a ruídos e a mudanças que afetem a disponibilidade de recursos computacionais.
- Capacidade de se adaptar aos recursos. Adaptando o número de amostras, a qualidade da solução depende dos recursos computacionais disponíveis. Ao trocar o processador do robô por um mais rápido, Monte Carlo pode utilizar estes recursos adicionais sem alterações no código fonte.

3.4 Mapeamento

Como já mencionado anteriormente, para que um robô possa se locomover de forma autônoma em ambientes não estruturados é necessário que ele saiba a cada instante sua localização. Isso pode ser alcançado utilizando os métodos descritos na seção anterior. Porém, esses métodos requerem a existência de um mapa do ambiente, que pode ser difícil de se construir manualmente caso não haja um disponível. Nessa seção será exposto um método de mapeamento, a partir dos dados providos pelos sensores do robô.

O processo de mapeamento de ambientes pode ser dividido em duas tarefas: síntese do mapa e exploração. A síntese do mapa é uma tarefa passiva do ponto de vista do controle. Ele consiste em dadas as observações do ambiente, construir uma representação do mesmo. A exploração é um processo ativo que visa controlar o robô de forma que ele "visualize"com seus sensores todas as partes do ambiente.

Os mapas gerados por métodos de mapeamento automático possuem várias vantagens sobre a confecção manual e sobre a aquisições por modelos CAD ou plantas de projetos arquitetônicos:

- ***Mudanças no ambiente:*** O ambiente do robô muda com o tempo. Deste modo, torna-se difícil manter um mapa preciso manualmente.
- ***Relacionando o Mapa aos Sensores:*** É difícil para um usuário predizer quais características do mundo serão facilmente reconhecíveis pelos sensores do robô. Quando é utilizado um sensor de varredura de distâncias baseado em raio laser, dependendo da altura em que ele for instalado no robô, o mapa que representa as características identificáveis por ele poderá ser diferente para cada altura. Um mapa provido pelo usuário pode então ser de pouco valor.
- ***Nível de Detalhe:*** Para alguns propósitos, o nível de detalhe requerido pode ser mais alto do que o que pode ser obtido por meio de um projeto arquitetônico.

3.4.1 Tipos de Mapas

Pesquisas recentes produziram dois paradigmas fundamentais para a modelagem de ambientes fechados para robôs: o métrico *(grid-based)* e o topológico. Abordagens métricas, como as propostas por [39] entre outros, representam ambientes por malhas de células igualmente espaçadas entre si. Cada célula pode indicar a presença de um obstáculo na região correspondente no ambiente. Abordagens topológicas, representam os ambiente por grafos. Os nós nestes grafos correspondem a situações, lugares ou marcos distintos. Eles são conectados por arcos se existir um caminho direto entre eles [40].

Ambas abordagens possuem pontos fortes e fracos.

- As métricas permitem uma fácil construção e manutenção de mapas em ambientes amplos. Elas eliminam as ambigüidades de locais distintos baseando-se na posição geométrica do robô. A posição do robô é estimada incrementalmente, baseando-se nas informações odométricas providas pelas leituras dos sensores. Desse modo, as abordagens métricas normalmente usam um número ilimitado de leituras de sensores para determinar a posição do robô. Assumindo que a posição do robô pode ser rastreada precisamente, diferentes posições, para as quais as medidas dos sensores se parecem, têm suas ambigüidades naturalmente eliminadas. Locais fisicamente

próximos são reconhecidos como tal, até mesmo quando as medidas dos sensores diferem (que é o caso dos ambientes dinâmicos onde, por exemplo, humanos podem bloquear os sensores).

- Abordagens topológicas determinam a posição do robô relativa ao modelo principalmente baseando-se em marcos. Por exemplo, se o robô atravessa dois lugares que se parecem, abordagens topológicas freqüentemente tem dificuldade em determinar se estes locais são distintos ou não. Além disso, como a entrada sensorial normalmente depende fortemente do ponto de vista do robô, as abordagens topológicas tendem a falhar no reconhecimento de locais geometricamente próximos, mesmo em ambientes estáticos, tornando difícil a construção de mapas grandes, particularmente se a informação sensorial é altamente ambígua.

Entretanto, as abordagens métricas sofrem por sua enorme complexidade de espaço e tempo. A resolução da malha devem ser fina o suficiente para capturar cada detalhe importante do mundo. A vantagem chave da representação topológica é a sua capacidade de compactação. A resolução dos mapas topológicos correspondem diretamente a complexidade do ambiente. A capacidade de compactação dos mapas topológicos os dá três vantagens chave sobre as métricas [38]:

1. permitem realizar um planejamento rápido;
2. facilitam a interface com planejadores simbólicos;
3. provêm interfaces mais naturais para instruções humanas (EX: "Vá para a sala A").

Já que as abordagens topológicas normalmente não requerem a exata determinação da posição geométrica do robô, elas freqüentemente se recuperam melhor de erros odométricos (erros que devem ser constantemente monitorados e compensados em abordagens métricas).

[34] classificou os tipos de mapas da seguinte maneira:

- ***Mapas de Locais Reconhecíveis:*** O mapa consiste em uma lista de locais que podem ser reconhecidos pelo robô de forma confiável. Nenhuma relação geométrica pode ser recuperada.
- ***Mapas Topológicos:*** Além dos locais reconhecíveis, o mapa guarda quais locais são conectados por caminhos atravessáveis. Conectividade entre locais visitados pode ser recuperada.
- ***Mapas Métrico-Topológicos:*** Este termo é usado para mapas nos quais distância e ângulo são adicionados à descrição do caminho. Informações métricas sobre os caminhos podem ser recuperadas.

- ***Mapas Totalmente Métricos:*** Os locais dos objetos são especificados em um sistema de coordenadas fixo. Informações métricas podem ser recuperadas sobre quaisquer objetos no mapa.

3.4.2 Localização *Grid*

A localização *grid* aproxima o subseqüente utilizando um filtro de histograma através da decomposição da apresentação do espaço. O filtro discreto de Bayes mantém como subseqüente uma coleção de valores de probabilidade discreta

$$Bel(_t) = p_{k,t} \tag{3.3}$$

onde cada probabilidade $p_{k,t}$ é definida através da célula do *grid* x_k. O conjunto de todas as células do *grid* formam a porção do espaço de todas as apresentações legítimas:

$$range(X_t) = x_{1,t}Ux_{2,t}U...x_{k,t} \tag{3.4}$$

Na versão mais básica da localização *grid*, a partição do espaço de todas apresentações é variável em relação ao tempo, e cada célula do *grid* é do mesmo tamanho. A localização *grif* é bem idêntica ao filtro binário de Bayes da qual é derivada. A tabela a seguir provê um pseudo-código para a implementação mais básica.

```
1: (pk,t−1, ut, zt,m)
2: para todo k faça
3:         pk,t=Pi pi,t−1 motion model(mean (Xk),ut,mean(Xi))
4:         pk,t = n measurement model(Xt, mean(Xk),m)
5: endfor
6: retorne pk,t
```

Figura 3.4: Algoritmo de Localização *Grid* [6]

Esse algoritmo requer como entrada os valores de probabilidade discreta $p_{k,t-1}$,ao longo da mais recente medição, controle e mapa. as funções **motion_model** e **measurement_model** podem ser implementadas utilizando qualquer técnica disponível. Esse algoritmo assume que cada célula possui o mesmo volume.

A figura 3.5 ilustra a localização *grid* em um exemplo unidimensional. Esse diagrama é equivalente ao do filtro de Bayes, exceto pela natureza da representação. O robô inicia com uma incerteza global, representada por um histograma uniforme. De acordo com o que é capitado pelos sensores, a célula do *grid* correspondente assume seu valor de probabilidade. O exemplo ilustra a habilidade para representar distribuições multi-modais com a localização *grid*.

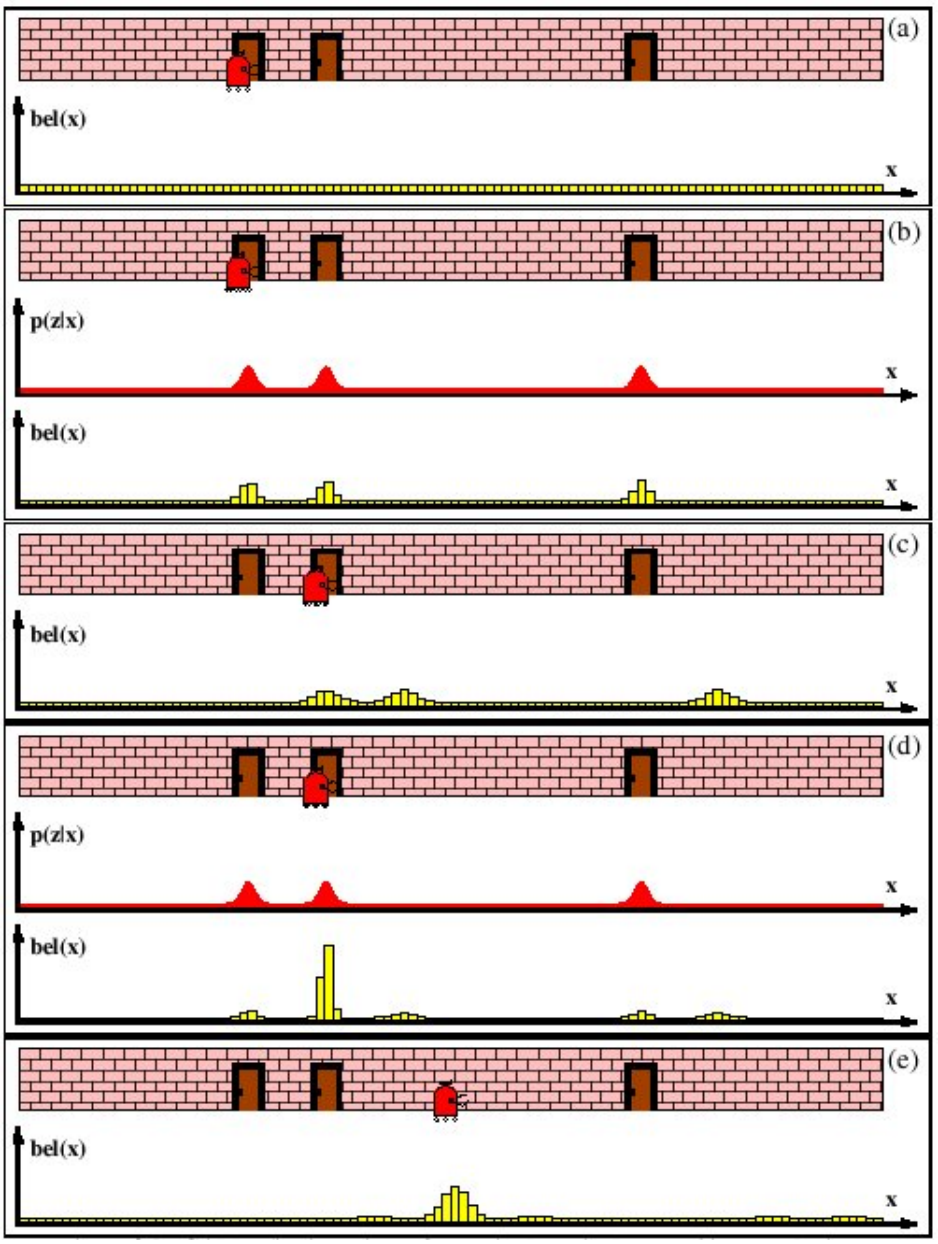

Figura 3.5: Localização *Grid* utilizando decomposição métrica de fina escala. Cada figura descreve a posição do robô ao longo de sua crença $bel(x_t)$ representado por um histograma em um *grid* [6]

3.4.3 *Occupancy Grid*

Nessa seção, é descrito um método para o mapeamento de ambientes que usa uma representação probabilística e reticulada da informação espacial chamada *Occupancy Grid*. Esse método trata do problema da construção de mapas consistentes a partir de sensores que apresentam ruído e outras incertezas.

Occupancy Grid é um campo aleatório multidimensional que fornece estimativas dos estados das células em um espaço reticulado. A representação dos mapas no *Occupancy Grid* é uma matriz de ocupação multidimensional (tipicamente 2D ou 3D) que mapeia o espaço em células, onde cada célula armazena uma estimativa probabilística de

seu estado. Os estados possíveis são ocupado ou livre [41].

As estimativas do estado das células são obtidas por meio da interpretação das leituras de distâncias usando modelos probabilísticos dos sensores. Um procedimento de estimativa Bayesiano permite a atualização incremental do *Occupancy Grid* usando leituras realizadas por vários sensores sobre múltiplos pontos de vista.

A variável de estado associada a uma célula $m_{<x,y>}$ do *Occupancy Grid* é definida como uma variável aleatória discreta com dois estados, ocupada e vazia, denotadas OCC e EMP [5]. Dessa forma, o *Occupancy Grid* corresponde a um campo aleatório de estados discretos e binários. Uma vez que os estados das células são exclusivos e exaustivos, $P(m_{<x,y>} = OCC) + P(m_{<x,y>} = EMP) = 1$.

Para interpretar os dados sensoriais (medidas de distâncias), é usado um modelo probabilístico do sensor $P(m_{<x,y>}|o^{(t)}, s^{(t)})$ que define a probabilidade de uma célula $m_{<x,y>}$ estar ocupada dados a observação do sensor $o(t)$ e a posição do robô $s(t)$. Esse modelo é utilizado em um procedimento de estimativa Bayesiano para determinar as probabilidades de estado das células do *Occupancy Grid*.

Partindo da estimativa atual do estado da célula $b(^{t-1)}(m_{<x,y>})$ baseado nas observações anteriores e dada uma nova observação $o^{(t)}$, a estimativa atualizada é dada por:

$$b^{(t)}(m_{<x,y>}) = 1-(1+\frac{P(m_{<x,y>})|o^{(t)}, s^{(t)})}{1-P(m_{<x,y>})|o^{(t)}, s^{(t)}}\frac{1-P(m_{<x,y>})}{P(m_{<x,y>})}\frac{b^{(t-1)}(m_{<x,y>})}{1-b^{(t-1)}(m_{<x,y>})})^{-1} \quad (3.5)$$

Nessa formulação recursiva, a estimativa anterior do estado da célula, $b(^{t-1)}m_{<x,y>}$,serve como priori e é obtida diretamente do *Occupancy Grid*. Essa nova estimativa é então armazenada no mapa.

Esse procedimento é realizado para cada célula que esteja no alcance perceptual do sensor. [5] utiliza um sensor (SICK PLS) que faz uso de um raio laser para realizar medições. Como o raio laser é linear, todas as células são atualizadas entre o ponto onde o raio encontrou um obstáculo até o ponto onde está o robô utilizando o algoritmo de conversão matricial de segmentos de reta de Bresenham [42] para determinar quais as células atualizar. A figura 3.6 descreve a atualização do mapa referente a um único raio laser. Note que as células que o raio laser intercepta são atualizadas. As células em branco possuem alta probabilidade de estarem ocupadas e as em preto possuem alta probabilidade de estarem vazias. Note que, segundo o modelo probabilístico do sensor, as células mais próximas do ponto onde o raio laser identificou um obstáculo possuem probabilidades mais altas de ocupação do que as próximas ao robô.

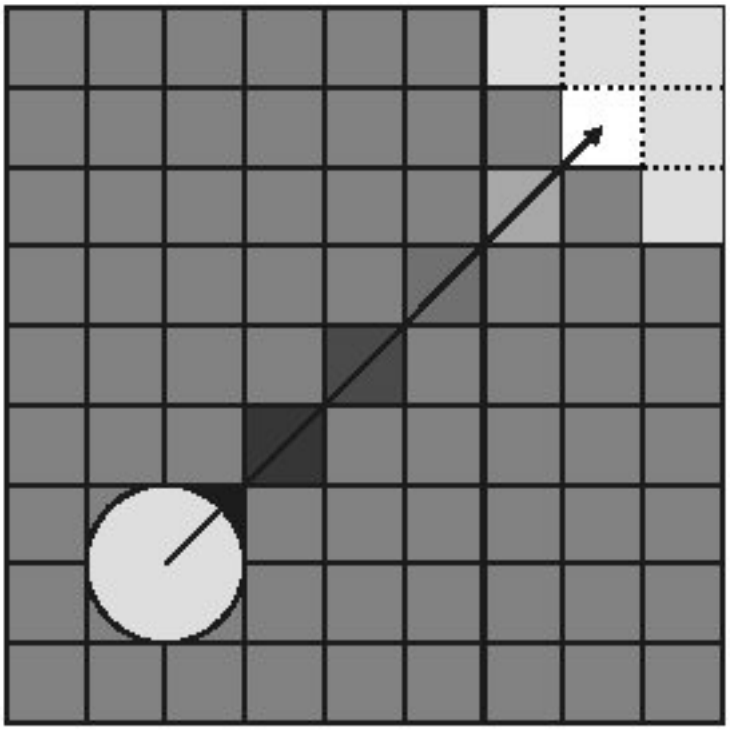

Figura 3.6: Atualização do mapa por um dos raios projetados pelo sensor. O círculo representa o robô. A seta indica o raio. A parte em cinza claro no canto superior direito do mapa é um obstáculo[5]

3.5 Considerações Finais

Neste capítulo, foram apresentadas as principais questões relacionadas a categoria de algoritmos de localização e mapeamento probabilístico. Foi mostrado também, uma pequena descrição do Algoritmo de Monte Carlo [7], além do mapeamento através de *Occupancy Grid*.

Capítulo

4

Os Algoritmos de Localização e Mapeamento

4.1 Considerações Iniciais

Este capítulo aborda os principais aspectos do SLAM (*Simultaneous Localization and Mapping*), além de resultados experimentais desta técnica realizados por [1] na *University of Southern California*. Também é feita uma breve descrição do mapeamento por *scan matching* [16].

4.2 *Scan Matching*

Existem vários modelos de sensores de distância na literatura que efetuam a correlação entre os dados gerados pelas medidas e o mapa. Uma técnica comum é conhecida como *scan matching* ou *map matching*. Essa técnica consiste na habilidade de transformar dados de leitura de dispositivos, tais como lasers, em mapas *occupancy grid*. Tipicamente, o *scan matching* compila pequenos números de escaneamentos consecutivos em mapas locais denominados *mlocal* [6]. A figura 4.1 ilustra tal mapa local. O modelo compara este mapa, *mlocal*, com um mapa global m armazenado previamente em memória. Isso ocorre de tal modo que quanto maior a similaridade entre os dois mapas, maior é a probabilidade atribuída ao mapa local, possibilitando uma construção mais precisa do mapa final.

O *scan matching* apresenta características favoráveis: assim como modelos probabilísticos, ele é fácil de computar, embora não produza probabilidades homogêneas na posição do parâmetro x_t. Uma maneira de aproximar essa probabilidade e se obter homogeneidade é utilizar um *kernel* Gaussiano homgêneo e executar o *scan matching* a partir

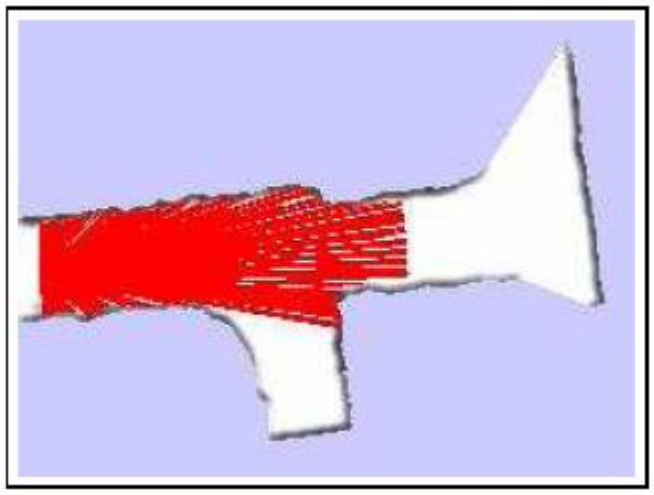

Figura 4.1: Exemplo de mapa local gerado por 10 leituras consecutivas, uma delas é mostrada [6]

dos dados gerados pelo *kernel*.

Uma vantagem chave do *scan matching* sobre modelos probabilísticos comuns, é que ele considera explicitamente os espaços livres, enquanto os modelos probabilísticos apenas consideram o ponto final da leitura, tratada como espaço ocupado, ou então ruído.

4.3 SLAM

O problema mais fundamental em robótica, a localização e mapeamento simultâneo [6] é tratado nesta seção. Comumente abreviado por SLAM - *Simultaneous Localization and Mapping*, o problema é também conhecido como *Concurrent Mapping and Localization* CML.

O problema SLAM surge quando o robô não tem acesso ao mapa do ambiente, ou não tem acesso a sua posição. Ao invés disso, tudo que é fornecido a ele são medidas $z_{1:t}$ e controles $u_{1:t}$. O termo localização e mapeamento simultâneo descreve o problema resultante: No SLAM, o robô adquire o mapa do ambiente enquanto simultaneamente se localiza em relação a esse mapa. SLAM é significativamente mais difícil que todos problemas de robótica discutidos, pois o mapa é desconhecido e precisa ser estimado ao longo do caminho, as posições são desconhecidas e também precisam ser estimadas.

De uma perspectiva probabilística, existem duas formas principais do problema SLAM, que são de igual importância prática. Uma é conhecida como *online* SLAM: Envolve estimação subseqüente através da posição momentânea ao longo do mapa:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{4.1}$$

Aqui x_t é a posição no tempo t, m é o mapa e $z_{1:t}$ e $u_{1:t}$ são medições e controles respectivamente. Esse problema é chamado *online* SLAM desde que envolva a estimação

de variáveis que persistem no tempo t. Muitos algoritmos para o problema *online* SLAM são incrementais: Eles descartam as medições e controles passados assim que eles foram processados.

O segundo problema SLAM é chamado de problema SLAM completo. No SLAM completo, procura-se calcular um subseqüente através de todo caminho $x_{1:t}$ ao logo do mapa, ao invés de apenas a posição corrente x_t:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{4.2}$$

Essa diferença sutil na formulação do SLAM entre *online* e completo tem ramificações no tipo de algoritmos que podem ser implementados. Em particular, o SLAM *online* é o resultado da integração das posições passadas do SLAM completo [1].

Uma segunda característica chave do problema SLAM tem a ver com a natureza do problema de estimação. O SLAM possui uma componente contínua e discreta. O problema de estimação contínua engloba as variáveis de localização dos objetos no mapa e de localização do próprio robô. Objetos podem ser *landmarks*, ou objetos detectados pelos sensores. A natureza discreta tem a ver com correspondência: Quando um objeto é detectado, um algoritmo SLAM deve estipular a relação desse objetos com os objetos detectados previamente. Essa relação é tipicamente discreta: Ou o objeto é o mesmo que o anterior, ou não é.

Nas duas versões do problema SLAM, *online* e completo, a estimação do subseqüente é o padrão ouro do SLAM. A captura completa do subseqüente deve ter conhecimento sobre o mapa e a posição, ou o caminho. Na prática calcular o subseqüente completo é infactível. Problemas surgem de duas fontes:

1. A alta dimensionalidade do espaço de parâmetro contínuo.

2. O grande número de variáveis discretas correspondentes.

Muitos algoritmos de estado-da-arte constroem mapas com milhares de características ou mais. Mesmo sob correspondência conhecida, o subseqüente dos mapas sozinhos envolvem distribuições de probabilidade com espaços de 10^5 ou mais dimensões. Esse é um contraste muito grande com problemas de localização, nos quais são estimados sobre espaços contínuos tri-dimensionais. Além disso, na maioria das aplicações, a correspondência é desconhecida. O número de possíveis arranjos ao vetor de todas variáveis correspondentes, $c_{1:t}$ cresce exponencialmente no tempo t. Por isso, algoritmos SLAM práticos que podem suportar o problema de correspondência devem se basear em aproximações.

4.4 Resultados Experimentais da Solução SLAM de [1]

Essa seção descreve os experimentos realizados pelo pesquisador Denis Wolf utilizando a solução SLAM descrita anteriormente. Os logs gerados como resultados nesses experimentos foram utilizados como dados de entrada para a execução da solução embarcada que será descrita posteriormente no capítulo 8.

Em [1] é proposto um algoritmo capaz de diferenciar partes estáticas e dinâmicas do ambiente e representá-las apropriadamente no mapa. A abordagem é baseada na manutenção de dois *occupancy grids*. Um modelo *grid* para a parte estática e outro para a parte dinâmica do ambiente. A união dos dois provêm a descrição completa do ambiente ao longo do tempo. É também mantido um mapa contendo informações sobre *landmarks* estáticas detectadas no ambiente. Essas *landmarks* possibilitam a localização do robô. Resultados em simulações e com robôs reais são capazes de mostrar a eficiência dessa abordagem.

4.4.1 O mapeamento

São utilizados dois mapas *occupancy grids* distintos (S e D). O mapa estático S contém informações apenas sobre a parte estática do ambiente, tais como muros e outros obstáculos que nunca foram observado se movendo. O mapa dinâmico D contém informações sobre objetos que foram observados se movendo pelo menos uma vez.

No mapa estático S, a probabilidade de ocupação de uma célula representa a probabilidade de uma entidade estática estar presente naquela célula. Como as entidades dinâmicas não estão representadas nesse mapa, se uma célula estiver ocupada por um objeto móvel, a probabilidade de ocupação irá indicar espaço vazio (não ocupado por entidade estática). Do mesmo modo, partes estáticas do ambiente não são representadas no mapa dinâmico D. Assim, quando uma célula em D tem a probabilidade de ocupação indicando espaço livre, isso significa simplesmente que não há uma entidade móvel ocupando a célula nesse momento. Mas, isso não exclui a possibilidade de a célula estar ocupada por uma parte estática do ambiente. Com essa abordagem é possível identificar objetos móveis, mesmo que eles se movam fora da vista do robô em uma área já mapeada.

A ocupação probabilística de cada célula *grid* pode ser estimada baseada na posição do robô e sua medição com o sensor. Sejam $p(S^t_{x,y})$ e $p(D^t_{x,y})$ denotações da probabilidade de ocupação da célula *grid* com coordenadas $< x, y >$ no mapa estático S e dinâmico D, respectivamente. O conjunto de leituras dos sensores é representado por o e a posição do robô é representada por u. Foi utilizada uma discretização em relação ao tempo, onde o^t é a leitura do sensor no tempo t. Assim, o problema consiste em se estimar:

$$p(S^t_{x,y}, D^t_{x,y}|o^1...o^t, y^1...u^t) \tag{4.3}$$

A posição do robô u é apenas utilizada para calcular em qual região do mapa (célula *grid*) será atualizada. Assim, para simplificar, a posição do robô u e a coordenada da célula *grid* ($< x, y >$) não serão incluídas nas equações seguintes.

O primeiro passo para se atualizar corretamente tanto o mapa S quanto D é diferenciar as entidades estáticas das dinâmicas no ambiente. Isso pode ser feito somando-se informações anteriores sobre as partes estáticas do ambiente à equação 4.3. Essa informação permite separar as leituras dos sensores gerados por obstáculos estáticos e dinâmicos, e atualizar corretamente os dois mapas. Essa quantidade pode ser estimada por:

$$p(S^t, D^t|o^1...o^t, S^{t-1}) \tag{4.4}$$

Como a informação de ocupação de S e D são mutuamente exclusivas (uma entidade não pode ser parte dos mapas estáticos e dinâmicos ao mesmo tempo), é possível escrever a equação 4.4 como um conjunto de duas equações:

$$p(S^t|o^1...o^t, S^{t-1}) \tag{4.5}$$

$$p(D^t|o^1...o^t, S^{t-1}) \tag{4.6}$$

O foco do problema é estimar cada quantidade acima, para atualizar os mapas dinâmico e estático.

Atualização do Mapa Estático

A atualização da equação para o mapa estático S (equação 4.5) é diferente da técnica de *occupancy grid* comum, o qual assume que o ambiente não muda ao longo do tempo. É utilizado o conhecimento anterior sobre o ambiente e compara-se com a observação atual para se manter apenas as partes estáticas do ambiente no mapa S.

Como mostrado em [43], a equação 4.5 pode ser reescrita como:

$$\frac{p(S^t|o^1...o^t, S^{t-1})}{1 - p(S^t|o^1...o^t, S^{t-1})} = \frac{p(S^t|o^t, S^{t-1})}{1 - p(S^t|o^t, S^{t-1})} \cdot \frac{1 - p(S)}{p(S)} \cdot \frac{p(S^{t-1})}{1 - p(S^{t-1})} \tag{4.7}$$

A equação 4.7 dá a fórmula recursiva para atualizar o mapa estático S. O termo $p(S)$ é o prior para ocupação. Se ele é setado para 0.5 (incerteza), ele pode ser cancelado. A ocupação para o mapa estático $p(S^t)$ é agora calculada baseada na informação anterior sobre este mapa $p(S^{t-1})$ e no modelo inverso do sensor $p(S^t|o^t, S^{t-1})$.

Nota-se que a informação sobre a ocupação anterior também é parte do modelo

inverso do sensor. Essa informação permite determinar se algum espaço previamente livre está agora ocupado, o que significa que uma entidade dinâmica se moveu para esse lugar. Também é possível detectar se alguma entidade que era previamente considerada estática se moveu. A tabela 4.1 mostra as possíveis entradas para o modelo inverso do sensor e os valores resultantes.

S^{t-1}	o^t	$p(S^t \mid S^{t-1}, o^t)$
Livre	**Livre**	**Baixa**
Desconhecida	**Livre**	**Baixa**
Ocupada	**Livre**	**Baixa**
Livre	**Ocupada**	**Baixa**
Desconhecida	**Ocupada**	**Alta**
Ocupada	**Ocupada**	**Alta**

Tabela 4.1: Modelo de Observação Inversa para o Mapa Estático [13]

A primeira coluna representa os possíveis estados de ocupação das células no mapa estático anterior S^{t-1}. Os estados possíveis são: *Livre*, *Desconhecida* e *Ocupada*. Para ser considerada *Livre*, a probabilidade de ocupação de uma célula *grid* deve estar abaixo de um valor mínimo pré-determinado. Se a probabilidade de ocupação está acima de um valor máximo estipulado (0.9 no experimento) a célula é considerada *Ocupada*. Valores entre 0.1 e 0.9 a célula é considerada *Desconhecida*.

A segunda coluna segunda coluna o^t representa a informação capturada pelos sensores. Nesse caso cada célula pode estar *Livre* ou *Ocupada* de acordo com a leitura do sensor na posição corrente do robô.

Os valores resultantes da observação inversa na terceira coluna são representados simplesmente por valores *Alta* ou *Baixa*. Valores altos são aqueles acima de 0.5 (que vão aumentar a probabilidade de ocupação da célula) e valores baixos são valores abaixo de 0.5 (que vão diminuir a probabilidade de ocupação da célula).

A tabela 4.1 mostra seis combinações possíveis. Nas primeiras três linhas da tabela o^t =*Livre*, que é o caso trivial onde não há obstáculos detectados pelo sensor. Independente das informações de ocupações anteriores, o modelo de observação inversa irá resultar em valores baixos, que irão diminuir a probabilidade de ocupação. Na quarta linha da tabela (S^t =*Livre* e o^t =*Ocupado*) é o caso onde há uma forte evidência que o espaço estava livre de entidade estática e agora está ocupado. Nesse caso considera-se a presença de um objeto dinâmico e por isso a probabilidade de ocupação no mapa estático diminui. Na quinta linha (S^t =*Desconhecido* e o^t =*Ocupado*) há uma incerteza em relação à ocupação anterior. Os objetos detectados são considerados inicialmente estáticos até que eles se movam. Então o modelo de sensor irá aumentar a probabilidade de ocupação no mapa estático. Esta situação ocorre quando o robô é inicializado e todas as células *grid* refletem incerteza sobre sua ocupação. Na última linha da tabela é o caso trivial

onde o sensor confirma a crença anterior de que a célula estava ocupada por um objeto estático. Isso acarreta em valor alto para a ocupação no mapa estático.

Atualização do Mapa Dinâmico

O mapa dinâmico D contém informações apenas sobre as partes móveis do ambiente. É denotado por $p(D^t)$ a probabilidade de ocupação de determinada região do mapa estar ocupada por um objeto móvel em um tempo t. Baseado na leitura dos sensores e informação anterior e sobre a ocupação no mapa estático S, é possível identificar as partes móveis do ambiente e representá-las no mapa dinâmico D.

Similar a equação 4.5 a equação 4.6 pode ser reescrita da seguinte maneira [43]:

$$\frac{p(D^t|o^1...o^t, S^{t-1})}{1-p(D^t|o^1...o^t, S^{t-1})} = \frac{p(D^t|o^t, S^{t-1})}{1-p(D^t|o^t, S^{t-1})} \cdot \frac{1-p(D)}{p(D)} \cdot \frac{p(D^{t-1})}{1-p(D^{t-1})} \tag{4.8}$$

A equação 4.8 é similar a equação 4.7 no sentido de que a nova estimação para a ocupação de $p(D^t)$ é baseada na ocupação anterior do mapa ($p(D^{t-1})$) e no modelo de sensor $p(D^t|o^t, S^{t-1})$. Para atualizar o mapa dinâmico D, a equação 4.8 também leva em consideração a informação sobre a ocupação anterior no mapa estático S em seu modelo de sensor.

É importante notar que dessa forma o modelo não fica guardando os estados anteriores do mapa dinâmico, somente interessa manter informações dos objetos dinâmicos atuais.

Similarmente à tabela 4.1, a tabela 4.2 mostra os valores do modelo de observação inversa utilizado para atualizar o mapa dinâmico.

S^{t-1}	o^t	$p(D^t\|S^{t-1}, o^t)$
Livre	**Livre**	**Baixa**
Desconhecida	**Livre**	**Baixa**
Ocupada	**Livre**	**Baixa**
Livre	**Ocupada**	**Alta**
Desconhecida	**Ocupada**	**Baixa**
Ocupada	**Ocupada**	**Baixa**

Tabela 4.2: Modelo de Observação Inversa para o Mapa Dinâmico [13]

É interessante observar que a probabilidade de ocupação no mapa dinâmico só aumenta na situação da quarta coluna (S^t =*Livre* e o^t =*Ocupado*). Isso caracteriza a presença de um objeto móvel ocupando uma região que estava vazia anteriormente.

A figura 4.2 mostra um exemplo de atualização do mapa. Na figura 4.2a, os espaços em preto no *grid* representam regiões ocupadas, os espaços em cinza são regiões desconhecidas e os espaços em branco regiões vazias. As três possibilidades são equivalentes as das tabelas 4.1 e 4.1 (S^{t-1}). A figura 4.2b é similar a segunda coluna das tabelas 4.1

e 4.2 (o^t). A figura 4.2c refere-se a coluna 3 da tabela 4.1 ($p(S^t|o^t, S^{t-1})$). E finalmente, a figura 4.2d representa o modelo de sensor inverso, que será aplicado para atualizar o mapa dinâmico D, coluna 3 da tabela 4.2 ($p(D^t|o^t, S^{t-1})$).

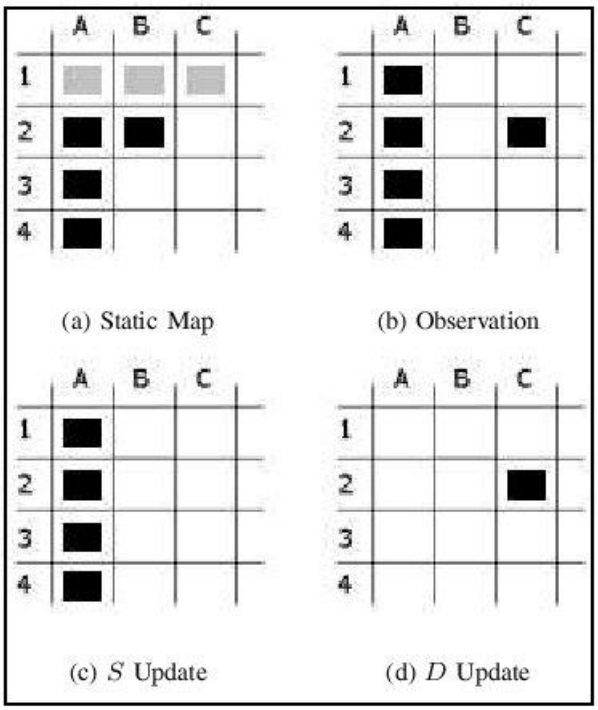

Figura 4.2: Atualização para mapas dinâmicos e estáticos [1]

Esse exemplo ilustra dois casos interessantes na atualização de mapas. No primeiro caso, a célula B2 estava ocupada (tempo $t-1$) mas a leitura do sensor indica um espaço livre no local (tempo t). Isso significa que um objeto parado foi mapeado como uma parte estática do ambiente. Como o objeto se moveu, o mapa estático foi corretamente atualizado. No segundo caso, a célula C2 estava livre (tempo $t-1$) mas a leitura do sensor indica que a região está ocupada (tempo t). Isso significa que um objeto móvel se locomoveu para esse espaço. A atualização aplicada para a célula no mapa estático (figura 4.2c) irá representá-la como espaço vazio pois o objeto móvel não será representado no mapa estático. Esse objeto será representado somente no mapa dinâmico como visto na figura 4.2d.

Para a localização foi utilizado a técnica baseada em *landmarks*, peças no ambiente que são identificadas como tais pelos sensores do robô. Se o robô tem informações a priori sobre a posição dos *landmarks*, ele pode estimar sua posição ao detectá-los.

Para validar as idéias apresentadas na seção 4.4 foram utilizados experimentos e simulações [1] [13]. Os testes experimentais foram feitos usando ActiveMedia Pionner Robots equipado com um sensor laser SICK e uma máquina Athlon XP 1600+. Player [44] foi utilizado para o controle de baixo nível dos robôs e o Player/Stage [45] foi utilizado para o experimento simulado.

As figuras 4.3 e 4.4 mostram os resultados dos experimentos, através da atualização

dos mapas ao longo do tempo.

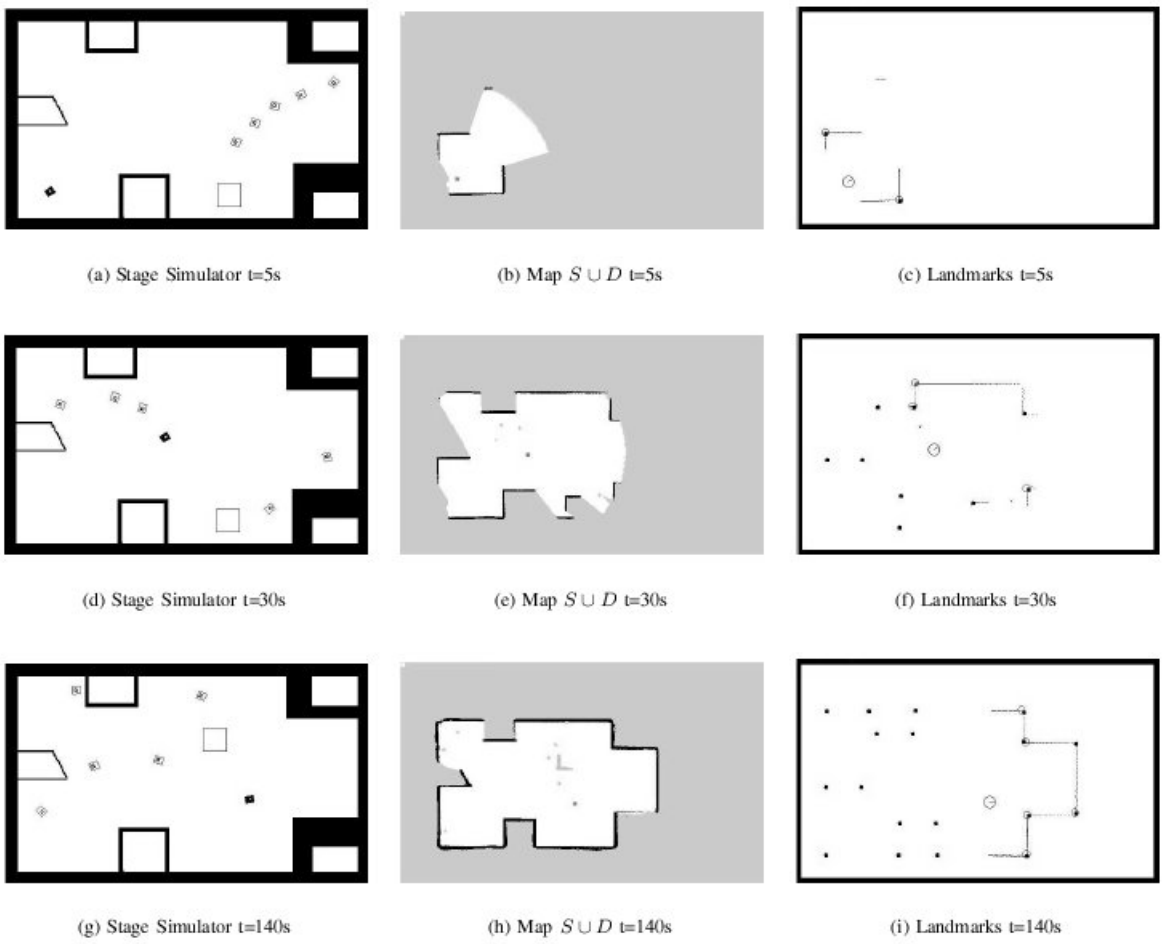

Figura 4.3: Simulação com 5 objetos móveis [1]

4.4.2 Experimento com Mapeamento Autônomo 3D em Ambientes Urbanos

Essa seção descreve um trabalho voltado à geração de mapas densos 3D para ambientes urbanos utilizando laser em uma plataforma móvel [8]. Esses mapas mostram tanto detalhes de fina escala (em escala de centímetros) quanto larga escala (mapas com 500m de cada lado). Será mostrado um algoritmo básico para mapeamento 3D e resultados preliminares coletados na USC (*University of Southern California*) utilizando-se um veículo Segway RMP.

Comparado ao mapeamento *indoor*, o problema de mapeamento em ambientes urbanos tem algumas características distintas. Primeiramente, a localização de escala grossa (com incertezas de poucos metros) é usualmente disponível através de GPS. Assim, problemas como determinar se o robô saiu e retornou a mesma posição, são resolvidos trivialmente. Por outro lado, em ambientes *indoor*, geralmente o robô permanece em

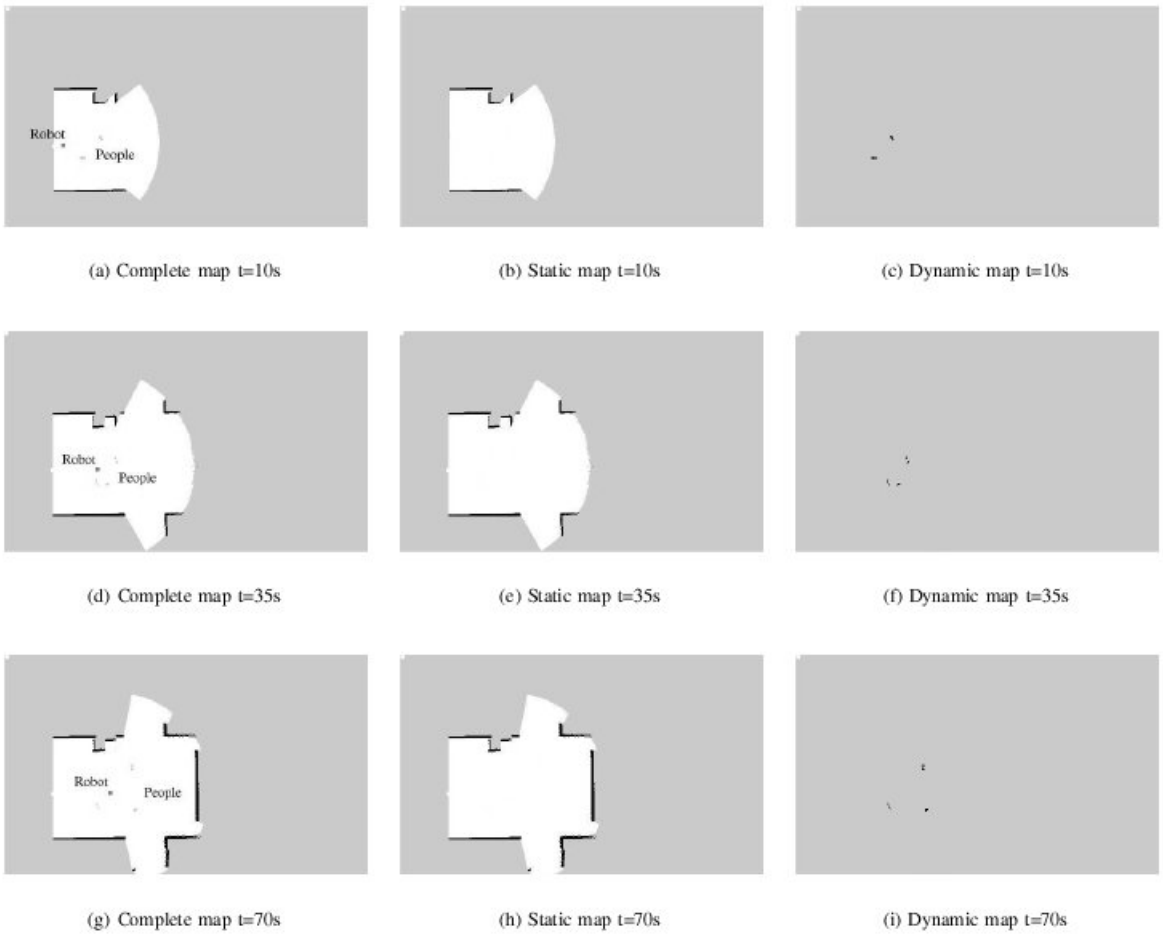

Figura 4.4: Mapas através do tempo [1]

alturas constantes, e em ambientes externos, é preciso levar em contas subidas e descidas. Por isso um mapeamento desse tipo precisa ser realizado em três dimensões.

No caso estudado a seguir, são assumidas duas considerações chaves: (1) a altitude do robô é constante, e (2) o ambiente é parcialmente estruturado (i.e. contém objetos construídos). O algoritmo de mapeamento básico possui quatro passos [8]:

1. **Localização fina:** odometria é combinada com IMU e o laser produz uma estimação incremental da posição do robô. Essa estimação é bem precisa, mas apresenta erros ilimitados de desvios ao longo da execução.

2. **Localização de escala grossa:** o GPS é utilizado para determinar uma posição aproximada do robô. Ao contrário da localização fina, essa estimação possui precisão de alguns metros, mas não sofre de erros cumulativos. Como uma alternativa ao GPS é introduzido um algoritmo Monte Carlo modificado, em princípio, para localizar o robô através de um rascunho de mapa obtido através de imagem aérea ou de satélite

3. **Localização de escala grossa para fina:** O desafio chave para mapeamento urbano repousa em misturar estimações de localização de escala grossa com fina de uma maneira para se manter tanto continuidade local quanto consistência global.

Para esse passo, características colhidas de múltiplas leituras são casadas e a trajetória do robô é otimizada para satisfazer restrições locais e globais.

4. **Geração do mapa:** utilizando a estimação de posição pelo passo 3, dados da leitura do laser são projetados em um espaço cartesiano 3D, formando uma nuvem de pontos estendida, figura 4.5. Partes do ambiente são caracterizadas pelo acúmulo de pontos e espaços abertos aparecem como a ausência de pontos.

Figura 4.5: Fotografia do campus da USC sua representação através de pontos gerados pelo algoritmo descrito em [8]

Segway RMP

O trabalho foi facilitado em grande parte pela utilização de um Segway RMP como plataforma de sensores (figra 4.6). O RMP é um veículo de duas rodas, rápido e suporta grandes cargas. No experimento o veículo foi configurado com um laser no plano horizontal e um no plano vertical. O horizontal é utilizado principalmente para localização de escala fina, enquanto que o laser vertical é usado para gerar o mapa 3D. O robô também é equipado com GPS e sensor IMU.

Resultados dos Experimentos em Ambiente Urbano [8]

Os experimentos realizados no campus da USC (*University of Southern California*) utilizando o algoritmo de mapeamento de ambientes urbanos com o robô Segway RMP geraram resultados interessantes que serão ilustrados a seguir.

Durante uma volta de aproximadamente 2km pelo campus da USC, o robô parte de determinada posição inicial e retorna a esta após o percurso (figura4.7). A trajetória do robô utilizando somente odometria, e odometria com leitura laser, mostra como os erros cumulativos gerados pela odometria pura causam um desvio de aproximadamente 2km.

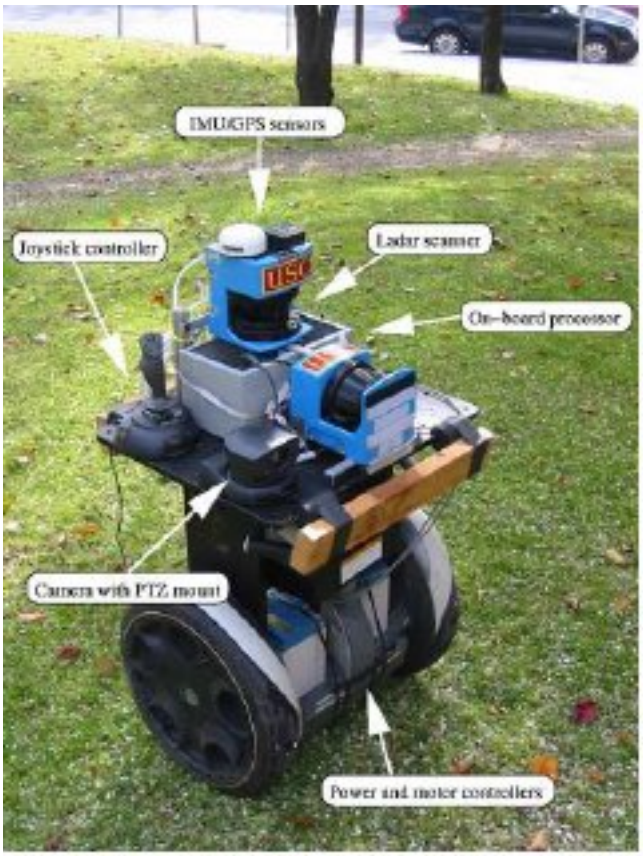

Figura 4.6: Robô RMP para mapeamento [8]

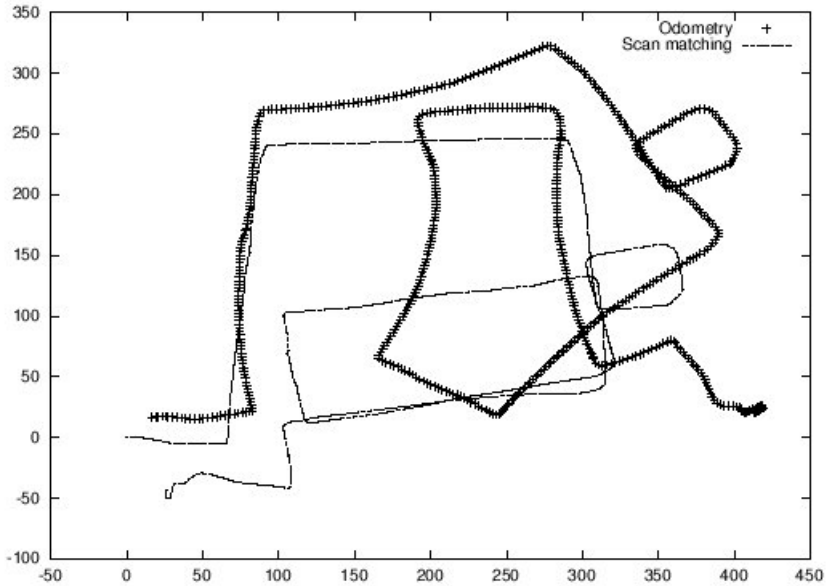

Figura 4.7: Odometria e odometria com laser [8]

O mesmo trajeto é ilustrado (figura 4.8) com a utilização de GPS e algoritmo de Monte Carlo. Nos locais onde os satélites são obstruídos por prédios altos, o algoritmo de Monte Carlo corrige a trajetória na ausência do GPS.

O algoritmo básico de Monte Carlo utiliza um filtro de partícula para manter a posição estimada do robô. Comparado a outros algoritmos Bayesianos de estimação

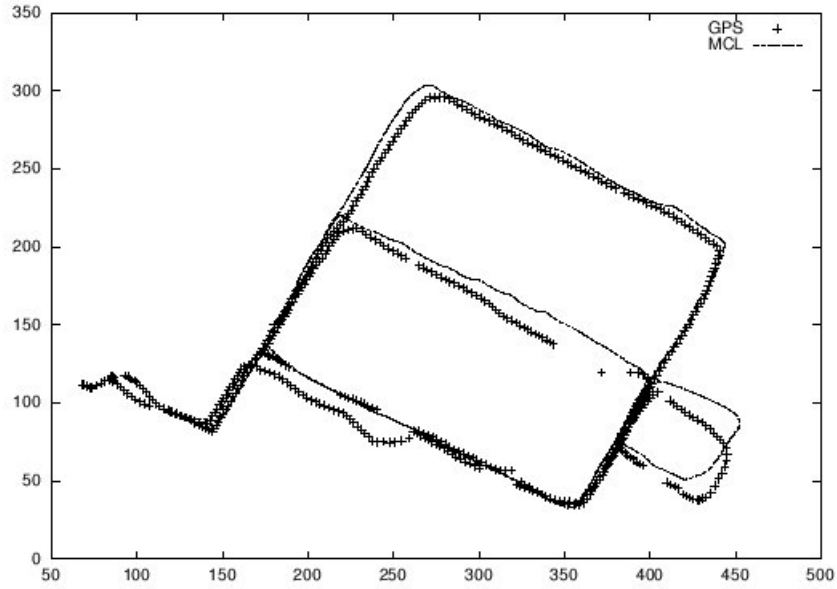

Figura 4.8: GPS e algoritmo de Monte Carlo [8]

(tal como filtros de Kalman), filtros de partículas tem a vantagem de serem aptos a representarem distribuições multi-modais. A figura 4.9 mostra o mapa do campus da USC, com o mapa utilizado pelo algoritmo de Monte Carlo, divido em regiões: Ocupadas em preto; Semi-ocupadas em cinza; E livres em branco.

Figura 4.9: Mapa utilizado pelo algoritmo MCL [8]

4.5 Considerações Finais

Neste capítulo foi apresentada a base teórica dos algoritmos de localização e mapeamento utilizados neste trabalho. Também foi abordado o problema SLAM e como os experimentos que geraram os arquivos de log utilizados neste trabalho foram conduzidos. Os capítulos 7 e 8 apresentam as implementações em C e embarcada respectivamente desses algoritmos até aqui estudados.

Capítulo

5

Computação Reconfigurável

5.1 Considerações Iniciais

Atualmente, são empregados dois principais métodos computacionais para execução de algoritmos: ASICSs e processadores programáveis.

ASICSs, acrônimo de *Application Specific Integrated Circuit*, são construídos especificamente para execução de uma tarefa, sendo portanto altamente eficientes. Entretanto, este nível de especialização resulta em um componente pouco flexível sendo que modificações em seu funcionamento são caras e em muitos casos inviáveis.

Em outro extremo existem os processadores programáveis. Os processadores executam uma série de instruções computacionais seqüencialmente a fim de processar e executar algoritmos.

Esse tipo de implementação resulta em um sistema altamente flexível, permitindo alterações dos algoritmos de forma simples rápida. Entretanto, o custo desta flexibilidade é o menor desempenho computacional devido a sobrecarga adicional para leitura, decodificação e processamento dos códigos de máquina.

A figura 5.1 lista as principais diferenças entre ASICs e processadores.

O desenvolvimento atual requer soluções flexíveis e com alto desempenho. Neste sentido a computação reconfigurável tem sido alvo de pesquisa e desenvolvimento tecnológico.

A computação reconfigurável permite a implementação de hardware dedicado e processamento de software, conforme a necessidade da aplicação.

Diversos fabricantes, como a Altera [10] e Xilinx [46], têm disponibilizado no mercado circuitos integrados com milhões de portas lógicas, permitindo a construção de complexos circuitos digitais e execução de tarefas com altíssimo desempenho.

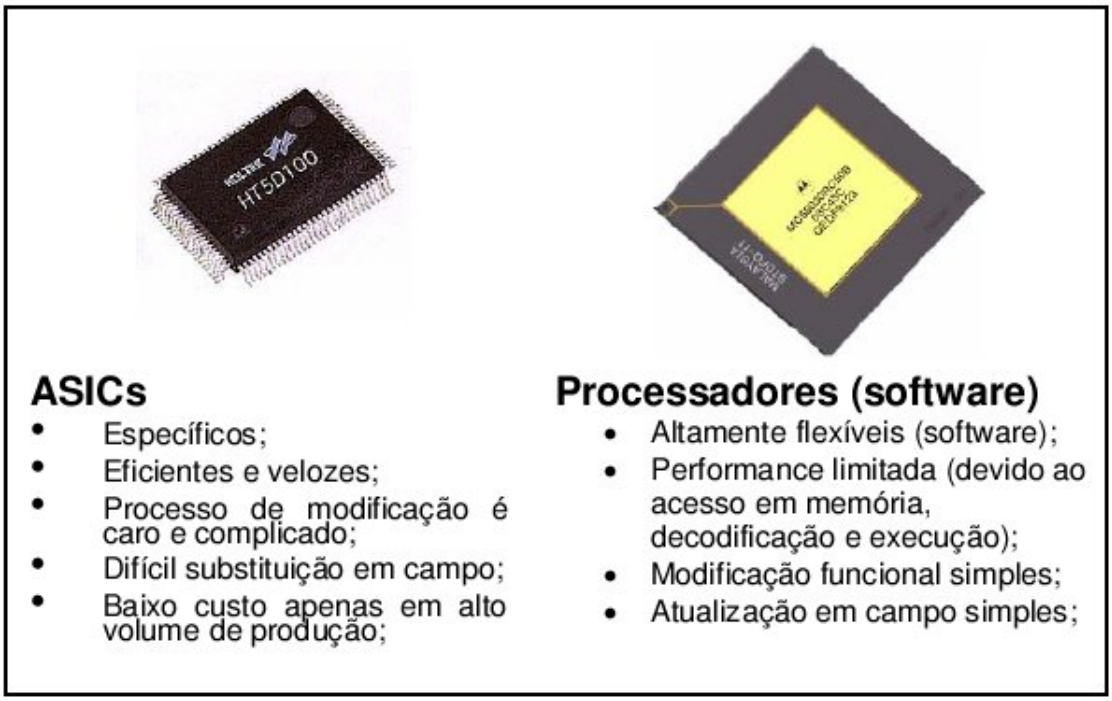

Figura 5.1: ASICs versus Processadores

5.1.1 Tecnologia Atualmente Empregada

Atualmente, a computação reconfigurável utiliza de dispositivos dotados de milhares ou pouco menos de milhões de unidades lógicas que podem ser interconectadas para formar unidades lógicas funcionais.

Arquiteturas empregadas nestes dispositivos usualmente integram um processador (capaz de executar instruções de software) e unidades de processamento em hardware.

Esta união resulta em sistemas suficiente flexíveis (devido ao emprego de software) e com grau de desempenho escalável, dependendo do número e função das unidades de processamento independente, implementadas diretamente no hardware [47].

5.2 Arquiteturas Fortemente Acopladas

A arquitetura mostrada na figura 5.2 disponibiliza unidades reconfiguráveis funcionais dentro de um processador central. Esta arquitetura permite um ambiente de programação tradicional com adição de instruções personalizadas e flutuantes, podendo ser modificadas no decorrer do processamento.

Nesta construção as unidades reconfiguráveis atuam de forma similar às unidades funcionais do processador, se comunicando através do barramento de dados e utilizando registradores para armazenamento das entradas e saídas das operações.

Uma segunda forma de utilizar as unidades reconfiguráveis, figura 5.3, é adicioná-las ao processador central como um co-processador.

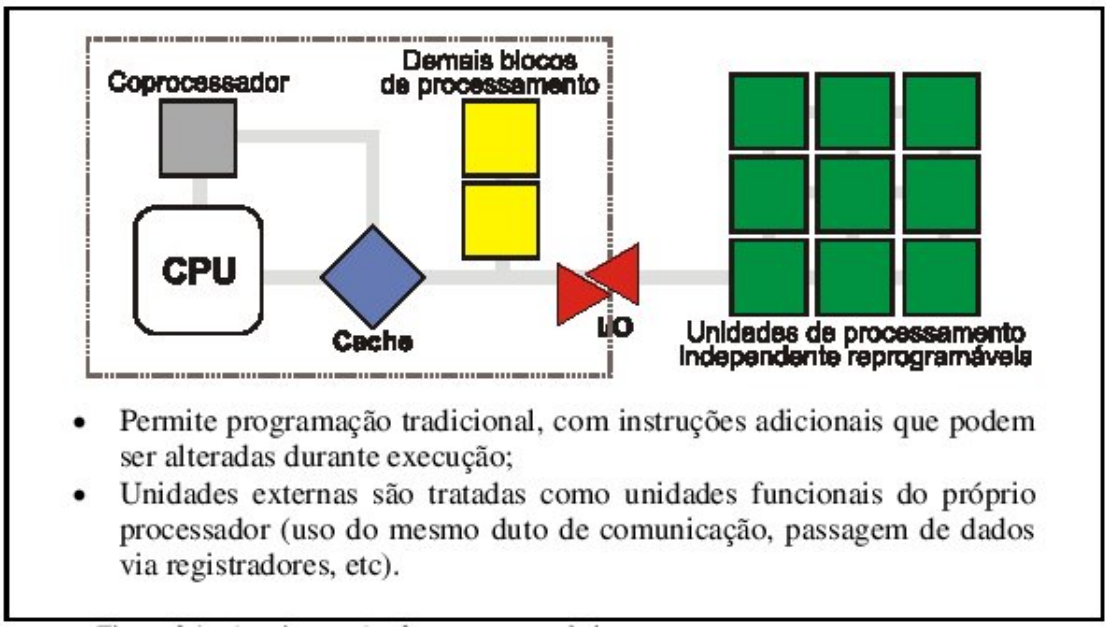

Figura 5.2: Arquitetura A - fortemente acopladas

Um co-processador é em geral uma unidade complexa de procesamento, capaz de executar operações computacionais sem a necessidades constante de envio e recepção de dados do processador. Isso permite que o co-processador execute tarefas computacionalmente intensas deforma paralela ao processador, inclusive com a capacidade de leitura e gravação direta à memória RAM do sistema.

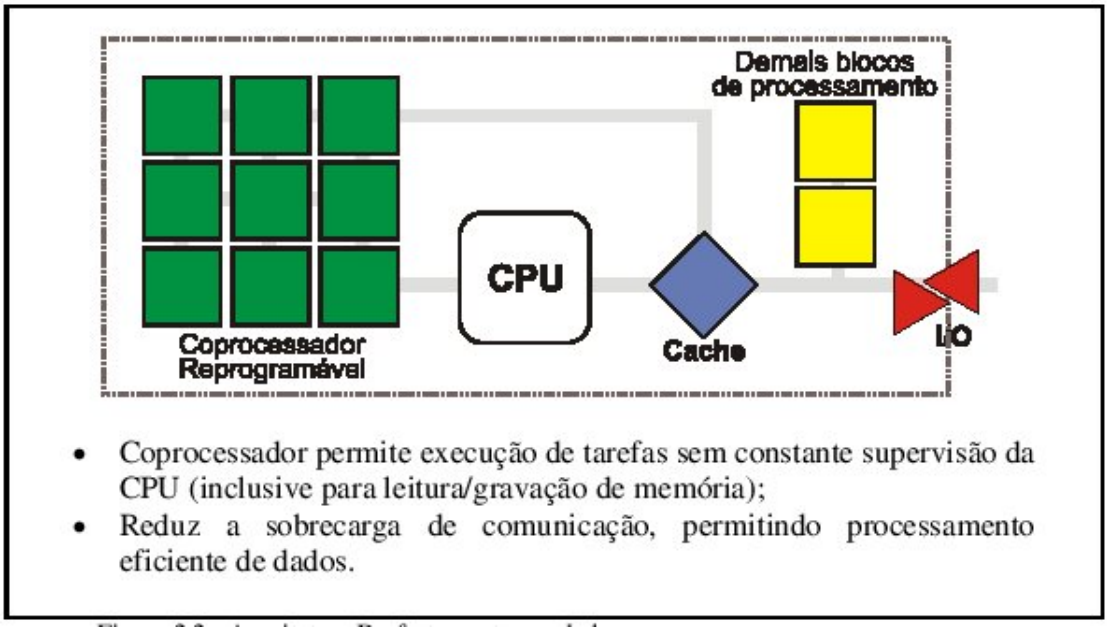

Figura 5.3: Arquitetura B - fortemente acopladas

Uma versão híbrida desta arquitetura implementa a unidade lógica reprogramável no barramento de cache. Durante a execução esta unidade pode funcionar como um cache regular ou realizar processamento adicional, caso desejado.

5.3 Arquiteturas Fracamente Acopladas

Arquiteturas fracamente acopladas utilizam as unidades lógicas reprogramáveis como unidades independentes de processamento.

A figura 5.4 ilustra a arquitetura que emprega as unidades reprogramáveis como unidades de processamento reprogramáveis. Esta arquitetura é pouco usual e é empregada apenas em aplicações específicas.

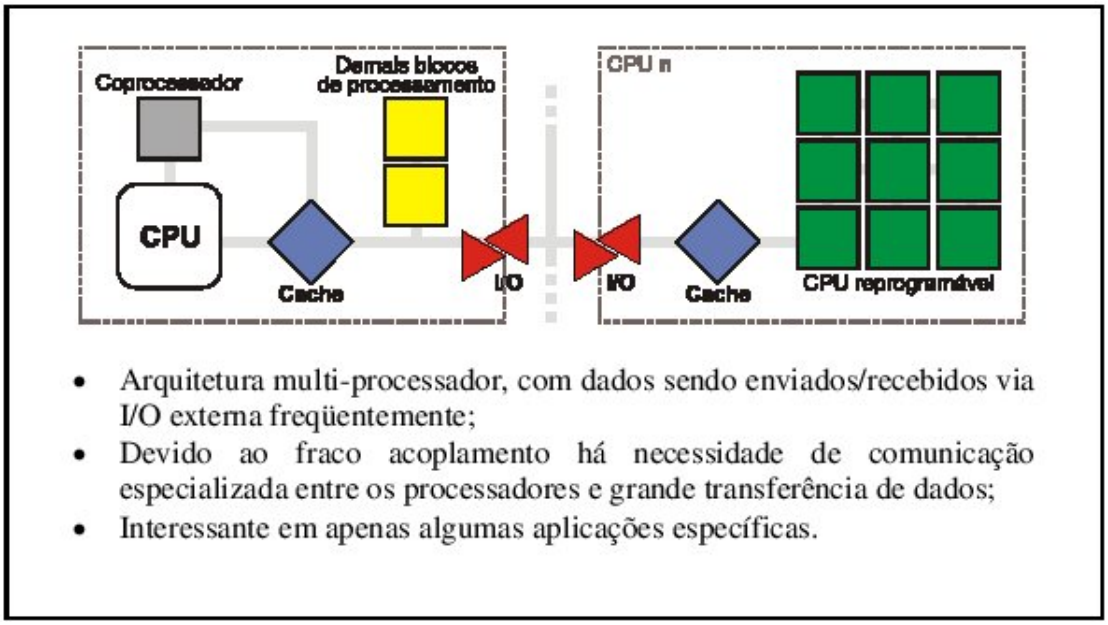

Figura 5.4: Arquitetura C - fracamente acopladas

A figura 5.5 ilustra uma arquitetura multi-processador onde todos os *cores* computacionais são independentes e se comunicam de forma assíncrona, por uma rede dedicada, para troca de dados [47]. Este tipo de configuração é comumente encontrada em sistemas distribuídos, onde vários processadores usuais são interconectados para a execução paralela de tarefas.

A principal desvantagem desta arquitetura é a execessiva carga de comunicação da rede e dificuldade para implementação de compiladores eficientes capazes de distribuir os algoritmos seqüenciais e reconfigurar as unidades lógicas reprogramáveis.

5.4 Aspectos de Software

Apesar do hardware reconfigurável ter apresentado muitos benefícios para algumas aplicações, ele precisa ser transparente ao programador. Isso requer um software como um ambiente de desenvolvimento que ajuda na configuração do hardware. Esse software pode variar de um assistente para criação manual até um sistema automático de projeto de circuito.

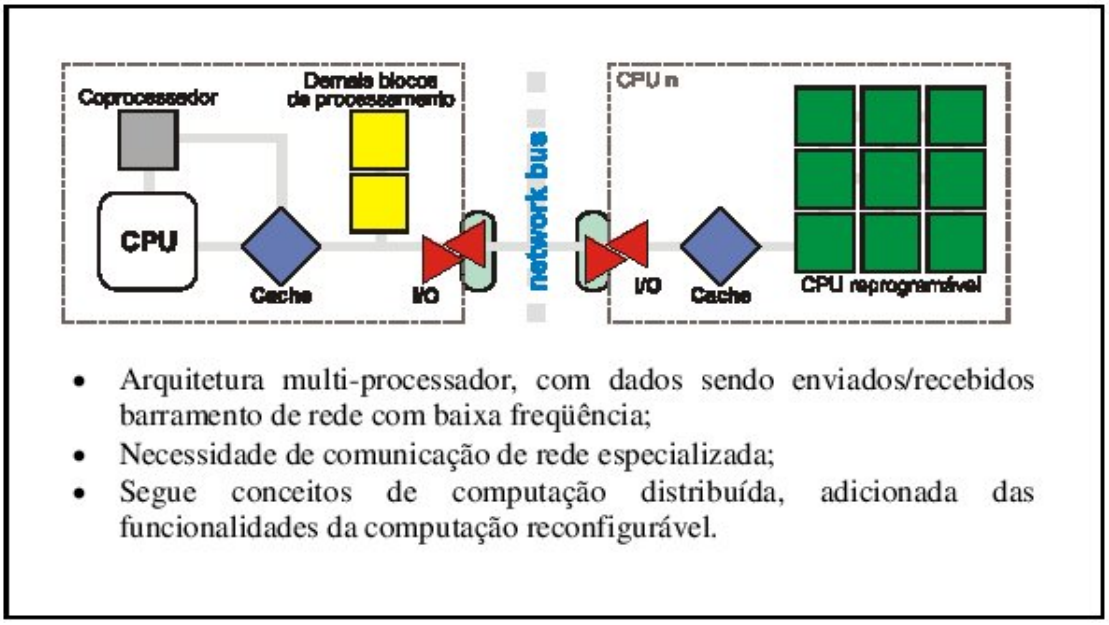

Figura 5.5: Arquitetura D - fracamente acopladas

A criação manual de circuito é um método poderoso para a criação de projetos de alta qualidade, entretanto, isso requer grande conhecimento no sistema reconfigurável empregado além de um tempo bem maior para o projeto. Por outro lado, um sistema de compilação automática provê uma maneira rápida e simples de programar sistemas reconfiguráveis, deixando o uso do hardware reconfigurável mais acessível para programadores de aplicações genéricas, mas a qualidade pode decair.

5.4.1 Fases de Projeto

Tanto para a criação de circuito manual ou automática, o processo de projeto é realizado em fases distintas [9], como mostra a figura 5.6 :

Os estágios em cinza representam o esforço manual por parte do programador, enquanto os estágios em branco são realizados automaticamente.

A descrição do circuito é o processo de descrever as funções que serão colocadas no hardware reconfigurável. Isso pode ser feito simplesmente através de um código em C que representa a funcionalidade do algoritmo a ser implementado em hardware. Em contrapartida, a descrição pode ser tão complexa como especificar os inputs e outputs e operações de cada bloco básico no hardware reconfigurável.

No caso das descrições em linguagens de alto nível HLL, tais como C/C++ e Java, o código deve ser compilado em uma netlist de componentes gate-level.

Uma vez detalhada a descrição em nível de elementos ou gate, essas estruturas devem ser traduzidas para os elementos lógicos presentes no hardware reconfigurável. Esse estágio é conhecido como technology mapping [9] e depende de cada arquitetura alvo específica.

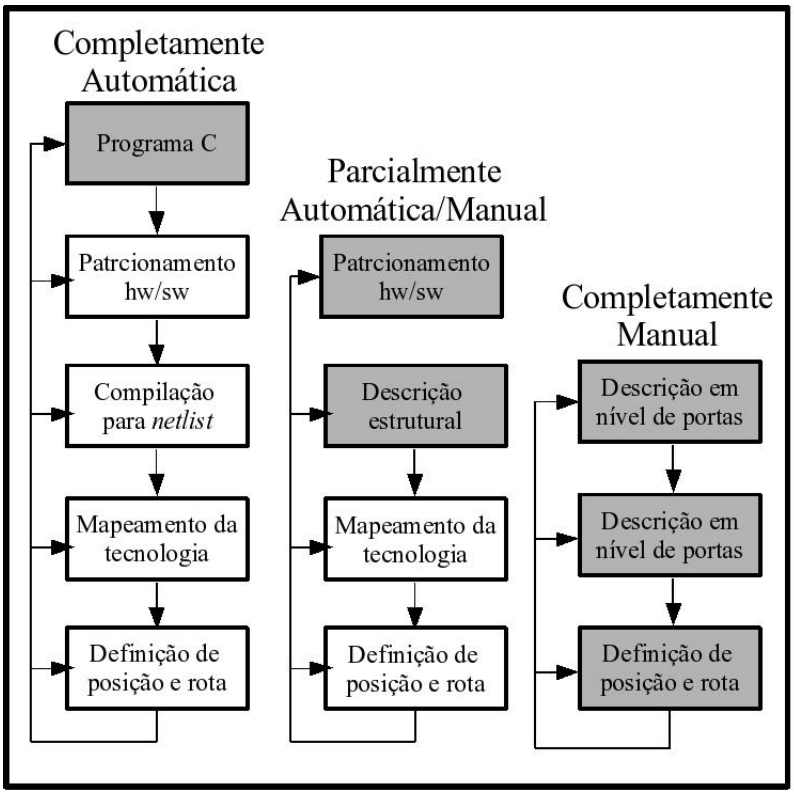

Figura 5.6: Fases de Projeto [9]

Para arquiteturas baseadas em LUT, essa fase consiste em particionar os circuito em pequenas subfunções que podem ser mapeadas em uma LUT simples [48] [49]. Algumas arquiteturas como as da série Xilinx 4000 [46], contém múltiplas LUTs por célula lógica. Essas LUTs podem ser utilizadas separadamente para gerar pequenas funções ou em conjunto para funções com inputs maiores [50]. Aproveitando a vantagem da utilização de múltiplas LUTs e roteamento interno, funções com mais entradas (inputs) que uma LUT simples, podem ser mapeadas na FPGA, conforme mostrado na figura 5.7 :

$$OUT = (A\overline{B}CD + ABCD + \overline{A}B\overline{C})E\overline{F} + (AB\overline{G} + A\overline{B}H)F$$

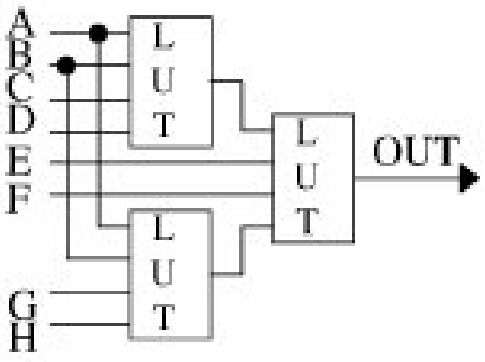

Figura 5.7: Função com entrada grande mapeada em uma célula mult-LUT na FPGA [9]

Após o mapeamento do circuito, os blocos resultantes devem ser mapeados no hardware reconfigurável. Cada bloco deve ser colocado em um local específico do hard-

ware, preferencialmente próximo a outros blocos com os quais ele irá se comunicar.

Em seguida, na fase de roteamento, os blocos que foram colocados no FPGA devem ser interligados. Cada sinal é designado a uma porção específica dos recursos de roteamento presentes no hardware reconfigurável. Isso pode se tornar muito complicado se na fase de colocação, os componentes que precisam se comunicar foram arranjados longe uns dos outros, pois se o sinal tiver que percorrer maiores distâncias, isso irá consumir mais recursos de roteamento. Torna-se necessário então que uma boa colocação (placement) seja realizada antes da fase de roteamento.

Essa fase é uma das que consome maior quantia de tempo no ciclo de desenvolvimento do circuito, portanto, antes de iniciá-la é pertinente que se verifique se o arranjo resultante da fase de colocação pode ser roteado antes de iniciar o roteamento. A verificação informa ao projetista se mudanças são necessárias no layout ou se uma maior estrutura reconfigurável é necessária [50].

5.4.2 Particionamento Hardware/Software

Para sistemas que incluem tanto hardware reconfigurável e microprocessador tradicional, o programa precisa ser particionado em seções para serem executadas no hardware reconfigurável e seções para serem executadas em software no microprocessador. Em geral, seqüências de controle complexas tais como laços de tamanho variável são mais eficientes se implementados em software, enquanto que operações com fluxos de execução constantes são mais eficientes se executadas em hardware.

A maioria dos compiladores para sistemas reconfiguráveis geram apenas a configuração em hardware do sistema, ao invés de hardware e software. Em muitos casos isso acontece porque o hardware reconfigurável não está acoplado com um microprocessador então somente a configuração do hardware é necessária.

Nos casos em que o hardware reconfigurável trabalha ao lado de um processador, alguns sistemas exigem que a compilação do hardware seja feita separada da do software, onde funções especiais são chamadas no software para configurar e controlar o hardware reconfigurável. A desvantagem dessa abordagem é que o projetista necessita identificar as funções que serão mapeadas em hardware e traduzi-las em chamadas especiais de hardware.

Para compiladores que gerenciam aspectos de projeto tanto para hardware quanto para software, o particionamento pode ser feito manualmente ou automaticamente pelo próprio compilador. Quando o mapeamento é feito pelo programador, diretivas do compilador são utilizadas para marcar seções do código do programa para a compilação do hardware. A linguagem NAPA C [51] fornece instruções para se especificar qual seção do código será executada no FIP (Fixed Instruction Processor) ou em hardware no ALP (Adaptative Logic Processor).

Alternativamente, o particionamento hardware/software pode ser feito automaticamente [52]. Nesse caso, o compilador utilizará funções de custo baseadas no ganho em aceleração através da execução de um fragmento de código em hardware para determinar onde o custo de configuração é superado pelos benefícios de execução em hardware.

5.5 Ferramentas para Computação Reconfigurável

Nesta seção será feita uma breve descrição de algumas ferramentas que trabalham com a tecnologia de FPGA utilizadas para o desenvolvimento do trabalho.

5.5.1 FPGA Stratix da Altera

Introduzida no mercado no ano de 2002, os dipositivos Stratix [10] vêm sendo utilizados tanto para aplicações comerciais quanto industriais. A família de dispositivos Stratix é baseada na tecnologia $1.5 - V$, $0.13 - \mu m$, processo SRAM, oferecendo até 79.000 elementos lógicos, 7 Mbits de memória embutida, blocos otimizados de DSP (*digital signal processing*) e entrada e saída de alto desempenho.

Os dispositivos Stratix apresentam um conjunto de funcionalidades, incluindo:

- Arquitetura de alta performance para acelerar projetos baseados em blocos lógicos;
- Recursos de memória TriMatrix abundantes para armazenamento *on-chip*;
- Blocos DSP com banda larga para aplicações com processamento de sinais intensos;
- Gerenciamento robusto de *clock* e síntese de frequência para *timing on-chip* e *off-chip* para maximizar o desmepneho do sistema utilizando PLLs embutidos (*phase-locked loops*);
- Qualidade de sinal maximizada a confiabilidade na transferência de dados terminações diferenciais e terminais *on-chip*.

A quantidade de blocos DSP, M512 RAM, M4K RAM e M-RAM variam juntamente com o número de colunas e linhas de acordo com cada dispositivo da família Stratix.

5.5.2 Kit Nios Altera

Para o desenvolvimento do trabalho foi utilizado o kit Nios da Altera edição Stratix [10] mostrado na Figura 5.8. O kit contém todas as ferramentas *Electronic Design Automation*(EDA) necessárias para se criar sistemas de alta performance em dispositivos lógicos programáveis.

Figura 5.8: Placa Stratix utilizada para o desenvolvimento [10]

O Nios II é um *core* de processador *Reduced Instruction Set Computer* (RISC) embutido (figura 5.9), desenvolvido pela Altera para a utilização em FPGAs. Este processador possui um pipeline de cinco estágios e pode ser utilizado nas versões 16 bits e 32 bits. Baseado na arquitetura *Harvard-Modified* sua CPU possui barramentos separados para dados e instruções com uma interface mestre para cada um deles. O processador permite que sejam desenvolvidas instruções customizadas, acelerando algoritmos de tempo crítico com hardware, conforme mostra a figura 5.10. As lógicas utilizadas para este fim podem inclusive acessar memórias e outros componentes externos ao processador.

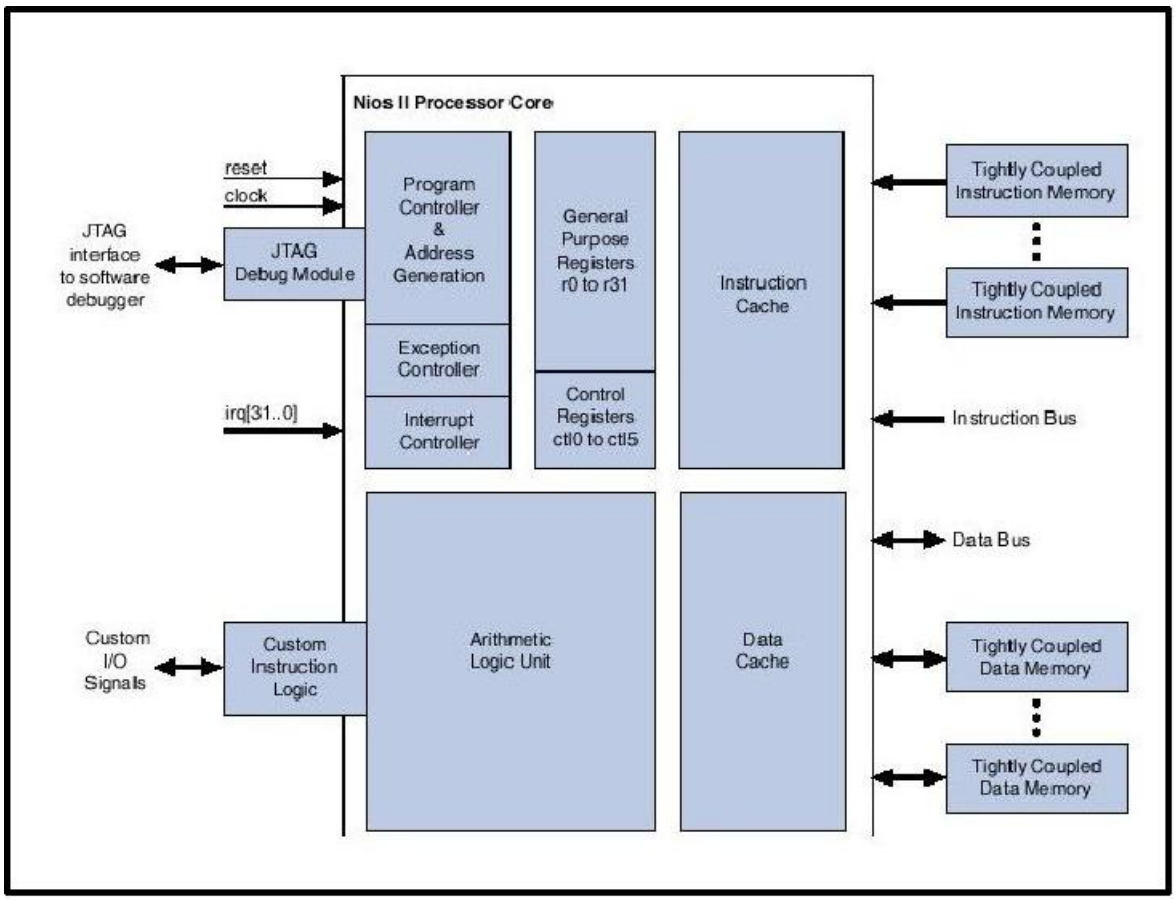

Figura 5.9: Arquitetura do *core* de prcessador Nios II [11]

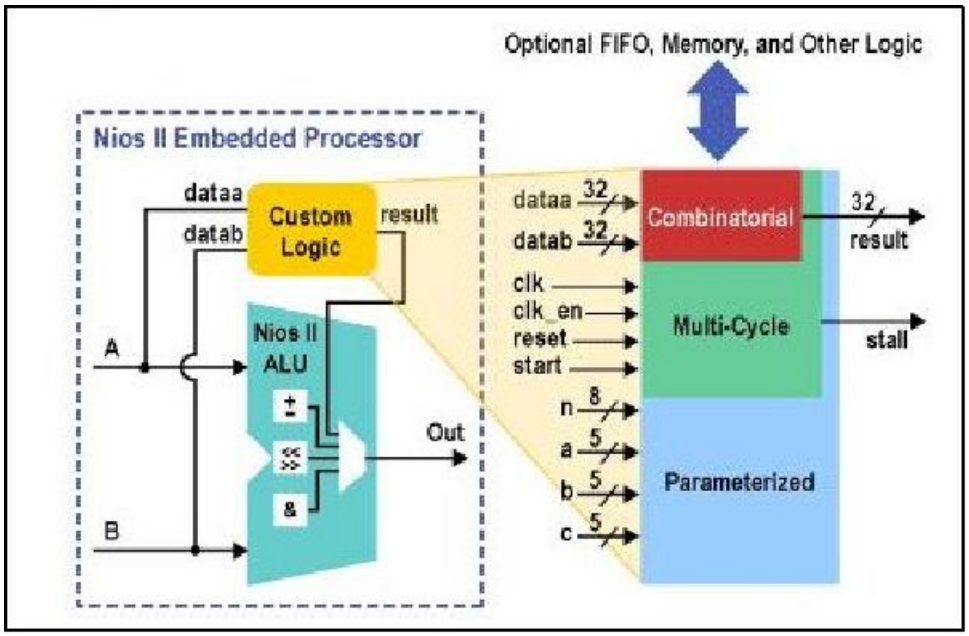

Figura 5.10: Lógica Customizada para Utilização de Novas Intruções no Nios II [10]

5.6 Considerações Finais

A computação reconfigurável está se tornando uma importante parte da pesquisa de arquitetura de computadores e de sistemas de software. Colocando-se a porção mais intensa de uma aplicação no hardware reconfigurável, ganha-se muito no desempenho.

Isso ocorre porque a computação reconfigurável combina muitos dos benefícios tanto da implementação em ASIC quanto em software. Como um software, o circuito mapeado é flexível e pode ser mudado ao longo da vida da aplicação. Como um ASIC, sistemas reconfigurável apresentam métodos para mapear circuitos em hardware.

Neste trabalho pretende-se identificar e implementar em hardware a porção mais intensa do algoritmo implementado por [13] no intuito de melhor o desempenho da aplicação.

Capítulo

6

Análise de Desempenho do Software

6.1 Considerações Iniciais

A avaliação de desempenho é necessária em todo o ciclo de vida de um sistema de computação, incluindo seu design, manufatura, compra/venda, upgrade, e assim por diante. Uma avaliação de desempenho é necessária quando um designer de sistema de computação quer comparar um número de designs alternativos e encontrar o melhor, ou então quando um administrador de sistema quer comparar uma porção de sistemas e decidir qual é o melhor para um dado conjunto de aplicações.

Mesmo não havendo alternativas distintas, a avaliação de desempenho de um sistema em questão ajuda a determinar a qualidade de sua desempenho e quando melhoras precisam ser realizadas. Infelizmente, os tipos de aplicações computacionais são tão numerosas que não é possível se ter um padrão de medida de desempenho, por isso, o primeiro passo na avaliação de desempenho é a escolha da medidas e técnicas corretas de desempenho para cada caso [53].

6.2 Sistemática para Análise de Desempenho

A maioria dos problemas são únicos. As métricas, cargas de trabalho, e técnicas de avaliação usadas para um problema geralmente não podem ser utilizadas para um próximo problema. Entretanto, existem passos comuns para todo projeto de avaliação de desempenho, dentre eles:

- **Estabelecimento de Metas e Definição do Sistema:** O primeiro passo em qualquer projeto de avaliação de desempenho é estabelecer as metas do estudo e

definir o que constitui o sistema através do delineamento de fronteiras. Dado o mesmo conjunto de hardware e software, a definição do sistema pode variar dependendo das metas do estudo. Dadas duas CPUs, por exemplo, a meta pode ser estimar seus impactos no tempo de resposta da interação com o usuário. Nesse caso, o sistema deve ser do tipo compartilhamento de tempo (*timesharing*), e os resultados podem depender significativamente dos componentes externos à CPU. Por outro lado, se duas CPUs são basicamente iguais, exceto pelas suas ULAs (Unidade Lógica-Aritmética) e a meta é decidir qual ULA deve ser escolhida, as CPUs devem ser consideradas como o sistema e apenas os componentes dentro da CPU devem ser considerados como parte do sistema.

A escolha das fronteiras do sistema afetam as métricas de desempenho assim como as cargas de trabalho utilizadas para comparar os sistemas. Além disso, entender as fronteiras é muito importante. Se não há como avaliar certo componente, é interessante que ele seja mantido fora da fronteira do sistema.

- **Lista de Serviços e Resultados:** Cada sistema provê um conjunto de serviços. Por exemplo, uma rede de computadores permite seus usuários enviarem pacotes para destinos específicos. Um sistema de base de dados responde a consultas. Um processador realiza um número de diferentes instruções. O próximo passo é na análise de um sistema é listar esses serviços. Quando um usuário requisita alguns desses serviços, existe um número possível de resultados, alguns desejáveis e outros não. Por exemplo, um sistema de base de dados pode responder a uma consulta corretamente, incorretamente (devido à inconsistências) ou não responder (devido a um *deadlock*). Uma lista de serviços e resultados possíveis é muito útil para a escolha das métricas e cargas de trabalhos corretas.

- **Seleção de Métricas:** O próximo passo é estabelecer critérios para comparar a desempenho. Esses critérios são chamados métricas. Em geral, as métricas estão relacionadas à velocidade, precisão, e disponibilidade dos serviços. A desempenho de uma rede, por exemplo, é medida pela velocidade (*throughput* e atrasos), precisão (taxa de erros), e disponibilidade dos pacotes enviados. A desempenho de um processador é medida pela velocidade (tempo gasto para execução) de várias instruções.

- **Lista de Parâmetros:** O passo seguinte no projeto de análise de desempenho é listar todos os parâmetros que afetam o desempenho. A lista pode ser dividida em parâmetros de sistema e de carga de trabalho. Parâmetros de sistema incluem tanto parâmetros de hardware quanto de software, os quais geralmente não variam de instalação para instalação de sistema. Parâmetros de carga de trabalho são características das requisições do usuário que variam de uma instalação para outra.

A lista de parâmetros pode não ser completa, já que depois da primeira passagem da análise, pode-se descobrir que existem parâmetros adicionais que interferem na desempenho. Esse novos parâmetros devem ser adicionados à lista, permitindo uma melhor compreensão do impacto de vários parâmetros e determinar quais dados devem ser coletados antes ou durante a análise.

- **Seleção de Fatores para Estudo:** A lista de parâmetros pode ser subdividida em duas partes: aquelas que vão variar durante a avaliação e aquelas que não irão. Os parâmetros que irão variar são chamados fatores, e seus valores são os níveis. Em geral, a lista de fatores, e seus níveis possíveis, são maiores que os recursos disponíveis permitem. Além disso, a lista se manterá crescendo até que se tornar óbvio que não há recursos disponíveis para estudar o problema. É melhor começar com uma lista pequena de fatores e um pequeno número de níveis para cada fator e estender a lista na próxima fase do projeto se os recursos permitirem. Por exemplo, deve decidir-se ter apenas dois fatores: quantidade e número de usuários. Para cada um desses dois fatores, deve-se escolher apenas dois níveis: pequeno e grande.

- **Seleção da Técnica de Validação:** As três mais difundidas técnicas de avaliação são modelagem analítica, simulação e medição em um sistema real. a seleção da técnica correta depende do tempo e dos recursos disponíveis para resolver o problema e o nível desejado de precisão.

- **Seleção de Carga de Trabalho:** A carga de trabalho consiste de uma lista de requisições de serviços ao sistema. Por exemplo, a carga de trabalho para comparar sistemas de base de dados comuns consiste em um conjunto de consultas. Dependendo da técnica de avaliação escolhida, a carga de trabalho pode ser expressa de diferentes formas. Para a modelagem analítica, a carga de trabalho é usualmente expressa como uma probabilidade de várias requisições. Para simulação, pode-se utilizar um conjunto de requisições medidas em um sistema real. Para medição real, a carga de trabalho pode ser um *script* de usuário para ser executado no sistema. Em todos os casos, é essencial que a carga de trabalho seja a representação do uso do sistema na vida real. Para produzir cargas de trabalho representativas, é necessário medir e caracterizar a carga de trabalho nos sistemas existentes.

- **Design de Experimentos:** Uma vez que se tem uma lista de fatores e seus níveis, é preciso decidir em uma seqüência de experimentos que oferecem o máximo de informação com esforço mínimo. Na prática, é útil conduzir um experimento em duas fases: Primeiramente, o número de fatores deve ser grande e o de níveis pequeno. O objetivo é determinar o efeito relativo de vários fatores. Em uma segunda etapa, o número de fatores é reduzido e o número de níveis desse fatores que têm impacto significativo são aumentados.

- **Análise e Interpretação de dados:** É importante observar que os resultados das medições e simulações são quantidades aleatórias, sendo que os resultados podem ser diferentes cada vez que o experimento é repetido. Na comparação de duas alternativas, é necessário levar em conta a variabilidade dos resultados.

  Interpretação dos resultados de uma análise é uma parte chave na arte de análise de desempenho. Deve ser entendido que a análise apenas produz resultados, e não conclusões. O resultado provê a base na qual o analista pode desenhar a conclusão.

- **Apresentação de Resultados:** O passo final no projeto de análise de desempenho é apresentar o resultado obtido. É importante que os resultados sejam apresentados de uma maneira fácil de entender. Isso usualmente requer a apresentação dos resultados em forma de gráficos e sem os jargões de estatísticas.

  Freqüentemente, nesse ponto do projeto, o conhecimento adquirido pelo estudo pode requerer que o analista volte e reconsidere algumas das decisões tomadas nos passos anteriores. Por exemplo, o analista pode querer redefinir as fronteiras do sistema ou incluir outros fatores e métricas de desempenho que não for consideradas anteriormente. O projeto completo, portanto, consiste em vários ciclos através dos passos, ao invés de uma única seqüência de passos.

6.3 Erros Mais Comuns na Avaliação de Desempenho

A maioria dos erros listados a seguir geralmente ocorrem de forma não intencional, ocasionados simplesmente por ponto de vista pouco abrangente, falta de conceitos e conhecimentos em relação às técnicas de avaliação.

- **Ausência de Objetivos:** Objetivos são uma parte importante de todo empenho. Qualquer empenho sem objetivos está fadado a falhar. Projetos de avaliação de desempenho não são uma exceção a essa regra. A necessidade de um objetivo pode parecer óbvia, mas muitos esforços para avaliação de desempenho iniciam-se sem objetivos bem definidos. As métricas, cargas de trabalho e metodologias são completamente dependentes dos objetivos. Traçar metas não é uma tarefa trivial, pois a maioria dos problemas são vagos, quando apresentados a primeira vez, e entender o problema suficientemente para traçar as metas é difícil.

- **Objetivos Tendenciosos:** Outro erro muito comum é a tendência implícita ou explícita nos objetivos iniciais. Se, por exemplo, o objetivo é mostrar que *nosso* sistema é melhor que o *deles*, o problema se torna encontrar as métricas e cargas de trabalho nas quais *nosso* sistema se torna melhor, ao invés de encontrar as métricas e cargas de trabalhos corretas para uma comparação imparcial.

- **Abordagem Não Sistemática:** Freqüentemente analistas adotam uma abordagem não sistemática por onde eles escolhem parâmetros de sistema, fatores, métricas e cargas de trabalho arbitrariamente. Isso leva a conclusões imprecisas.

- **Análise sem Entendimento do Problema:** Analistas inexperientes sentem que nada foi conseguido até que um modelo seja construído e alguns resultados numéricos sejam obtidos. Com experiência eles aprendem que uma grande parte do esforço da análise está em se definir o problema. O desenvolvimento de um modelo já é uma parte significativa do processo de solução do problema.

- **Métricas de Desempenho Incorretas:** A escolha de uma métrica de desempenho correta depende dos serviços providos pelo sistema que está sendo modelado. O que ocorre em muitos casos é que o analista escolhe as métricas mais fáceis de serem medidas ao invés das que são mais relevantes.

- **Cargas de Trabalho Não Representativas:** A escolha correta da carga de trabalho tem impacto no resultado do estudo da desempenho, já que uma escolha errada irá levar a resultados imprecisos.

- **Técnica de Avaliação Errada:** Dentre as três técnicas de avaliação (medição, simulação e modelo analítico) os analistas freqüentemente escolhem a que eles estão acostumados a utilizar para todos projetos de avaliação de desempenho. O analista deve ter um conhecimento básico de todas as três técnicas.

- **Ignorar Fatores Significativos:** Nem todos os parâmetros têm um efeito igual na desempenho, por isso é importante identificar aqueles que se variados, irão ocasionar um impacto significativo na análise de desempenho do sistema.

- **Design Experimental Inapropriado:** Uma seleção inapropriada números de simulações ou experimentos a serem realizados, bem como os parâmetros a serem utilizados, irão ocasionar um desperdício de recursos e tempo do analista.

- **Nível de Detalhamento Inapropriado:** Deve-se evitar formulações muito abrangentes ou muito restrita para uma modelagem eficaz.

- **Sem Análise:** Um dos problemas muito comuns com projetos de medição é que eles são executados por analistas que são muito bons em colher resultados, mas não tem tanto conhecimento para analisar os dados colhidos.

- **Análise Não Sensitiva:** Freqüentemente analistas colocam muita ênfase no resultado de suas análises apresentando-a como fatos ao invés de evidências.

- **Ignorar Erros de Entrada:** Ocorrem geralmente quando o parâmetro de interesse não pode ser medido e uma outra variável que pode ser medida é utilizada para estimar o parâmetro.
- **Não Assumir Mudanças Futuras:** Usualmente, assume-se que o futuro será igual o passado. Um modelo baseado em cargas e desempenho do passado é utilizado para prever desempenho no futuro.
- **Ignorar Variabilidade:** Estipular a variabilidade nesses tipo de problema é muito difícil, ou então impossível.
- **Apresentação Imprópria dos Resultados:** A maior parte das análises de desempenho é feita para ajudar em tomadas de decisões. Um analista que não produz nenhum resultado útil é tão falho quanto um que não consegue exibir resultados válidos de uma maneira compreensível.

6.4 Análise dos Algoritmos de Localização e Mapeamento

Baseando-se na sistemática descrita na seção 6.2, este trabalho tem como meta de análise de desemepenho, medir a eficiência dos algoritmos de localização e mapeamento a serem implementados em hardware.

A métrica a ser utilizada é o tempo de execução de cada função. Essa tarefa será realizada com a utilização da ferramenta gprof [19]. Busca-se reduzir o tempo de execução das funções mais dispediosas através da implementação destas em hardware.

A técnica de medição a ser utilizada será a medição no sistema real, já que o código fonte em C da implementação de [1] está disponível, o que possibilita a compilação do programa para estudo com a ferramenta gprof.

A carga de trabalho a ser utilizada consiste em mapas e *logs* de teste referentes a experimentos realizados no campus da USC (*University of Southern California*) descritos no capítulo 4.

Outro aspecto pertinente em relação à análise de desempenho diz respeito à análise e interpretação de dados, que serão feitos através de tabelas geradas pela ferramenta gprof. Os dados presentes nestas tabelas podem ser ilustrados através de gráficos para auxiliar na interpretação dos resultados, o que é muito importante para uma análise de desempenho bem elaborada.

6.4.1 Breve descrição da ferramenta GPROF

A ferramenta gprof produz um perfil de execução de programas em C, Pascal ou Fortran77. O efeito das rotinas chamadas são incorporadas no perfil de cada porção que as chama. Os dados do perfil são tirados do diagrama de chamadas (*call graph profile*) no arquivo padrão "gmon.out"que é criado pelos programas compilados com opção "-pg". Essa opção também faz uma ligação com a biblioteca de rotinas que são compiladas para traçamento de perfil (*profiling*). Gprof lê o dado do objeto no arquivo padrão "a.out"e estabelece a relação entre sua tabela de símbolos e o diagrama de chamada do "gmon.out". Gprof calcula a quantidade de tempo gasta em cada rotina. Em seguida esses tempos são propagados ao longo das extremidades do diagrama de chamadas. Ciclos são descobertos e chamadas dentro de um ciclo são mostradas para explicitar o compartilhamento de tempo do ciclo.

A ferramenta mostra quanto tempo o programa gastou em cada função, e quantas vezes essa função foi chamada. O diagrama de chamadas mostra para cada função, quais funções a chamaram, e quais outras funções ela chamou, e quantas vezes isso aconteceu. É fornecido também uma estimativa de quanto tempo foi gasto em cada sub-rotina de cada função.

% tempo (t)	t cumulat.	t próp.	chamadas	próp. cham.	tot. cham.	nome
42.09	801.16	801.16	100000	0.00	0.00	calc_mean
31.11	1393.34	592.18	100000	0.00	0.00	log_data
16.61	1709.55	316.21	733400000	0.00	0.00	normal
4.65	1798.16	88.60	300139284	0.00	0.00	sample

Tabela 6.1: Exemplo de *Flat Profile* gerado pelo gprof

No exemplo ilustrado pela tabela 6.1 observa-se que a ferramenta gprof é capaz de fornecer informações essenciais para uma análise eficiente. As funções são classificadas de acordo com a porcetagem de tempo de execução gasto, o que possibilita identificar facilmente os gargalos do algoritmo.

6.5 Considerações Finais

Neste capítulo foram discutidos alguns dos principais tópicos de uma metodologia para analisar o desempenho de um programa, servindo como base para o *codesign* hardware/software a ser realizados nos algoritmos de localização e mapeamaneto para robôs móveis. Também foi mostrado que a ferramenta gprof é capaz de realizar uma boa análise satisfazendo os requisitos descritos na seção 6.2.

Capítulo

7

Análise da Implementação do Algoritmo de Localização

7.1 Considerações Iniciais

O algoritmo analisado neste capítulo diz respeito a implementação do algoritmo de localização por [13], que utiliza um filtro de partículas baseado no algoritmo de Monte Carlo [54, 55] , que é uma implementação particular do filtro de Bayes.

7.2 Introdução

Experimentos foram realizados no campus da USC (*University of Southern California*), utilizando um robô segway RMP armazenando os dados em um arquivo (log.txt), gentilmente cedidos por Denis F. Wolf. O trajeto do experimento corresponde 2 km percorridos dentro do campus, realizando 3 loops completos, gerando bons mapas e resultados de localização, figura 7.1.

7.3 Funcionamento

A função "calc_lh_map()"calcula o mapa de probabilidade (*likelihood map*), que é o mapa onde as informações do laser serão aplicadas, a função "init_particles()"inicializa as partículas. Essas funções são executadas uma única vez, e em seguida é chamada a função "run_filter()"que tem um laço que permanece em execução até que todos os dados sejam processados.

Basicamente o loop dentro da "run_filter()"consiste em:

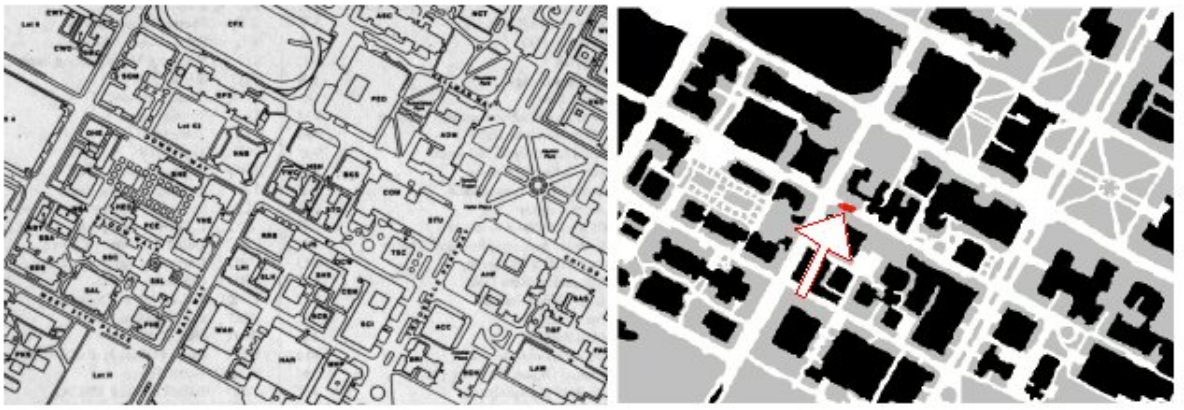

Figura 7.1: Mapa do campus da USC escaneado (a esquerda) e localização utilizando MCL, a estimação realizada pelo filtro de partículas está indicada pela seta. [1]

1. update_data_file(): lê uma série de dados do arquivo de log ("log.txt");
2. calc_mean(): calcula localização e variância do robô;
3. log_data(): salva localização e variância em um arquivo("new_log.txt");
4. em seguida, é calcula a diferença de posição do robô (x, y, a) estimada pelo odômetro. Essa diferença é passada como argumento para a função seguinte ("do_action_model2()");
5. do_action_model2(): baseado na diferença de posição do odômetro, essa função propaga as partículas no mapa;
6. print_particles(): Desenha as particulas na tela;
7. do_observation_model(): Calcula a probabilidade de cada partícula representar corretamente a posição do robô baseado nos dados do laser projetados sobre o mapa de probabilidade (*likelihood map*).
8. resample(): elimina as partículas com baixa probabilidade de representarem corretamente a posicão do robô.

O algoritmo permanece repetindo os passos de 1 a 8 até que todos os dados sejam processados.

7.3.1 Fluxo de Execução

O algoritmo é inicializado com uma chamada à função "gtk_init()"(figura 9.1 - Apêndice) que carrega a interface do programa (figura 9.2) e oferece ao usuário três opções de botões para serem utilizados. O primeiro botão "Connect"chama a função "on_button1_clicked()", o segundo, "Run"faz chamada à função "on_button3_clicked()"e

o terceiro botão "Quit"finaliza a execução do programa fazendo a chamada à função "on_button2_clicked()". Todas as funções do programa serão detalhadas adiante da seção 7.3.2.

Para um melhor entendimento o diagrama do fluxo de execução pode ser dividido em dois níveis, conforme as figuras 7.2 e 7.3.

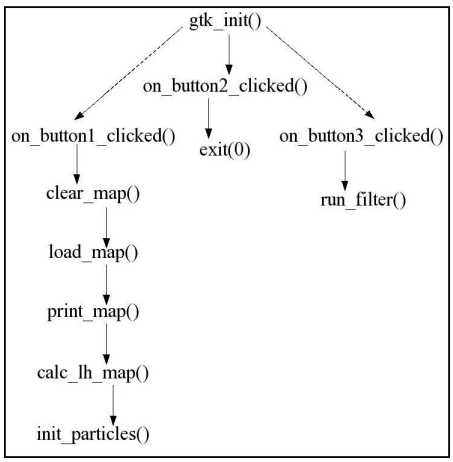

Figura 7.2: Diagrama do fluxo de execução do algoritmo - nível 1

As funções "print_map()", "init_particles()"e "run_filter()"fazem chamadas a outras funções conforme ilustrado na figura 7.3.

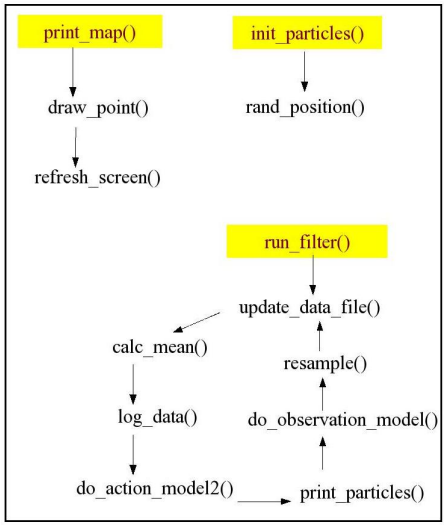

Figura 7.3: Diagrama do fluxo de execução do algoritmo - nível 2

Além dos três botões "Connect", "Run"e "Quit", a interface do programa apresenta duas caixas para que o usuário forneça o nome do arquivo de log e do arquivo de mapa a ser utilizado durante a execução do programa. Na figura 9.2 observar-se a utilização dos arquivos "log.txt"e "map3.map", ambos referentes aos experimentos realizados no campus da USC.

7.3.2 Descrição das Funções

As principais funções do programa, descritas nas tabelas abaixo, estão implementadas nos arquivos "callbacks.c"(tabela 7.1), "robot_models.c"(tabela 7.2), "map.c"(tabela 7.3) e "part_filter.c"(tabela 7.4).

Função	Descrição
on_button1_clicked()	Botão "CONNECT": Inicia a execução do programa, conecta o player/stage e inicializa as partículas no mapa
on_button2_clicked()	Botão "QUIT": Fecha o programa
on_button3_clicked()	Botão "RUN": Inicializa a execução do filtro de partículas

Tabela 7.1: Arquivo callbacks.c

Função	Descrição
do_action_model2()	Baseado na diferença de posição do odômetro, essa função propaga as partículas no mapa
do_observation_model()	Calcula a probabilidade de cada partícula representar corretamente a posição do robô baseado nos dados do laser projetados sobre o mapa de probabilidade (likelihood map)

Tabela 7.2: Arquivo robot_models.c

Função	Descrição
clear_map()	Zera a matriz map[700][500]
load_map()	Abre para leitura o arquivo com o mapa "map_file_name"; Lê as dimensões X ,Y e tamanho do grid do mapa; Lê cada ponto "p"do mapa e atribui valores: -1 se p < -1 1 se p > 1 0 caso contrário
print_map()	Imprime todos os pontos do mapa na tela
calc_lh_map()	Calcula a vizinhança de cada ponto no mapa
print_lh_map()	Imprime os pontos do mapa de vizinhança

Tabela 7.3: Arquivo map.c

Funções Computacionalmente Custosas

As três funções mais custosas computacionalmente são "calc_mean()", "log_data()"e "normal()". As duas primeiras são chamadas pela função "run_filter()", e "normal()"é chamada pela função "do_action_model2()"que por sua vez também é chamada por "run_filter()". Através do perfil plano (tabela 7.5) gerado pela ferramenta gprof é possível observar que juntas essas três funções são responsáveis por quase 90% do tempo total de execução do programa. A tabela 7.5 apresenta apenas as funções mais importantes

Função	Descrição
init_particles()	Espalha aleatoriamente as partículas no mapa através da função "rand_position"
calc_mean()	Calcula localização e variância do robô
print_particles()	Desenha as partículas em suas respectivas posições no mapa
print_odom()	Imprime no mapa os pontos referentes à odometria
round_angle()	Inverte o ângulo dos sensores
rand_position()	Atribui valores aleatórios às coordenadas das partículas
sample()	Obtém partícula de amostra do mapa
resample()	Obtém partícula de amostra do mapa utilizando uma taxa de amostra passada como argumento para a função para eliminar as partículas com baixa probabilidade e representar corretamente a posição do robô
update_laser_ang()	Atualiza o ângulo do sensor laser através de dados lidos do arquivo "laser_ang.txt"
update_data_file()	Atualiza valores odométricos, de leitura dos sensores e tempo, de acordo com os valores do arquivo "log.txt"
log_data()	Armazena os valores resultantes da execução do programa (localização e variância) no arquivo "new_log.txt"
run$_filter$()	Função que executa o filtro de partículas através de várias chamadas às funções anteriormente descritas nesta tabela

Tabela 7.4: Arquivo part_filter.c

utilizadas pela análise. A tabela completa gerada no estudo realizado pode ser encontrada nos anexos (tabela 9.1 Apêndice A).

% tempo (t)	t cumulat.	t próp.	chamadas	próp. cham.	tot. cham.	nome
42.09	801.16	801.16	100000	0.00	0.00	calc_mean
31.11	1393.34	592.18	100000	0.00	0.00	log_data
16.61	1709.55	316.21	733400000	0.00	0.00	normal
4.65	1798.16	88.60	300139284	0.00	0.00	sample
4.04	1875.07	76.91	1	0.08	1.90	run_filter
0.74	1889.15	14.08	1466	0.00	0.00	print_particles

Tabela 7.5: Perfil plano gerado pelo gprof

No código da função "calc_mean()", figura 9.3 (Apêndice), observa-se dois laços de repetição que foram desenrolados para uma implementação em hardware visando aumentar o desempenho na execução do programa.

Para uma implementação em hardware da função ilustrada na figura 9.3, foi realizado um estudo das dependências de dados durante a execução dessa função, conforme pode ser observado nos DFGs (*Data Flow Graph*) da figura 7.4 (a).

Os diagramas de fluxo são isométricos para as coordenadas cartesianas X,Y e para a coordenada A, relativa ao ângulo para orientação do robô, conforme pode ser observado nos diagramas das figuras 9.4 e 9.5 (Apêndice).

Observa-se que as operações envolvendo aritmética de ponto flutuante são bem simples incluindo apenas adições, subtrações, multiplicações, divisões e módulos, o que

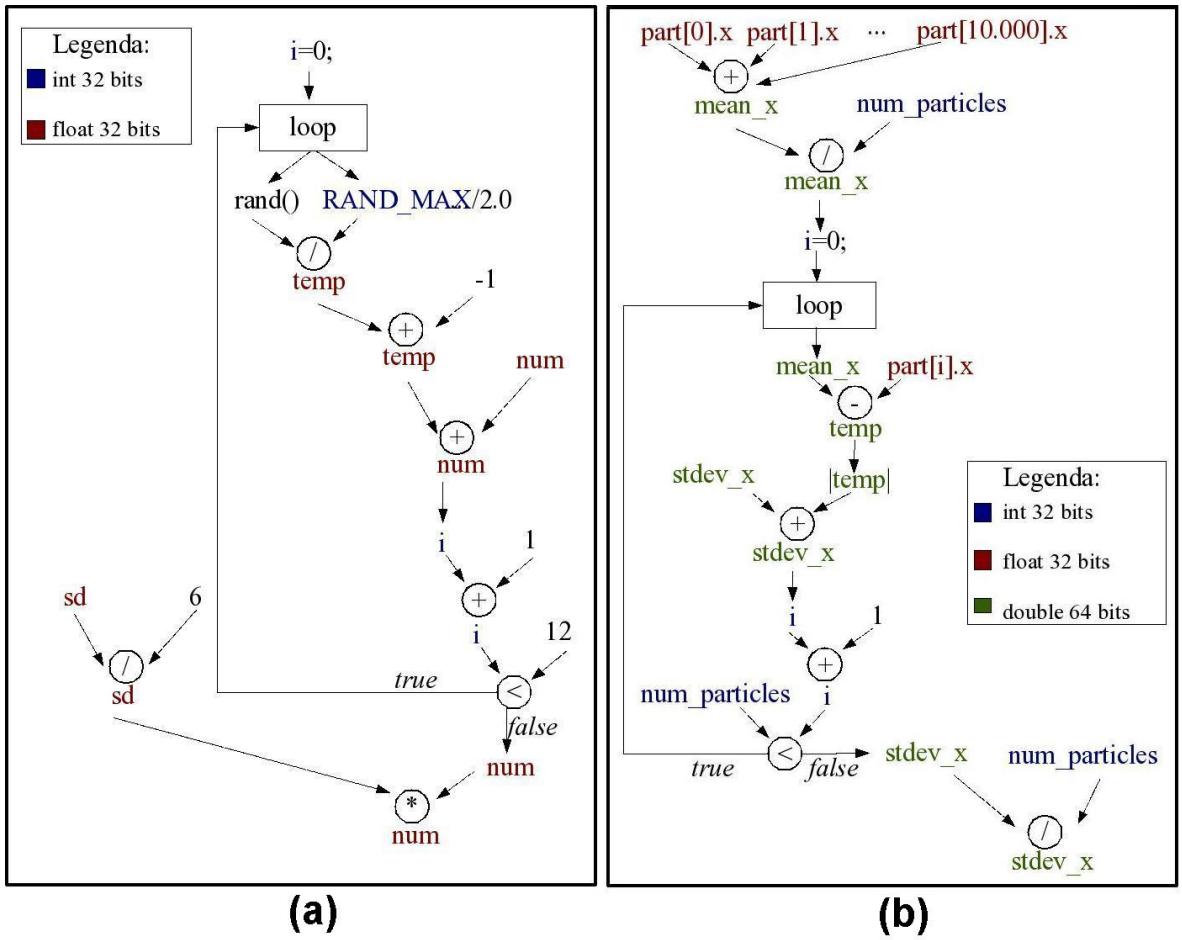

Figura 7.4: **(a)** DFG da função "normal()" **(b)** DFG da função "calc_mean()"em relação à coordenada X.

facilitou muito a implementação embarcada. É muito comum no âmbito da robótica móvel deparar-se com modelos mais complexos utilizando senoidais [56] e redes triangulares 3D (*triangular mesh* 3D) [57]. Assim, as operações referentes aos diagramas da figura 7.4 foram facilmente portadas para executar em hardware através da unidade de ponto flutuante FPMU [12] que será detalhado no capítulo 8.

As variáveis utilizadas pela função "calc_mean()"podem ser classificadas conforme a tabela 7.6:

Variável	**Tipo**	**Tamanho**
mean_x, mean_y, mean_a	double	64 bits
temp	double	64 bits
stdev_x, stdev_y, stdev_a	double	64 bits
part[i].x, part[i].y, part[i].a	float	32 bits
i	int	32 bits
num_particles	int	32 bits

Tabela 7.6: Tipo e tamanho em bits das variáveis da função "calc_mean()".

A função "log_data()", figura 9.6 (Apêndice) é computacionalmente custosa por realizar muita entrada e saída. Mudanças na implementação de entrada e saída para uma implementação em hardware foram realizadas no intuito de melhorar o desempenho. Os dados que eram armazenados em arquivos de log a cada fase de execução do programa, agora são simplesmente armazenados no mapa, não gerando intensas operações de entrada e saída.

A entrada e saída na implementação original do algoritmo, não se levando em consideração os dados impressos na tela durante a execução do programa, é realizada da seguinte forma:

1. load_map(): abre o arquivo de mapa ("map3.map") para obter as dimensões, grid e dados do mapa;

2. update_laser_ang(): lê o arquivo ("laser_ang.txt") contendo os ângulos de laser a serem utilizados durante a execução do programa;

3. update_data_file(): lê dados do arquivo de ("log.txt") que foram coletados durante o experimento no campus da USC.

4. log_data(): salva localização e variância em um arquivo("new_log.txt");

Na abertura dos arquivos citados nos 4 passos acima, cada registro é lido e escrito, através de variáveis específicas para esse fim. Desta forma, a organização de cada um desses arquivos pode ser observada nas figuras 9.7, 9.8, 9.9 e 9.10 (Apêndice).

A função "normal()", 9.11 (Apêndice), apesar de simples, é computacionalmente custosa devido ao grande número de vezes (733.400.000) que ela é chamada durante a

execução do programa. Uma implementação em hardware dessa função é de grande valia para uma ganho em desempenho na execução do programa.

Para uma implementação em hardware da função ilustrada na figura 9.11, foi realizado um estudo do DFG (*Data Flow Graph*) durante a execução dessa função, conforme pode ser observado na figura 7.4 (b). As variáveis utilizadas pela função "normal()"podem ser classificadas conforme a tabela 7.7:

Variável	**Tipo**	**Tamanho**
temp	float	64 bits
sd	float	64 bits
num	float	32 bits
i	int	32 bits
RAND_MAX	int	32 bits

Tabela 7.7: Tipo e tamanho em bits das variáveis da função "normal()".

Utilização da FPMU

Para a implementação das funções "normal()"e "calc_mean()"em hardware foi utilizada uma unidade de ponto flutuante de 32 bits denominada FPMU (*Float Point Modular Unit*) [12] [58]. Entretanto, o código em C original apresenta algumas variávies de ponto flutuante de 64 bits (tabelas 7.7 e 7.6). Assim, as variávies do tipo *double* de 64 bits em C, foram substituídas por *float* de 32 bits. Para verificar o impacto desta mudança, realizou-se um estudo da perda de precisão na execução do algoritmo original utilizando-se variáveis de 32 bits ao invés de 64 bits.

A análise foi feita com base nos dados gerados pela execução do algoritmo, que são armazenados no arquivo "new_log.txt". Na figura 9.12 (Apêndice) é ilustrado um trecho do arquivo "new_log.txt"com os dados de execução obtidos com variáveis de 64 bits, e na figura 9.13 (Apêndice) com 32 bits. Nota-se que os dados relativos aos tempos iniciais e finais ("time[0]", primeiro valor do registro e "time[1]", segundo valor do registro) apresentam os mesmos valores na execução em 32 bits. Isto ocorre pois o tempo de execução deste trecho do código, responsável pela atualização dos valores das variáveis é muito curto, sendo que as variáveis "time[0]"e "time[1]"de 32 bits não conseguem representar esta diferença. Entretanto a informação do tempo de execução é irrelevante para o resultado final do algoritmo. Os resultados da análise de perda de precisão podem ser observados na tabela 7.8.

Através dos resultados ilustrados na tabela 7.8 verifica-se que a redução do tamanho das variávies de 64 para 32 bits não ocasiona perdas significativas para o resultado da execução do algoritmo. Vale lembrar que o algoritmo de [13] utiliza números aleatórios, através das funções "rand()"e "srand()"da biblioteca padrão C para calcular

Variação Mínima	Variável	Variação Máxima
+/- 0,4437%	glb_pos.mean_x	+/- 0,7672%
+/- 0,1514%	glb_pos.mean_y	+/- 0,3369%
+/- 0,1621%	glb_pos.mean_a	+/- 0,8501%
+/- 0,2864%	glb_pos.stdev_x	+/- 0,4403%
+/- 0,0155%	glb_pos.stdev_y	+/- 0,4614%
+/- 0,0621%	glb_pos.stdev_a	+/- 0,3138%

Tabela 7.8: Variação da Precisão dos Resultados Utilizando Variávies de 32 bits

as probabilidades de cada partícula, portanto os resultados obtidos por cada execução diferem, mesmo com a utilização do mesmo código com as mesmas entradas.

Threads

No código original do algoritmo de [13], a função "run_filter()", responsável pelo principal laço de execução do programa era chamada através da utilização de *threads* conforme o trecho extraído do código abaixo:

```
pthread_create(&thread1, NULL, (void *)&run_filter, NULL);
```

Esse tipo de implementação levava a uma interpretação equivocada em relação ao custo de execução de cada função pela ferraementa gprof. Era levado em consideração apenas o fluxo de execução inicial do programa, excluindo da análise o peso da execução da principal função do programa ("run_filter()"), conforme pode ser observado pelo trecho de perfil plano da tabela 7.9.

% tempo (t)	t cumulat.	t próp.	chamadas	próp. cham.	tot. cham.	nome
73.18	2.92	2.92	2	2.92	2.92	init_particles
24.06	3.88	0.96	1	0.96	0.96	calc_lh_map
1.50	3.94	0.06	1	0.06	0.06	load_map
1.00	3.98	0.04	1	0.04	0.04	print_map
0.25	3.99	0.01	1	0.01	0.01	clear_map
0.00	3.99	0.01	15891697	0.00	0.00	normal
0.00	3.99	0.01	2368746	0.00	0.00	draw_point

Tabela 7.9: Perfil plano gerado pelo gprof de maneira equivocada

No perfil plano da tabela 7.9 as duas funções ("init_particles()"e "calc_lh_map()") que são executadas antes da chamada de "run_filter()"com a utilização de *threads*, são tomadas como as mais custosas computacionalmente.

Para uma análise real do problema, o programa foi modificado, excluindo a utilização de *threads*, já que esse tipo de abordagem muito utilizada em linguagens de auto nível não foi necessária para uma implementação direta em hardware.

Dessa forma foi possível analisar corretamente o código através da ferramenta gprof e concluir que o conjunto das funções mais custosas do programa é representado por "calc_mean()", "log_data()"e "normal()".

Execução em Linha de Comando

No intuito de portar o programa em C desenvolvido para uma implementação em hardware, o código foi alterado excluindo-se tudo o que se referia a parte gráfica para uma execução apenas em linha de comando. Também foi excluída a função "log_data()"pois a entrada e saída foi implementada de maneira diferente da original.

Dessa forma, analisando-se o novo programa executando apenas em linha de comando, a ferramenta gprof forneceu os seguintes resultados, de acordo com a tabela 7.10:

Função	**% tempo**	**tempo (s)**	**n chamadas**
normal()	45.168	271.604	766100000
sample()	18.892	113.77	318875399
calc_mean()	16.23	97.46	766100000
run_filter()	9.7	58.47	766100000
do_action_model2()	7.33	44.29	766100000
resample()	1.61	9.71	766100000
do_observation_model()	0.65	3.91	766100000
calc_lh_map()	0.18	1.09	766100000
rand_position()	0.16	0.94	766100000
init_particles()	0.08	0.46	766100000
update_data_file()	0.04	0.23	766100000
round_angle()	0	0	766100000
clear_map()	0	0	766100000
load_map()	0	0	766100000
update_laser_ang()	0	0	766100000

Tabela 7.10: Perfil plano gerado pelo gprof na execução da implementação apenas em linha de comando

Os valores foram calculados a partir da média entre cinco tabelas geradas pelo gprof. Observa-se que as funções mais custosas são "normal()", "sample()"e "calc_mean()". Juntas, elas representam mais de 80% do total de execução do algoritmo. As figuras 7.5 e 9.14 (Apêndice) ilustram a porcentagem de execução de cada função.

O tempo total de execução do programa na média das 5 execuções resultou em 601,93 segundos (10 minutos aproximadamente) e o tempo gasto por cada função pode ser observado pela figura 7.6:

Existem funções que são computacionalmente pouco custosas, mas que devido a um número muito grande de vezes que elas são chamadas, elas representam uma porcentagem considerável no total do tempo de execução, como por exemplo a função "normal()". Em contrapartida, a função "run_filter()"é chamada apenas uma vez durante a execução do

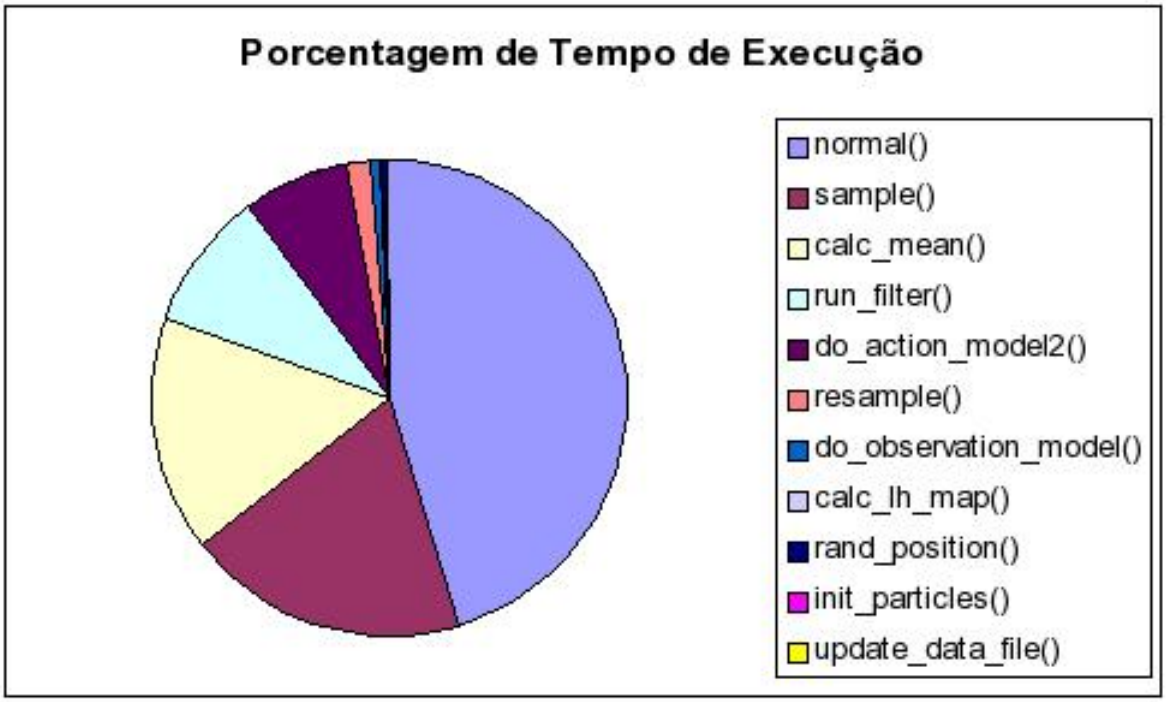

Figura 7.5: Porcentagem de execução de cada função do programa

Figura 7.6: Tempo de execução de cada função do programa

programa e representa $9,7\%$ do total de tempo de execução, sendo proporcionalmente a função mais custosa do programa.

7.3.3 Descrição das Constantes e Variáveis

As constantes presentes no código estão definidas no arquivo "defs.h"da biblioteca do filtro de partículas. Todas as principais variáveis e constantes foram classificadas conforme a sua função desempenhada no programa. Dessa forma, foi possível se obter um melhor entendimento do código, o que é essencial para uma análise eficiênte. As tabelas 7.11 e 7.12 listam, respectivamente, todas as principais constantes e variáveis do programa juntamente com uma descrição da função de cada uma.

Constante	**Descrição**	**Valor**
MAX_N_PARTICLES	Número máximo de partículas	100000
MIN_N_PARTICLES	Número mínimo de partículas	200
LASER_SCAN_SKIP	Determina a granulosidade da leitura do sensor laser (de 50 em 50 graus no caso)	50
OBS_MODEL_FREQ	Freqüência do modelo de observação	100
CLIENT_READ_SKIP	Freqüência do leitor cliente	8
PRINT_SKIP	Granulosidade da impressão de partículas (a cada 5 calculada, uma é impressa)	5

Tabela 7.11: Constantes utilizadas

7.4 Considerações Finais

Neste capítulo, foi descrita de maneira detalhada a análise realizada sobre o algoritmo de localização a ser portado para uma solução embarcada. Todo o código foi completamente entendido e as porções mais custosas identificadas para serem executadas diretamente em hardware.

Variável	**Descrição**	**Valor Ini.**
map_file_name	String com o nome do mapa a ser lido pelo filtro de partículas	map3 .map
log_file_name	String com o nome do arquivo de log que armazenará resultados	log.txt
x,y	Coordenadas do mapa	0
map[700][500]	Matriz que armazena os dados do mapa	0
num_particles	Número de partículas que são utilizadas	100000
MAP_grid_size	Variável que recebe o tamanho do grid em centímetros lido do mapa	5
MAP_mult_factor	Fator de multiplicação para fazer com que o mapa fique dividido em 100 partes iguais	20
MAP_dim_x	Variável que recebe o valor da coordenada x lida no mapa	N.E.
MAP_dim_y	Variável que recebe o valor da coordenada y lida no mapa	N.E.
lh_map[700][500]	Matriz que armazena os dados do mapa de vizinhança calculado durante a execução do programa	N.E.
rfx, rfy, rfa, rfms	Variáveis que recebem valores aleatórios para coordenadas x,y,ângulo e escala do mapa respectivamente	N.E.
part[MAX_N_ PARTICLES]	Vetor que armazena as partículas (particles) calculadas pelo filtro	N.E.
particles	Estrutura da partículas, compostas por coordenadas (x,y,ângulo), escala do mapa, w e acc_w	N.E.
f	Variável que recebe o valor do ângulo do laser lido do arquivo "laser_ang.txt"	N.E.
scan[361][2]	Matriz que armazena todos os valores assumidos pela variável f	N.E.
odom[12]	Vetor que armazena valores odométricos obtidos do arquivo de log	N.E.
mean_x, mean_y, mean_a	Variáveis que armazenam médias entre cada coordenada x,y,a	0
stdev_x, stdev_y, stdev_a	Variáveis que armazenam desvios entre cada coordenada x,y,a	0
xo,yo,ao	Valores iniciais das coordenadas x,y e ângulo do sensor	Lido do mapa
xn,yn,an	Variávies que armazenam os valores atualizados das coordenadas x,y e ângulo do sensor	N.E.
dif_x, dif_y, dif_a	Diferenças entre os valoes inicias, e os valores atuais de x,y,a	N.E.

Tabela 7.12: Variáveis utilizadas

Capítulo

8

Implementação dos Algoritmos de Localização e Mapeamento em FPGA

8.1 Considerações Iniciais

Este capítulo trata da implementação em FPGA dos algoritmos de filtro de partículas e mapeamento descritos anteriormente. Também é feita uma breve abordagem dos recursos utilizados para essa implementação em hardware reconfigurável.

8.2 A unidade de ponto flutuante FPMU

A FPMU (Float Point Modular Unit) [12] foi integrada como uma instrução personalizada ao processador Nios II da Altera para efetuar em hardware as principais operações de ponto flutuante durante a execução da solução embarcada. Essa decisão foi tomada pois as funções mais custosas computacionalmente analisadas no capítulo 7 realizam esse tipo de operação em grande quantidade.

A técnica de se utilizar uma unidade de ponto flutuante para executar diretamente em hardware visando uma melhora em desempenho é bem comum na literatura atual. [57].

8.2.1 Representação de números de ponto flutuante

Para representar números reais, o mais comum é o uso da notação científica, usando a maior parte dos bits para representar a mantissa e a menor parte para representar o expoente, figura 8.1. Assim, 18 estes números de ponto flutuante podem somente ser representados com um número específico de dígitos significativos. Isto tem um grande impacto na operação com aritmética de ponto flutuante. É fácil ver o impacto de utilizar

precisão limitada na aritmética de ponto flutuante: adotando um formato simplificado de ponto flutuante decimal, por exemplo, com uma mantissa com três dígitos significativos e um expoente decimal com dois dígitos, sendo a mantissa e o expoente com valores sinalizados.

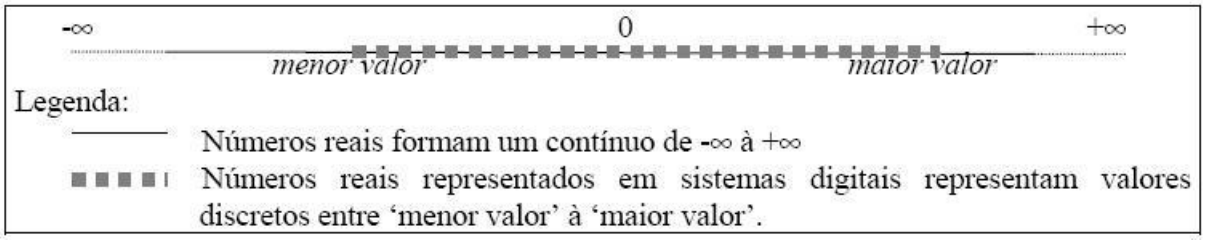

Figura 8.1: Diferença dos valores em número real para a representação em ponto flutuante em sistemas digitais

Ao adicionar ou subtrair dois números em notação científica, é necessário o ajuste dos expoentes dos dois números para o mesmo valor. Por exemplo, antes de adicionar $1.23e1$ a $4.56e0$, ajusta-se os valores para que eles tenham o mesmo expoente. Um caminho a seguir é converter $4.56e0$ para $0.456e1$ e então adicionar a $1.23e1$, produzindo como resultado $1.686e1$. Infelizmente, o resultado não cabe em três dígitos significativos; assim, é preciso truncá-lo em três dígitos significativos. Arredondamento, geralmente produz uma melhor precisão, e assim obtém-se $1.69e1$. Como se pode ver na figura 8.2, a perda de precisão afeta a exatidão do valor. Entretanto, essa pequena perda não é significativa, conforme as análise realizadas no capítulo 7.

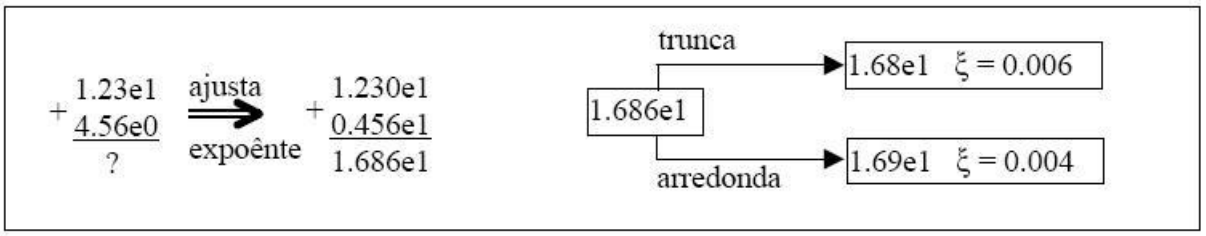

Figura 8.2: Arredondando o resultado para obter maior precisão [12]

No exemplo anterior foi possível arredondar o resultado porque os cálculos foram feitos com cinco dígitos significativos. Se os cálculos estivessem limitados a três dígitos significativos, isto é, a FPMU trabalhando internamente com três dígitos, ocorreria o truncamento do último dígito, resultando assim $1.68e1$ ao invés de $1.69e1$, que contém um erro maior do que o resultado arredondado. Com isso conclui-se que: dígitos extras durante a computação melhoram a exatidão dos resultados.

8.2.2 Padrão para Ponto Flutuante IEEE 754

A FPMU integrada ao processador Nios II utiliza o padrão para ponto flutuante IEEE 754. A representação de um número em ponto flutuante na forma binária é apresen-

tada na figura 8.3, onde "s"representa o sinal do ponto flutuante (negativo igual a "1"), "y"representa o expoente, e "x"representa a mantissa.

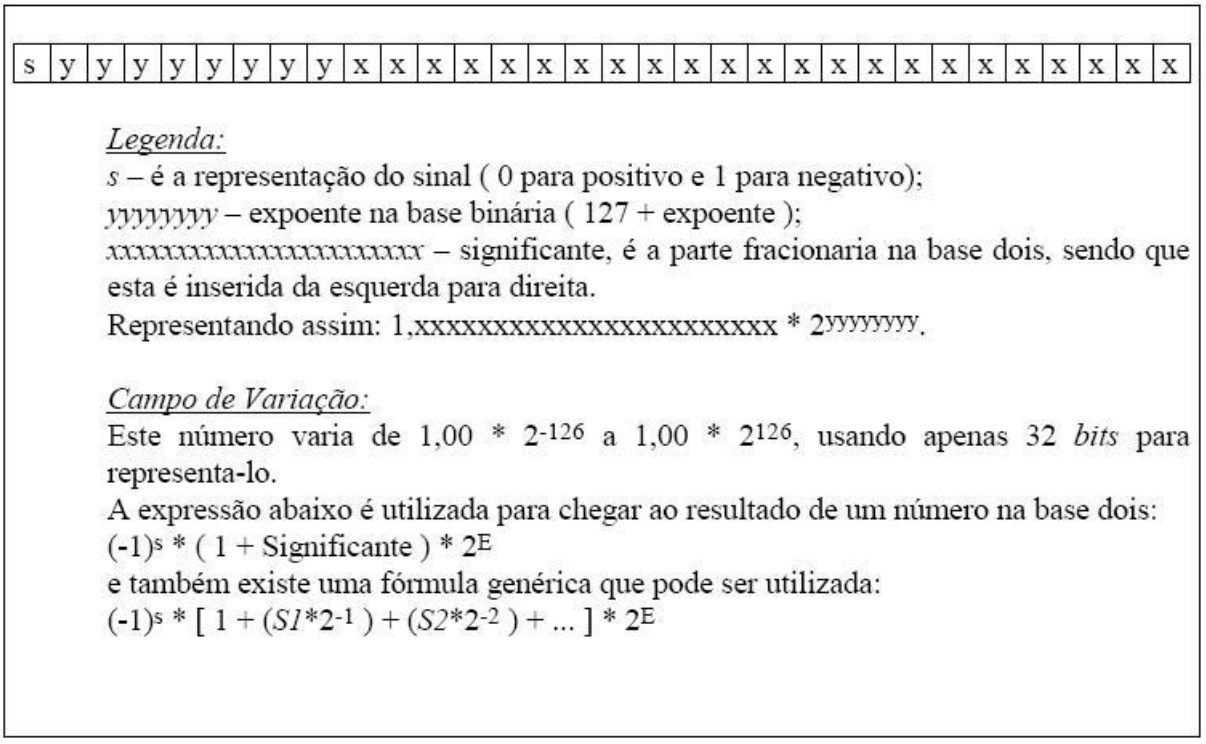

Figura 8.3: Formato binário de um Número em Ponto Flutuante [12]

O padrão IEEE-754 padroniza o tamanho de bits da mantissa e do expoente para a representação de número em ponto flutuante. Também estão reservados outros dois formatos estendidos para ponto flutuante: 'simples estendido' e 'duplo estendido', com 43 bits ou mais e com 79 bits ou mais, respectivamente.

Um dos problemas tradicionais com números em ponto flutuante está na dificuldade de se lidar com *underflow*, *overflow* e números não-inicializados. O padrão IEEE lida com estes problemas explicitamente. Em números não-normalizados, o bit "1"implícito à esquerda do ponto binário agora se torna um "0". Números não-normalizados podem distinguir-se dos normalizados, pois não é permitido aos números normalizados ter o expoente de zeros.

Dois zeros estão presentes neste esquema, positivo e negativo determinado pelo bit de sinal. Ambos possuem o expoente e a mantissa igual a zero. Aqui também o bit à esquerda do ponto binário é implicitamente zero em vez de um.

Quando ocorre um *overflow* é fornecida uma representação especial para infinito, que consiste em um expoente com todos os bits em "1"(não permitido para números normalizados) e uma mantissa igual a zero.

Este número pode ser usado como operando que se comporta como as regras matemáticas usuais de infinito. Por exemplo, infinito mais qualquer coisa é infinito. Similarmente, qualquer número dividido por zero resulta em infinito. Para o tratamento

de operações cujo resultado é indefinido, infinito dividido por infinito, por exemplo, usa-se outro formato especial, denominado NaN (Not a Number). Este pode também ser tratado como operando com resultados previsíveis.

8.3 Instruções personalizadas para o Nios II

Com as instruções personalizadas no processador Nios II é possível tirar proveito da flexibilidade das FPGAs atendendo-se os requerimentos de desempenho do sistema. Instruções personalizadas permitem ao projetista adicionar funcionalidades personalizadas à unidade lógica e aritmética do processador Nios II. A figura 8.4 ilustra a ULA do Nios II com o bloco lógico de instruções personalizadas.

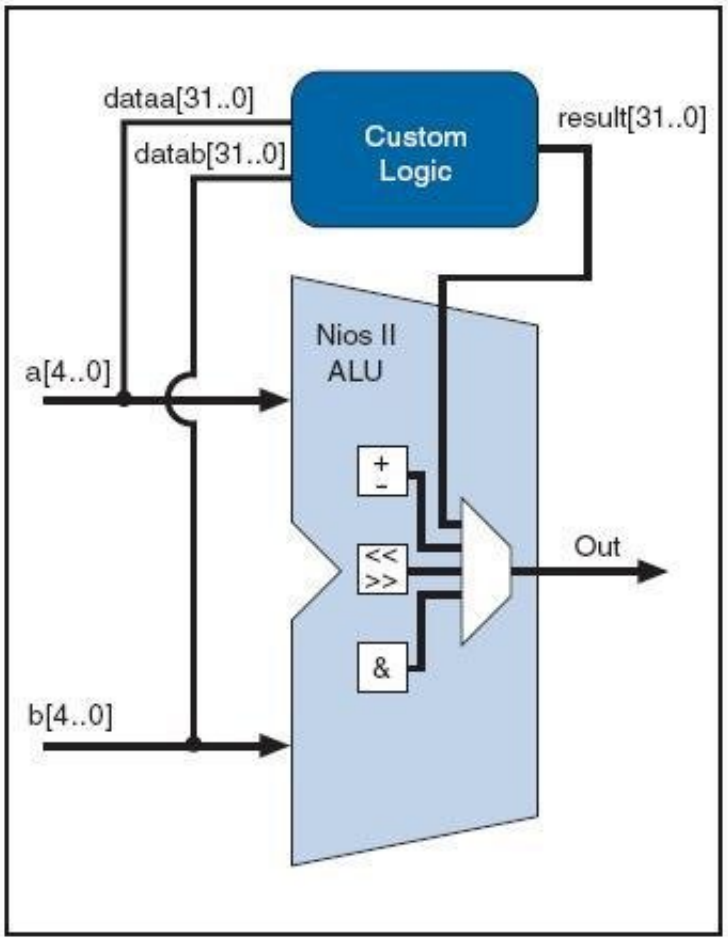

Figura 8.4: Diagrama de blocos da ULA do Nios II [10]

A operação básica do bloco lógico de instrução personalizada do Nios II é receber entrada através de dataa[31..0] e/ou datab[31..0] e dirigir o resultado para a porta result[31..0].

A figura 8.5 ilustra um diagrama de blocos de hardware de uma instrução personalizada no processador Nios II.

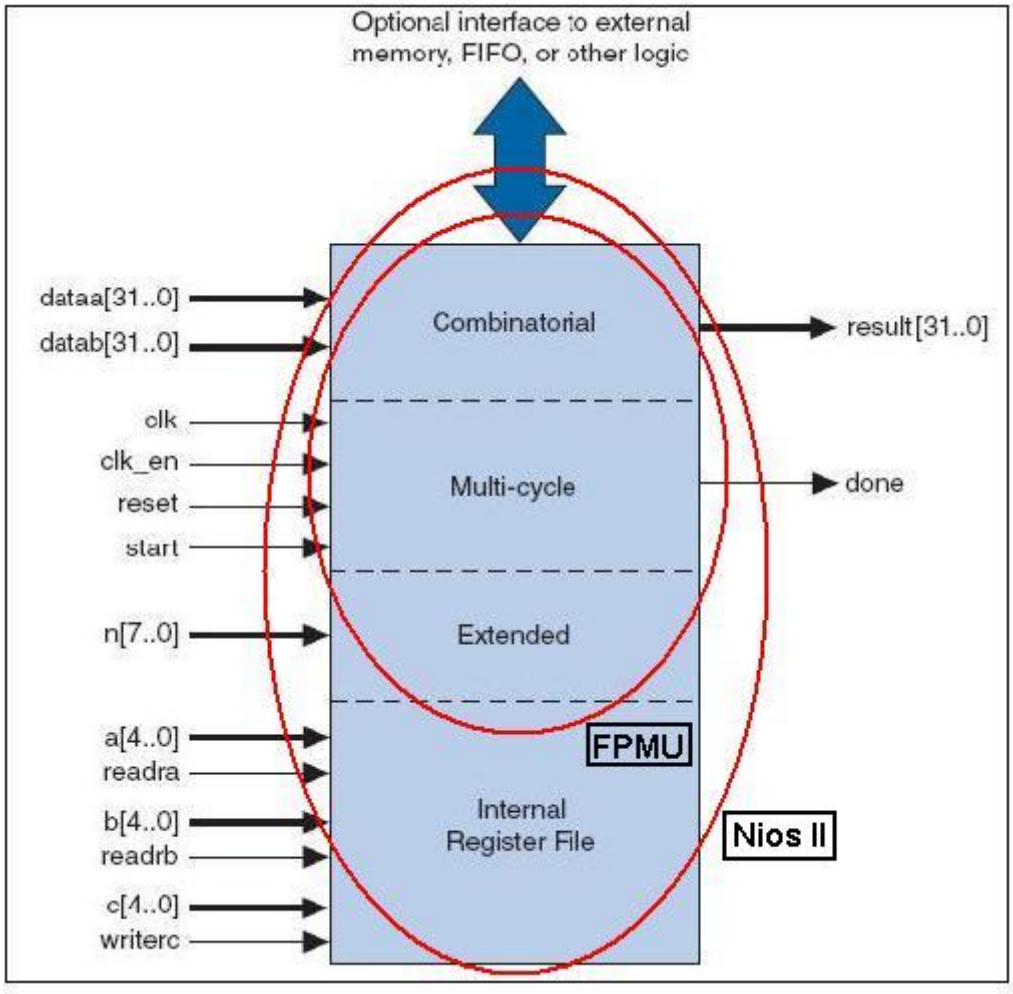

Figura 8.5: Diagrama de blocos do hardware do Nios II [10]

O processador Nios II suporta diferentes tipos de arquiteturas de instruções personalizadas. A figura 8.5 lista os sinais adicionais que acomodam diferentes tipos de arquiteturas. Apenas as portas utilizadas para o tipo específico de cada arquitetura são necessárias. Neste projeto utilizou-se a arquitetura do tipo estendia (*Exetended*).

8.3.1 Implementação da FPMU em VDHL

Conforme ilustrado na seção 8.3, a FPMU utiliza a arquitetura do tipo estendia das instruções personalizadas do processador Nios II. O trecho de código em linguagem de descrição de hardware, VHDL, na figura 9.15 (Apêndice) mostra a implementação do módulo de nível mais alto da FPMU que define o componente a ser utilizado pelo processador Nios II.

No trecho de código ilustrado na figura 9.15, observa-se a presença de todas as portas presentes no diagrama Extended da figura 8.5. O processo de execução da FPMU pode ser observado no trecho de código em VHDL na figura 9.16 (Apêndice):

O sinal "b_instruct"recebe o código de seleção de operação através de "n", os sinais

"b_fp_a"e "b_fp_b"recebem os operadores dataa[31..0] e datab[31..0] respectivamente. Ao final da operação, o sinal "done"é setado para "1"e o resultado é obtido através de "result".

8.3.2 Uso da FPMU como instrução personalizadas no Nios II

Utilizando-se o ferramenta SOPC Builder da Altera, a FPMU foi integrada ao processador Nios II conforme pode ser observado na figura 9.17 e 9.18 (Apêndice). É possível observar na figura 9.17 que todas as portas especificadas em VHDL foram corretamente identificadas pelo SOPC Builder.

O hardware elaborado (*designed*) para ser utilizado com o processador Nios II (figura 8.6) é composto pelos módulos listados a seguir.

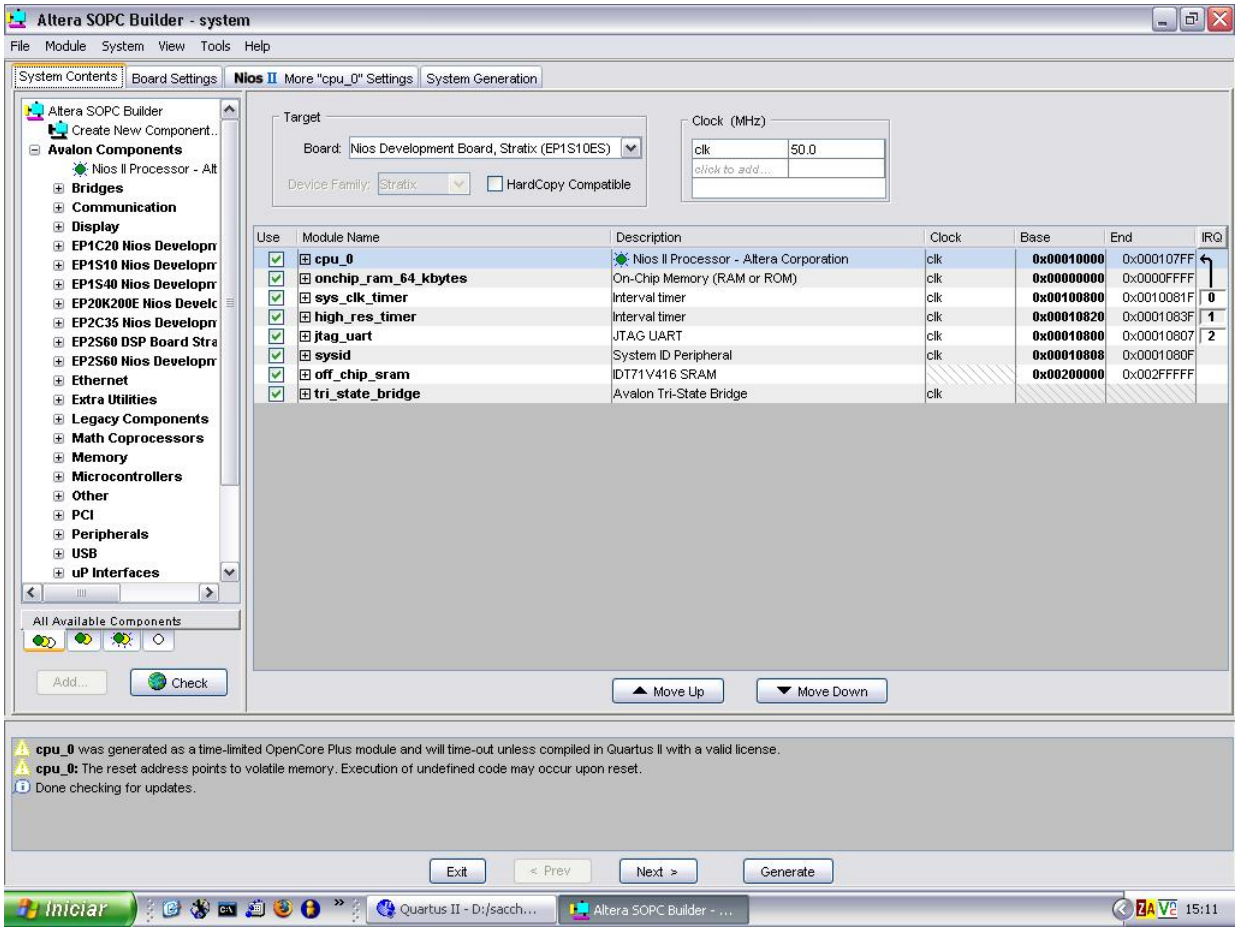

Figura 8.6: Composição do hardware através da ferramenta SOPC Builder utilizado em conjunto com o processador Nios II

- **cpu_0 :** módulo correspondente ao processador Nios II;
- **on_chip_ram_64kbytes :** memória RAM de 64 Kbytes;
- **sys_clk_timer :** *clock* do sistema;
- **high_res_timer :** contador de tempo utilizado para reinicializar o hardware;

- **jtag_uart :** módulo responsável pela comunicação entre a placa Stratix e o computador (PC) utilizado para baixar o FPGA para a placa;
- **sysid :** identificador de periféricos utilizado pelo processador Nios II;
- **off_chip_sram :** memória SRAM de 1 Mbyte utilizada para armazenar os dados de entrada e saída;
- **tri_state_bridge :** *bridge* de comunicação para o barramento Avalon.

8.3.3 Utilização da FPMU pelo algoritmo de localização

As funções "normal()"e "calc_mean()"foram alteradas para executarem diretamente em hardware as operações com ponto flutuante, que eram responsáveis por grande consumo de recursos de execução do programa.

A utilização da FMPU pelo programa é ilustrada pelos trechos de código fonte abaixo. O primeiro trecho mostra como uma operação de ponto flutuante era realizada em software pelo programa original. O trecho de código subsequente ilustra a mesma operação com a utilização da FPMU.

```
mean_x+=part[i].x;
```

```
mean_x = ALT_CI_FPMU(ADD_OPCODE,mean_x,part[i].x);
```

8.4 Execução em FPGA

Para executar os algoritmos descritos no capítulo 7 foram necessárias algumas modificações realizadas no código fonte original. Feitas as modificações, o código foi novamente compilado e executado. Após a fase de execução, foi efetuada uma análise do desempenho através da ferramenta nios2-elf-gprof.

8.4.1 Metodologia

Desde o seu início, o trabalho foi desenvolvido baseado em uma metodologia que divide todo o processo de *codesign* em 5 etapas distintas [59]:

1. **Traçar o perfil do código:** refrente ao capítulo 7, onde através da ferramenta gprof foram levantados os tempos de execução de cada função utilizada pelo programa;
2. **Identificar laços internos críticos:** também no capítulo 7 foram identificados os laços internos críticos das funções mais custosas (figura 7.4 no capítulo 7, e figuras 9.4 e 9.5 no Apêndice A);

3. **Criar instruções lógicas personalizadas (customizadas):** as operações em aritmética de ponto flutuante presentes nos laços identificados anteriormente, foram portadas para executar em hardware, conforme a seção 8.3 deste capítulo (figura 8.5);

4. **Importar a instrução lógica personalizada no projeto:** utilizando-se a ferramenta SOPC Builder da Altera, as instruções lógicas personalizadas foram importadas ao projeto (figruas 9.17 e 9.18 no Apêndice A);

5. **Chamar a instrução personalizada no código C ou Assembly:** Neste projeto não foi utilizada linguagem Assembly. A chamada para instrução personalizada em C pode ser verificada na seção 8.3.3 deste capítulo.

Basicamente, esta metodologia se divide em duas etapas de maior grandeza, sendo elas: uso de ferramentas de *profiling* e identificação de partes críticas do código, referente às duas primeiras etapas do processo; e criação e utilização de instruções personalizadas, conforme as 3 estapas finais. Essa divisão em duas macro-etapas corresponde, respectivamente, aos capítulos 7 (*Análise da Implementação do Algoritmo de Localização*) e 8 (*Implementação dos Algoritmos de Localização e Mapeamento em FPGA*) deste trabalho.

8.4.2 *Scan Matching* utilizando a bibiloteca omap do pacote pmap

O pacote pmap prove várias bibliotecas e utilitários para mapeamento SLAM baseado em laser. Os quatro componentes principais são:

- LODO: biblioteca para dados de laser estabilizados e odometria. Essa biblioteca utiliza um algoritmo SLAM incremental para corrigir a estimação de posição;
- PMAP: biblioteca para mapeamento baseada em filtro de partículas mantidos sobre possíveis mapas que geram um mapa final correto topologicamente mas desalinhado;
- RMAP: biblioteca para relaxamento das restrições locais que utiliza um algoritmo para alinhar e corrigir topologicamente o mapa;
- OMAP: biblioteca para mapeamento *occupancy grid* que converte um conjunto de leituras de lasers em mapa.

Neste trabalho foi utilizado somente a biblioteca OMAP para gerar os mapas a partir das leituras de lasers, já que o algoritmo desenvolvido por Denis Wolf é responsável apenas pela localização. O pacote pmap completo, com os quatro componentes listados acima, já é uma implementação completa do algoritmo SLAM, mas neste trabalho optou-se por compor uma nova solução de localização e mapeamento.

Essa nova solução foi obtida pela execução em conjunto do algorimto de localização através do filtro de partículas detalhado no capítulo 4 com o mapeamento por *occupacy grid* utilizando o OMAP, do pacote pmap [16]. A seção a seguir descreve esssa tarefa.

8.4.3 Alterações realizadas nos algoritmos para execução no Nios II

No programa original os dados de entrada, referentes à leitura dos lasers do robô, eram fornecido através de um arquivo ("log.txt") com um tamanho aproximado de 80M. Devido a restrições de tamanho por parte da FPGA utilizada, uma placa Stratix da Altera, não era possível fornecer o arquivo de log ao programa em execução.

Para contornar este problema, os dados de entrada do arquivo "log.txt"foram incorporados ao código fonte do programa. Dessa forma, a função "update_data_file()"realiza uma simples atribuição de variáveis ao invés de abrir uma arquivo de 80M para obter valores a serem lidos.

Similarmente à função "update_data_file()", "load_map()"que abria o arquivo "map3.map"e "update_laser_ang()"que abria o arquivo "laser_ang.txt"também foram alteradas para realizarem simples atribuições de variáveis. Assim, o problema de espaço para armazenamento dos arquivos de entrada, e difícil acesso à arquivos para leitura de dados por parte da FPGA utilizada foram resolvidos.

A utilização de um laser *range finder* SICK está sendo implementada para que captura de dados em tempo real nos experimentos a serem realizados no campus da USP de São Carlos. A Figura 8.7 ilustra o laser conectado à placa de FPGA Stratix da Altera. Dessa forma, é obtida uma solução embarcada funcional, e será possível analisar o desempenho da execução em tempo real desta solução do algoritmo pra navegação de robôs móveis.

Problemas em relação à qualidade dos dados capturados por sensores, tais como laseres, são bem conhecidos na literatura atual [56], entretanto, tal análise não foi incorporada ao escopo do trabalho.

Uma outra restrição em relação ao espaço de memória disponível, constatada após as alterações até aqui descritas, é relativa às estruturas de dados utilizadas para o armazenamento dos dados a serem processados pelo programa. Os registros, vetores e matrizes que armazem os mapas, dados de odometria, dados da leitura de laser e demais dados de entrada, necessitavam de um espaço de memória de uma magnitude muito maior que a disponível pela FPGA Stratix II. O erro reportado pela ferramenta Nios II IDE é ilustrado abaixo.

```
cc1: out of memory allocating 4072 bytes after a total of 537763840 bytes
make: *** [obj/data_file.o] Error 1
```

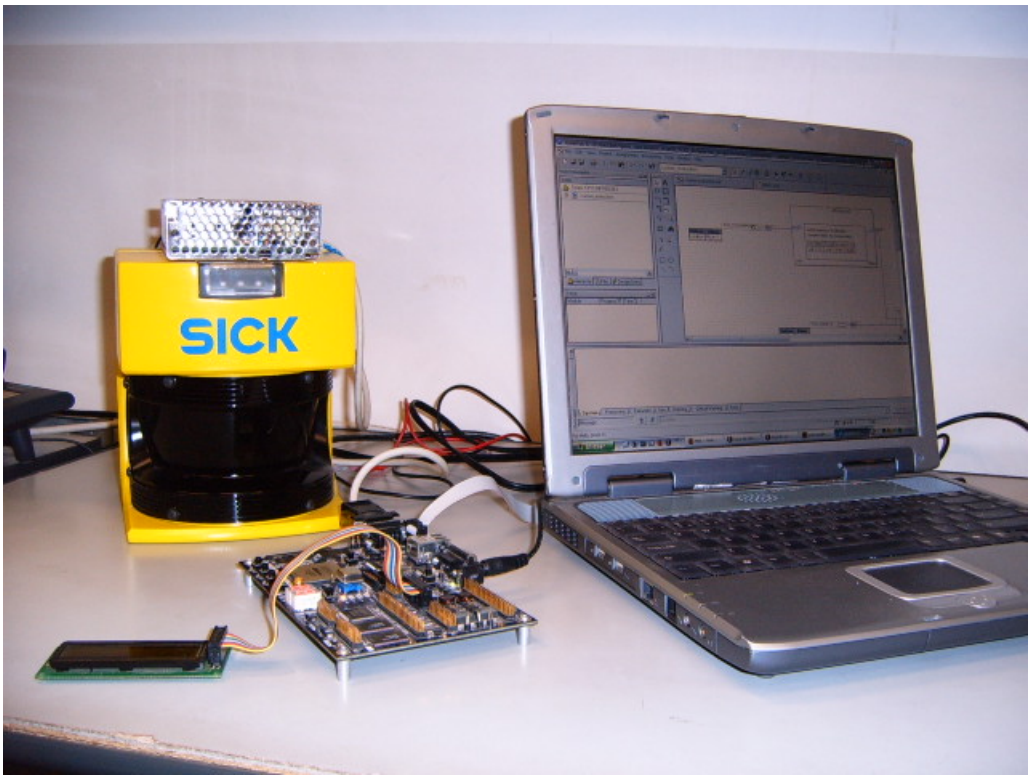

Figura 8.7: Laser *range finder* SICK conectado à porta serial da placa FPGA Stratix para execução da solução embarcada no processador Nios II

Este contratempo foi superado pela alteração das estruturas de dados definidas no arquivo "defs.h". Em detrimento de uma maior precisão nos resultados obtidos, o tamanho do mapa, vetores com dados de entrada e número de partículas foram reduzidos. Para grantir que análise posterior do novo código obtido, não distoasse muito da análise previamente realizada e descrita no capítulo 7, o novo código, portado para atender as restrições da FPGA, foi novamente compilado, executado e analisado no Linux através da ferramenta gprof.

O computador utilizado para esses novos experimentos foi uma máquina com processador AMD Athlon(TM) XP 2000+, com 1662.622MHz, 256 KB de memória cache, 1G de memória RAM e com sistema operacional Mandrake 10.1, e kernel 2.6.8.1-12mdk. Os dois perfis planos das tabelas 8.1 e 8.2 ilustram o resultado obtido pelas análises realizadas.

A Tabela 8.3 compara: a porcentagem de tempo gasto pelas três funções na solução embarcada que foram modificadas para executar operações de ponto flutuante em hardware; com a porcentagem de tempo gasto na execução da implementação original dessas funções. Dessa forma, a comparação é realizada de forma justa, mostrando que na solução embarcada o tempo de execução das funções é proporcionalmente menor, além da questão de consumo de energia, que no caso do Nios II é bem menor que um processador de PC comum.

O mesmo processo de análise foi realizado para o algoritmo de *scan matching*.

Função	% tempo	tempo (s)	n chamadas
rand_position()	0.41	0.24	100
rand()	0.26	0.15	
normal()	0.24	0.14	9900
calc_lh_map()	0.21	0.12	1
floor()	0.20	0.12	
do_action_model2()	0.19	0.11	9900
sample()	0.08	0.05	9900
run_filter()	0.06	0.03	1
calc_mean()	0.04	0.02	99
update_data_file()	0.01	0.01	100
update_laser_ang()	0.00	0.00	1
arred()	0.00	0.00	1121
gettimeofday()	0.00	0.00	793
clear_map()	0.00	0.00	1
round__angle()	0.00	0.00	198
log_data()	0.00	0.00	
init_particles()	0.00	0.00	1
load_map()	0.00	0.00	1
main()	0.00	0.00	1

Tabela 8.1: Perfil plano gerado pelo nios2-elf-gprof em FPGA

Função	% tempo	tempo (s)	n chamadas
rand_position()	50.00	0.11	100
normal()	18.18	0.04	9900
run_filter()	9.09	0.02	1
sample()	4.55	0.01	9137
do_action_model2()	0.00	0.00	9900
arred()	0.00	0.00	1191
round_angle()	0.00	0.00	198
update_data_file()	0.00	0.00	100
calc_mean()	0.00	0.00	99
log_data()	0.00	0.00	99
clear_map()	0.00	0.00	1
init_particles()	0.00	0.00	1
load_map()	0.00	0.00	1
print_map()	0.00	0.00	1
update_laser_ang()	0.00	0.00	1

Tabela 8.2: Perfil plano gerado pelo gprof na execução em sistema Linux

Função	Denis Wolf		Embarcado	
	% tempo	*tempo (s)*	*% tempo*	*tempo (s)*
normal()	45.16%	0.04s	0.24%	0.14s
sample()	18.89%	0.01s	0.08%	0.05s
calc_mean()	16.23%	< 0.01s	0.04%	0.02s

Tabela 8.3: Comparação do desempenho da implementação do algoritmo de filtro de partículas em software utilizando PC comum, com a implementação em hardware utilizando o processador Nios II da Altera

Assim, foram analisadas quanto a porcetagem e tempo de execução, as três principais funções da biblioteca omap (*occupancy grid mapping*) do pacote pmap. Na tabela 8.4 podemos observar o mesmo comportamento ilustrado na tabela anterior (8.3) onde o tempo de execução da solução embarcada se mostra porporcionalmente inferior.

Função	Andrew Howard		Embarcado	
	% tempo	*tempo (s)*	*% tempo*	*tempo (s)*
omap_add()	84.62%	0.11s	0.64%	0.30s
pose2_add_pos()	15.38%	0.02s	0.18%	0.10s
omap_alloc()	<0.01%	< 0.01s	0.04%	0.02s

Tabela 8.4: Comparação do desempenho da implementação do algoritmo *scan matching* em software utilizando PC comum, com a implementação em hardware utilizando o processador Nios II da Altera

O resultado obtido e ilustrado pelas tabelas 8.3 e 8.4 mostra o sucesso do *codesign* realizado por esse trabalho, pois com o ganho em relação a porcentagem de tempo de execução da solução embarcada em relação às originais, o objetivo inicial foi alcançado. A figura 9.19 (Apêndice) ilustra o experimento que gerou esses resultados em execução na ferramenta da Altera Nios II IDE.

8.4.4 Considerações Finais

Os algoritmos estão embarcados, porém a validação de seu perfeito funcionamento corrente, será possível com os dados de odometria que são gerados pelo robô 3DX Pioneer (figura 8.8) já adquirido pelo LCR, mas que ainda não chegou a USP. Entretanto, o pesquisador Denis Wolf fornecerá um log com todos os dados capturados por robô do mesmo modelo adquirido pela USP, possibilitando uma validação dos testes realizados neste trabalho.

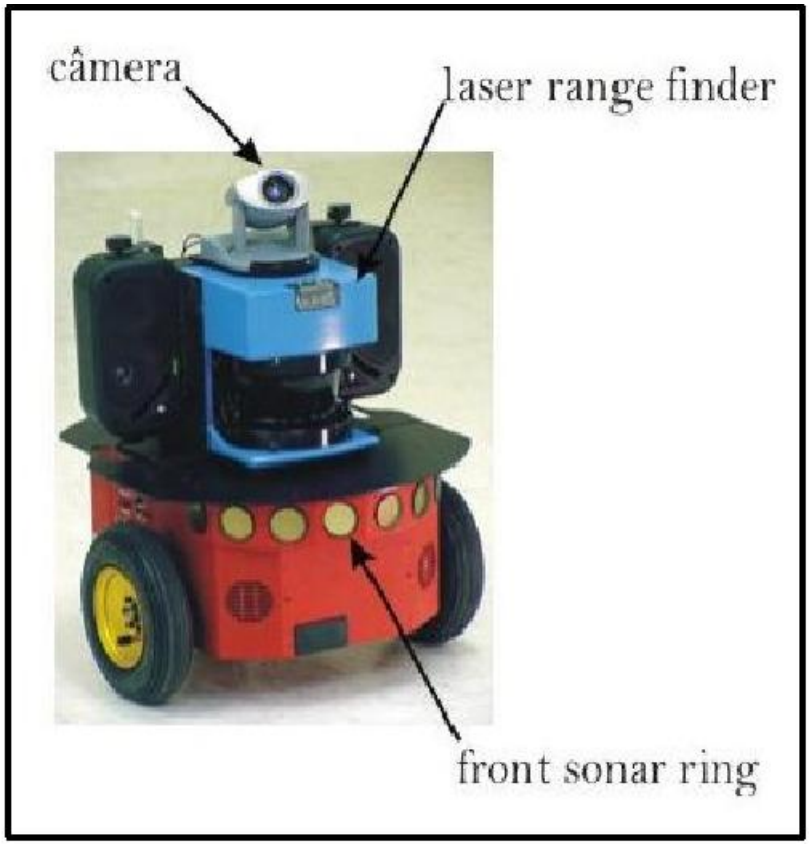

Figura 8.8: Robô Pioneer 3DX [3]

Capítulo

9

Conclusão

Este trabalho apresentou como, manualmente, implementações em linguagem de alto nível C para computadores pessoais (PC), foram portadas para serem executadas em hardware reconfigurável, visando aplicações de robótica móvel. Foram mostradas análises dos algoritmos em C para identificar porções computacionalmente custosas do programa para serem executadas em hardware.

A solução em hardware, apresenta melhorias significativas em relação a:

- desempenho;
- custo, já que na solução embarcada não é necessário um notebook acoplado ao robô, o que é fundamental para a queda de preço de robôs pessoais de consumo;
- consumo de energia, o que é essencial para atender as restrições de desenvolvimento de sistemas embarcados;
- abordagem utilizada, pois um software para PC comum foi modificado para uma solução embarcada que é mais adequada para a robótica móvel.

A tarefa de analisar e portar manualmente um algoritmo em C para uma solução em FPGA demonstrou-se extremamente trabalhosa, porém para o desenvolvimento da robótica no futuro, é essencial que as soluções sejam embarcadas para viabilizar a comercialização de robôs pessoais em larga escala. Dentro deste contexto, o laboratório LCR também está desenvolvendo uma ferramenta automática para a síntese de hardware embarcado denominada ARCHTECT+ [15], visando uma redução significativa no esforço de desenvolvimento de trabalhos deste gênero. Todo conhecimento adquirido pela realização desse trabalho servirá como referência para o desenvolvimento do ARCHTECT+.

9.1 Trabalhos Futuros

Alguns trabalhos futuros que podem ser realizados para a continuidade desta pesquisa são:

- Colocar a unidade de ponto flutuante FPMU como um componente externo ao processador e não como instrução personalizada do Nios II como é feito atualmente. Dessa forma, a FPMU pode ser utilizada como um *core* pronto, o que possibilita a sua fácil utilização em qualquer outro tipo de projeto que envolva aritmética de ponto flutuante;
- Realizar testes exaustivos em campo com o robô Pioneer 3DX conforme mencionado no capítulo 8;
- Implementar uma solução embarcada integralmente SLAM, seguindo a mesma metodologia deste trabalho em relação aos algoritmos de localização e mapeamento.
- Comparar a solução SLAM com a desenvolvida neste trabalho.

Referências Bibliográficas

[1] WOLF, D.; SUKHATME, G. Online simultaneous localization and mapping in dynamic environments. In: *ICRA 04. 2004 IEEE International Conference on Robotics and Automation*. [S.l.]: IEEE, 2004. v. 2, p. 1301–1306.

[2] EUROPE, U. N. E. C. for. *World Robotic*. http://www.unece.org/press/pr2004/04stat$_p$01e.*pdf*, 20

[3] BONATO, V. *Exame de Qualificação de Doutorado: Um Sistema de Mapeamento e Localização Simultâneos Usando Multi-câmeras para Robôs Móveis*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação ICMC-USP, 2005.

[4] JUNIOR, V. G. *Sistema de Visão Omnidirecional Aplicado no Controle de Robôs Móveis*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo POLI-USP, 2002.

[5] BIANCHI, R. E. *Sistema de navegação de robôs móveis autônomos para o transporte de documentos*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação ICMC-USP, 2002.

[6] THRUN, S.; BURGARD, W.; FOX, D. *Probabilistc Robotics*. [S.l.]: Addison-Wesley Pub Co, 2005. 1-480 p.

[7] THRUN, S. Probabilistic algorithm in robotics. *IEEE Transactions On Robotics And Automation*, Carnegie Mellon, p. 278–288, 2000.

[8] HOWARD, A.; WOLF, D.; SUKHATME, G. Towards 3d mapping in large urban environments. In: *Preceedings of 2004 IEEE/RSJ International Conference on Inteligent Robots and Systems*. [S.l.]: IEEE, 2004. p. 419–424.

[9] COMPTON, K.; HAUCK, S. Reconfigurable computing: A survey of systems and software. In: *ACM Computing Surveys*. [S.l.]: ACM, 2002. v. 34, n. 2, p. 171–210.

[10] DATA Book. http://www.altera.com, 1998. Altera Corporation, San Jose, CA.

[11] NIOS II Core. http://www.altera.com/literature/hb/nios2/n2cpu$_n$$ii$51002.*pdf*, 2005.

[12] RODRIGUES, M. I. *FPMU Ambiente de Desenvolvimento de Unidades de Aritmética Binária em Ponto Flutuante Baseada em Computação Reconfigurável*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas and de Computação ICMC-USP, 2002.

[13] WOLF, D.; SUKHATME, G. Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, 2005.

[14] WOLF, D. F. *Desenvolvimento de Algoritmos para Robôs Móveis - Projeto de Pesquisa para Bolsa Recém-Doutor da FAPESP*. [S.l.], novembro 2005.

[15] GONçALVES, R. A. et al. Architect-r: A system for reconfigurables robots design. In: ACM. *ACM Symposium on Applied Computing (SAC 2003)*. Melbourne, Florida, EUA,: ACM Press, 2003. p. 679–683.

[16] HOWARD, A. *PMAP - Simple Mapping Utilities*. http://www-robotics.usc.edu/ ahoward/pmap, 2005.

[17] THRUN, S. Bayesian landmark learning for mobile robot localization. Carnegie Mellon University, p. 33–41, 1998.

[18] GONçALVES, R. A. *ARCHITECT-R: uma ferramenta para o desenvolvimento de robôs móveis reconfiguráveis*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas and de Computação ICMC-USP, 2002.

[19] GPROF online documentaion. *http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html$_m$ono/gprof.html*.

[20] SALANT, M. A. *Introdução à Robótica*. [S.l.]: McGraw Hill, 1990.

[21] MEZENCIO, R. *Implementação do Método de Campos Potenciais para Navegação de Robôs Móveis Baseada em Computação Reconfigurável*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação ICMC-USP, 2002.

[22] JUNIOR, J. O.; JUNIOR, V. G. Visual servo control of a mobile robot using omni-directional vision. In: UNIVERSITY OF TWENTE, NETHERLANDS. *Proceedings of Mechatronics 2002*. [S.l.], 2002. p. 413–422.

[23] EVERETT, H. R. *Sensors for Mobile Robots*: Theory and application. [S.l.]: MIT Press, 1995. 528 p.

[24] MURPHY, R. *Introduction to AI Robotics*. [S.l.]: MIT, 2000.

[25] SHANAHAN, M. Reinventing shakey. In: *Workshop on Logic-Based Artificial Intelligence*. [S.l.]: Computer Science Department, University of Maryland, 1999. p. 1–9.

[26] GOLDBERG, K. et al. *Algorithmic Foundations of Robotics*. [S.l.: s.n.], 1995. 528 p.

[27] BROOKS, R. *Intelligence Without Reason*. http://people.csail.mit.edu/u/b/brooks/public$_h$*tml/*
1293.*pdf*, *abril*1991.*A.I.MemoNo*.1293.

[28] ARKIN, R. C. *Behavior-Based Robotics*. [S.l.]: MIT Press, 2000. 490 p.

[29] GOWDY, J. *Emergent Architectures: A Case Study for Outdoor Mobile Robots*. Tese (Doutorado) — Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2000.

[30] MANIERE, E.-C.; SIMMONS, R. Architecture, the backbone of robotics systems. *ICRA*, ICRA, p. 67–72, 2000.

[31] THRUN, S. et al. Robust monte carlo localization for mobile robots. 2000.

[32] KORTENKAMP, D.; BONASSO, R. P.; MURPHY, R. *Artificial Intelligence and Mobile Robots: Case Studies of Sucessful Robot Systems*: Case of studies of successful robot systems. [S.l.]: AAAI / MIT Press, 1998. 389 p.

[33] COX, I. J.; WILFONG, G. T. Autonomous robot vehicles. *Springer Verlag*, 1990.

[34] LEE, D. *The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Robot*: Distinguished dissertations in computer science. [S.l.]: Cambridge University Peters, 1996.

[35] FOX, D. et al. Position estimation for mobile robots in dynamic environments. *AAAI/IAAI*, p. 983–988, 1998.

[36] FOX, D.; BURGARD, W.; THRUN, S. Active markov localization for mobile robots. *Robotics an Autonomous Systems*, v. 25, p. 195–207, 1998.

[37] OLSON, C. F. Probabilistic self-localization for mobile robots. *IEEE Transactions On Robotics And Automation*, v. 16, p. 55–66, 2000.

[38] SCATENA, J. M. *Implementação de Mapas Topológicos para Navegação de Robôs Móveis baseada em Computação Reconfigurável*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação ICMC-USP, 2003.

[39] BORENSTEIN, J.; KOREN, Y. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions On Robotics And Automation*, IEEE, v. 7, p. 278–288, 1991.

[40] THRUN, S. Learning metric topological maps for indoor mobile robot navigation. v. 99, p. 21–71, 1998.

[41] ELFES, A. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, v. 22, p. 46–57, 1989.

[42] FOLEY, J. D. et al. *Computer Graphics: Principles and Practice*. [S.l.]: Addison-Wesley Pub Co, 1995.

[43] WOLF, D.; SUKHATME, G. Towards mapping dynamic environments. In: *ICRA 03. International Conference on Advanced Robotics*. [S.l.]: IEEE, 2003. p. 594–600.

[44] GERKEY, B. P.; VAUGHAN, R. T.; HOWARD, A. *Player Version 1.5*. http://playerstage.sourceforge.net, junho 2004.

[45] GERKEY, B. P.; VAUGHAN, R. T.; HOWARD, A. *Stage Version 1.3.3*. http://playerstage.sourceforge.net, dezembro 2004.

[46] THE Programmable Logic Data Book. http://www.xilinx.com, 1994. XILINX, INC San Jose, CA.

[47] CARDOSO, J. M. P.; VESTIA, M. P. Architectures and compilers to support reconfigurable computing. In: *ACM Student Magazine*. [S.l.: s.n.], 1999.

[48] BROWN, S. et al. *Field Programmable Gate Arrays*. [S.l.]: Kluwer Academic Publishers, 1992.

[49] ABOUZEID, P. et al. Input-driven partitioning methods and application to synthesis on table-lookupbased fpgas. In: *IEEE Trans. Comput. Aid. Des.Integ. Circ. Syst.* [S.l.: s.n.], 1993. v. 12, n. 7, p. 913–925.

[50] CONG, J.; HWANG, Y.-Y. Boolean matching for complex plbs in lut-based fpgas with application to architecture evaluation. In: *ACM/SIGDA International Symposium on FPGAs*. [S.l.]: ACM/SIGDA, 1998. p. 27–34.

[51] GOKHALE, M.; STONE, J. Napa c: Compiling for a hybrid risc/fpga architecture. In: *IEEE Symposium on Field-Programmable Custom Computing Machines*. [S.l.]: IEEE, 1998. p. 126–135.

[52] CHICHKOV, A. V.; ALMEIDA, C. B. An hardware/software partitioning algorithm for custom computing machines. In: *Lecture Notes in Computer Science 1304 Field Programmable Logic and Applications*. [S.l.: s.n.], 1998. p. 274–283.

[53] JAIN, R. *The Art of Computer System Performance Analysis*: techniques for experimental designs, measurement, simulation, and modeling. [S.l.]: John Wiley Sons, 1991.

[54] DELLAERT, F. et al. Monte carlo localization for mobile robots. In: *Proceedings of the IEEE International Conference on Robotics Automation*. [S.l.: s.n.], 1998.

[55] DELLAERT, F. et al. Robust monte carlo localization for mobile robots. In: *Artificial Intelligence*. [S.l.: s.n.], 2000.

[56] WEINGARTEN, J. W.; GRUENER, G.; GRUENER, G. A state-of-the-art 3d sensor for robot navigation. In: *55th International Astronautical Congress*. [S.l.: s.n.], 2004.

[57] SE, S. et al. Vision based modeling and localization for planetary exploration rovers. In: *55th International Astronautical Congress*. [S.l.: s.n.], 2004.

[58] RODRIGUES, M. I.; MARQUES, E. Fpmu - float point modular unit: Biblioteca de componentes para aritmética em ponto flutuante descrita em vhdl. In: *Anais ICMC*. [S.l.]: ICMC - USP, 2004. p. 1–11.

[59] CORPORATION, A. *Design with Nios II and SOPC Builder*. http://www.altera.com, julho 2004. Altera Corporation, San Jose, CA.

[60] WANG, C.; THORPE, C.; THRUN, S. Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowed urban areas. In: *ICRA 03. 2004 IEEE International Conference on Robotics and Automation*. [S.l.]: IEEE, 2003. p. 842–849.

[61] HAHNEL, D. et al. Mapping building with mobile robots in dynamic environments. In: *ICRA 03. 2004 IEEE International Conference on Robotics and Automation*. [S.l.]: IEEE, 2003. p. 1557–1563.

[62] GONçALVES, R. A. et al. Um sistema de computação reconfigurável. In: *In CORE2000 Workshop de Computação Reconfigurável*. [S.l.: s.n.], 2000.

[63] THRUN, S. A framework for programming embedded systems: Initial design and results. Carnegie Mellon, p. 1–45, 1998.

[64] THRUN, S. Towards programming tools for robotics that integrate probabilistic computation and learning. Carnegie Mellon, 2000.

[65] BORENSTEIN, J.; FENG, L. *Where Am I?*: Sensors and methods for mobile robot positioning. [S.l.]: University of Michigan and United States Departament of Energy's, 1996. 1-282 p.

[66] WOOD, R. G.; RUTENBAR, R. A. Fpga routing and routability estimation via boolean satisfiability. In: *ACM/SIGDA International Symposium on FPGAs*. [S.l.]: ACM/SIGDA, 1997. p. 119–125.

[67] DEHON, A. Fpga utilization and application. In: *ACM/SIGDA International Symposium on FPGAs*. [S.l.]: ACM/SIGDA, 1996. p. 115–121.

[68] TRIMBERGER, S. et al. A time-multiplexed fpga. In: *IEEE Symposium on Field-Programmable Custom Computing Machines*. [S.l.]: IEEE, 1997. p. 22–28.

[69] HAUCK, S.; BORRIELLO, G. Pin assignment for multi-fpga systems. In: *IEEE Trans. Comput. Aid. Desi. Integ. Circ. Syst.* [S.l.]: IEEE, 1997. v. 9, n. 16, p. 956–964.

[70] CADAMBI, S. et al. Managing pipeline reconfigurable fpgas. In: *ACM/SIGDA International Symposium on FPGAs*. [S.l.]: ACM/SIGDA, 1998. p. 55–64.

[71] RUPP, C. R. et al. The napa adaptive processing architecture. In: *IEEE Symposium on Field-Programmable Custom Computing Machines*. [S.l.]: IEEE, 1998. p. 28–37.

[72] CS2000 Advance Product Specification. http://www.electronicsweekly.com/Article20464.htm, 2000. CHAMELEON SYSTEMS, San Jose, CA.

[73] XC6200: Advance Product Specification. http://www.xilinx.com, 1996. XILINX, INC San Jose, CA.

[74] VIRTEXTM 2.5 V Field Programmable Gate Arrays: Advance Product Specification. http://www.xilinx.com, 1999. XILINX, INC San Jose, CA.

[75] TWO Flows for Partial Reconfiguration: Module Based or Difference Based. http://www.xilinx.com, setembro 2004. XAPP290 (v1.2) San Jose, CA.

[76] FPGA Data Book. http://www.lucent.com/press/0696/960603.mea.html, 1998. LUCENT TECHNOLOGIES Inc. Allentown, PA.

[77] EXCALIBUR backgrounder. http://www.altera.com/literature/an/an242.pdf, agosto 2000. Altera Corporation, San Jose, CA.

[78] CORPORATION, A. *Apex 20k*. http://www.altera.com/literature/hb/cfg/cfg$_c f$51005.*pdf*, *julho*

[79] NIOS II. http://www.altera.com/products/ip/processors/nios2/ni2-index.html, 2005.

[80] GONçALVES, R. A. *Proposta de uma ferramenta para auxílio ao desenvolvimento de hardware*. [S.l.], 2000. Technical Report ICMC-USP.

[81] GCC online documentaion. *www.gnu.org/software/gcc/onlinedocs*.

[82] THE Stanford SUIF Compiler Group. Suif Compiler system. *www.suif.stanford.edu*.

[83] ADITYA, S.; KATHAIL, V.; RAU, B. R. *Elcor's machine description system: Version 3.0*. [S.l.], outubro 1998. Hewlett Packard.

[84] CARDOSO, J. M. P. A novel algorithm combining temporal partioning and sharing of functional units. In: *IEEE Symposium on Field-Programmable Custom Computing Machines*. [S.l.]: IEEE, 2001.

[85] CARDOSO, J. M. P. *Compilação de Algoritmos em JAVA para Sistemas Computacionais Reconfiguráveis com Exploração de Paralelismo ao Nível das Operações*. Tese (Doutorado) — Universidade Técnica de Lisboa, 2002.

[86] THOMAS, G. An operating system for embedded systems. *Dr. Dobb's Journal*, ASME, v. 2, p. 238–241, 2000.

[87] MALDONADO, J. C.; MARQUES, E.; CARDOSO, J. M. P. Detalhamento de projeto architect+. *http://www.cnpq.br*, ASCIN/CNPq, p. 8–10, 2004.

Apêndice A

.

A seguir são apresentadas as figuras referentes às etapas de desenvolvimento deste projeto.

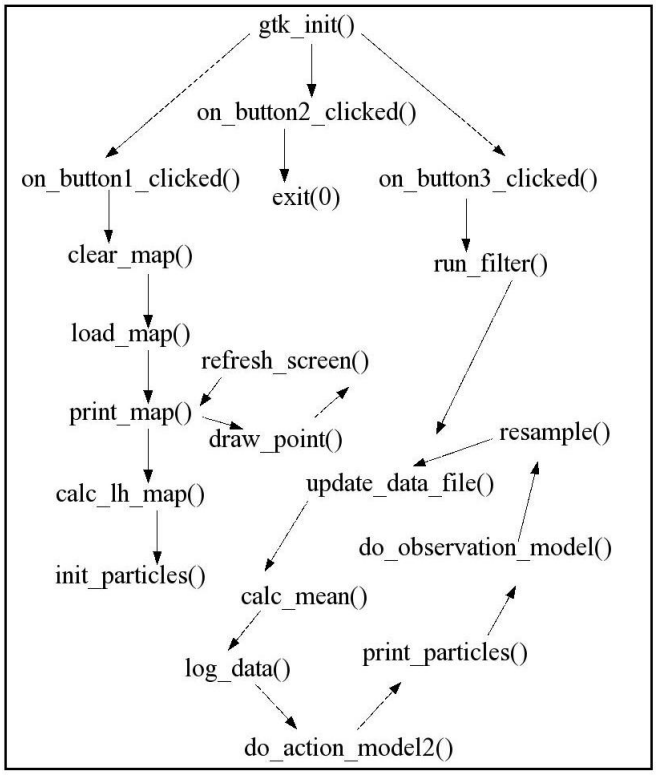

Figura 9.1: Diagrama do fluxo de execução do algoritmo de [1]

Figura 9.2: Interface do programa

```
void calc_mean()
{

register int i;
double mean_x=0, mean_y=0, mean_a=0, stdev_x=0, stdev_y=0, stdev_a=0;

for (i=0;i<num_particles;i++)
{
        mean_x+=part[i].x;
        mean_y+=part[i].y;
        mean_a+=part[i].a;
}

mean_x/=num_particles;
mean_y/=num_particles;
mean_a/=num_particles;

for (i=0;i<num_particles;i++)
{
        stdev_x+=fabsf(part[i].x-mean_x);
        stdev_y+=fabsf(part[i].y-mean_y);
        stdev_a+=fabsf(part[i].a-mean_a);
}

stdev_x/=num_particles;
stdev_y/=num_particles;
stdev_a/=num_particles;

glb_pos.mean_x=mean_x;
glb_pos.mean_y=mean_y;
glb_pos.mean_a=mean_a;

glb_pos.stdev_x=stdev_x;
glb_pos.stdev_y=stdev_y;
glb_pos.stdev_a=stdev_a;

}
```

Figura 9.3: Função "calc_mean()".

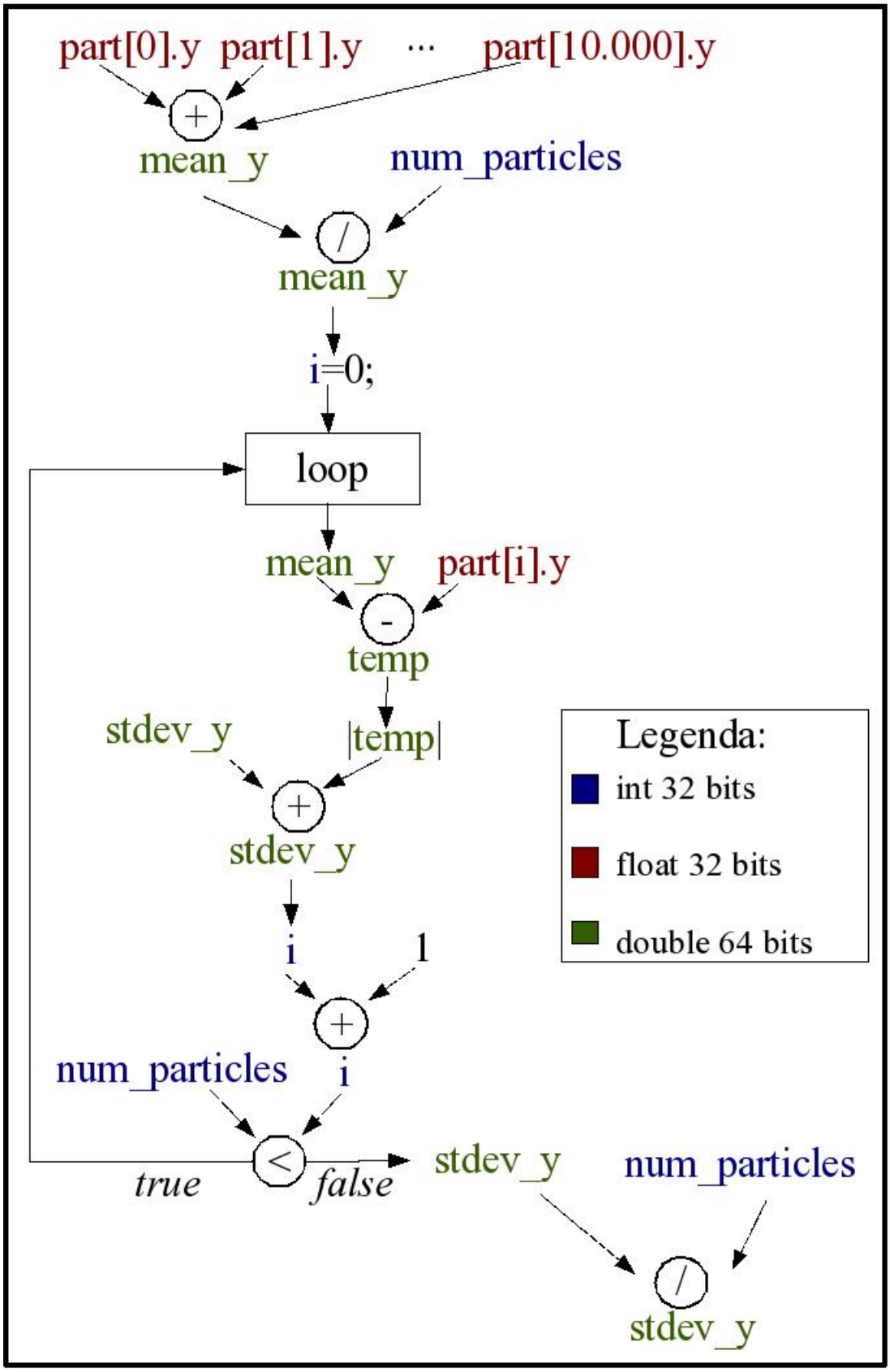

Figura 9.4: DFG da função "calc_mean()"em relação à coordenada Y

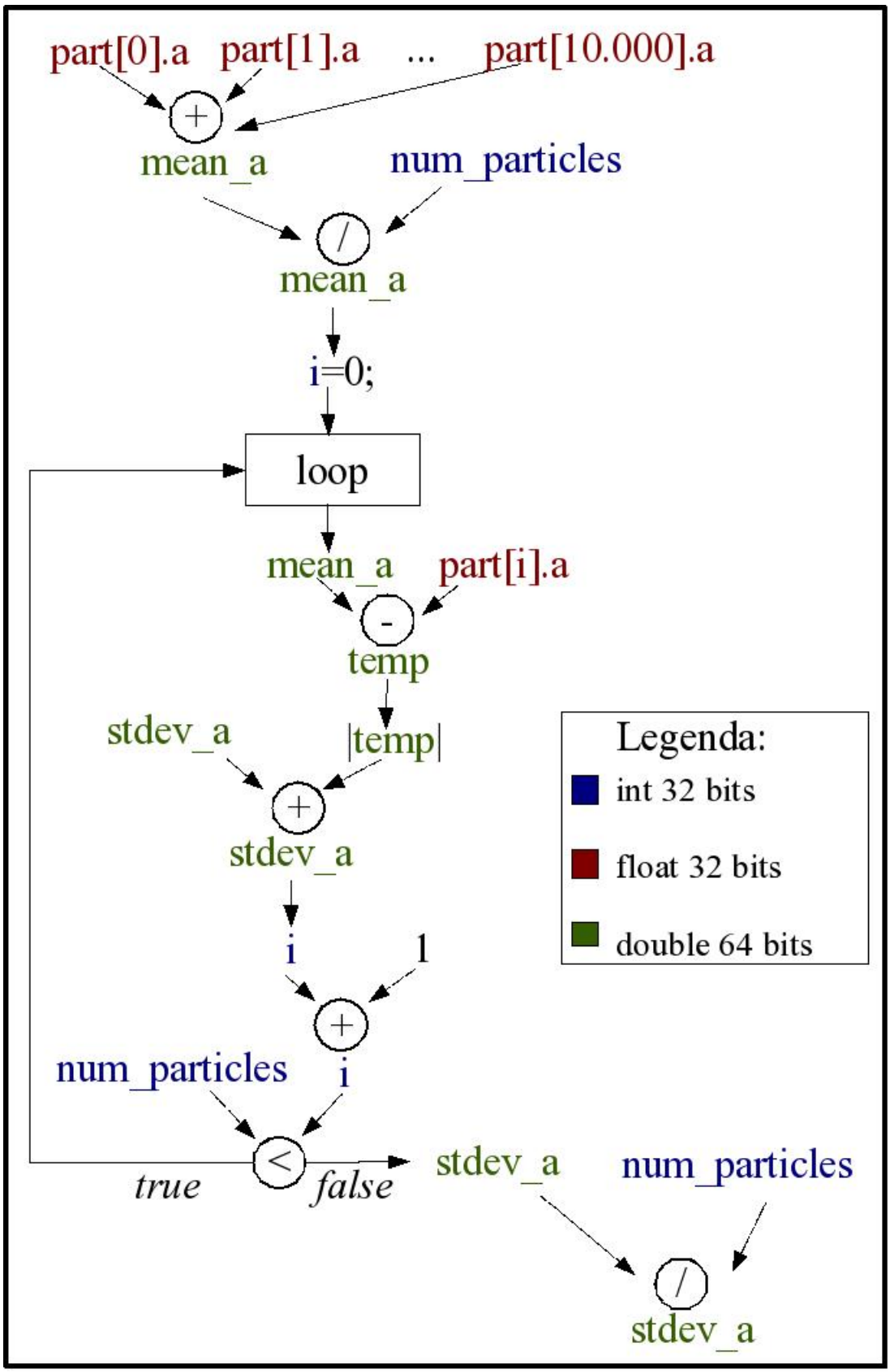

Figura 9.5: DFG da função "calc_mean()"em relação à coordenada a

```
void log_data(FILE *new_log, float odom[12], float scan[361][2], double time[2])
{

int i, j=0, ix, iy;

fprintf(new_log, "\n\n\%.6f",time[0]);
fprintf(new_log, "\n%.6f",time[1]);
fprintf(new_log,"\n%.3lf %.3lf %.3lf %.3lf %.3lf %.3lf ", glb_pos.mean_x,

for (i=0;i<num_particles;i++)
{
        ix=part[i].x*part[i].map_scale;
        iy=part[i].y*part[i].map_scale;
        if ((ix<0) ||(ix>=700) || (iy<0) ||(iy>=500) || (map[ix][iy]!=-1))
        j++;
}

if (j) g_print("\nNon Active particles: %i\n", j);

}
```

Figura 9.6: Função "log_data()".

```
------------ map3.map (organização) ------------
MAP_dim_x   MAP_dim_y   MAP_grid_size
p
p

...     (350000 valores de "p")

p
------------ map3.map (organização) ------------



------------ map3.map (exemplo)-----------------
500 700 0
-10.0
-10.0

...     (350000 valores de "p")

0.0
0.0
------------ map3.map (exemplo)-----------------
```

Figura 9.7: Arquivo map3.map

```
------------ laser_ang.txt (organização) ------------
scan[0][1]
scan[1][1]
scan[2][1]

... (180 valores de ângulo de laser)

scan[179][1]
------------ laser_ang.txt (organização) ------------



------------ laser_ang.txt (exemplo) ----------------
-1.571
-1.562
-1.553

... (180 valores de ângulo de laser)

1.571
------------ laser_ang.txt (exemplo) ----------------
```

Figura 9.8: Arquivo laser_ang.txt

```
------------ log.txt (organização) ----------------------------------
time[0]
odom[0]     odom[1]     odom[2]     ...   odom[11]
time[1]
scan[0][0]  scan[1][0]  scan[2][0]  ...   scan[180][0]
------------ log.txt (organização) ----------------------------------



------------ log.txt (exemplo) ---------------------------------------
1078256046.759
277.908          172.687          0.000 0.029       ...   -0.052
1078256046.736
81.910           81.910           81.910            ...   0.190
------------ log.txt (exemplo) ---------------------------------------
```

Figura 9.9: Arquivo log.txt

```
------------ new_log.txt (organização) ------------------------------
time[0]
time[1]
glb_pos.mean_x    glb_pos.mean_y    glb_pos.mean_a    glb_pos.stdev_x
      glb_pos.stdev_y   glb_pos.stdev_a
------------ new_log.txt (organização) ------------------------------



------------ new_log.txt (exemplo) ----------------------------------
1078256046.599000
1078256046.577000
284.625   211.450   -0.058   147.580   96.754   1.551
------------ new_log.txt (exemplo) ----------------------------------
```

Figura 9.10: Arquivo new_log.txt

```
float normal(float sd)
{

int i;
float num;

num=0;
for(i=0;i<12;i++)
        {
        num+=(rand()/(RAND_MAX/2.0))-1;
        }

num=num*sd/6;

return num;
}
```

Figura 9.11: Função "normal()".

```
------------------------------------ 64 bits ----------------------------
1078256046.439000
1078256046.418000
283.952   211.553   3.083   146.995   96.769   1.609

1078256046.279000
1078256046.280000
283.543   211.454   3.136   146.953   96.670   1.591

...

1078256039.800000
1078256039.771000
270.331   216.583   3.176   146.698   95.140   1.593

1078256039.640000
1078256039.613000
270.120   216.645   3.178   146.781   95.124   1.592

1078256039.480000
1078256039.454000
269.766   216.899   3.180   146.830   95.148   1.592

------------------------------------ 64 bits ----------------------------
```

Figura 9.12: Arquivo new_log.txt gerado a partir de variáveis com 64 bits

```
------------------------------------ 32 bits ------------------------------
1078256000.000000
1078256000.000000
285.212   210.819   3.088   146.574   96.754   1.608

1078256000.000000
1078256000.000000
285.056   210.786   3.144   146.560   96.611   1.586

...

1078256000.000000
1078256000.000000
272.405   216.911   3.203   146.052   94.701   1.588

1078256000.000000
1078256000.000000
272.304   216.908   3.207   145.992   94.644   1.587

1078256000.000000
1078256000.000000
272.141   217.125   3.211   145.957   94.560   1.587

------------------------------------ 32 bits ------------------------------
```

Figura 9.13: Arquivo new_log.txt gerado a partir de variáveis com 32 bits

% tempo (t)	t cumulat.	t próp.	chamadas	próp. cham.	tot. cham.	nome
42.09	801.16	801.16	100000	0.00	0.00	calc_mean
31.11	1393.34	592.18	100000	0.00	0.00	log_data
16.61	1709.55	316.21	733400000	0.00	0.00	normal
4.65	1798.16	88.60	300139284	0.00	0.00	sample
4.04	1875.07	76.91	1	0.08	1.90	run_filter
0.74	1889.15	14.08	1466	0.00	0.00	print_particles
0.45	1897.71	8.56	73	0.00	0.00	resample
0.16	1900.74	3.03	7300000	0.00	0.00	do_observation_model
0.07	1902.08	1.34	1	0.00	0.00	init_particles
0.05	1902.97	0.88	900001	0.00	0.00	update_data_file
0.02	1903.29	0.32	0	0.00	0.00	print_lh_map
0.02	1903.60	0.32	6259984	0.00	0.00	draw_point
0.01	1903.85	0.25	1	0.00	0.00	calc_lh_map
0.00	1903.87	0.02	1	0.00	0.00	load_map
0.00	1903.89	0.02	1	0.00	0.00	print_map
0.00	1903.91	0.02	1	0.00	0.00	update_laser_ang
0.00	1903.92	0.01	1467	0.00	0.00	refresh_screen
0.00	1903.93	0.01	1	0.00	0.00	create_window1
0.00	1903.93	0.00	3	0.00	0.00	lookup_widget
0.00	1903.93	0.00	1	0.00	0.00	add_pixmap_directory
0.00	1903.93	0.00	1	0.00	0.00	clear_map

Tabela 9.1: Perfil plano gerado pelo gprof

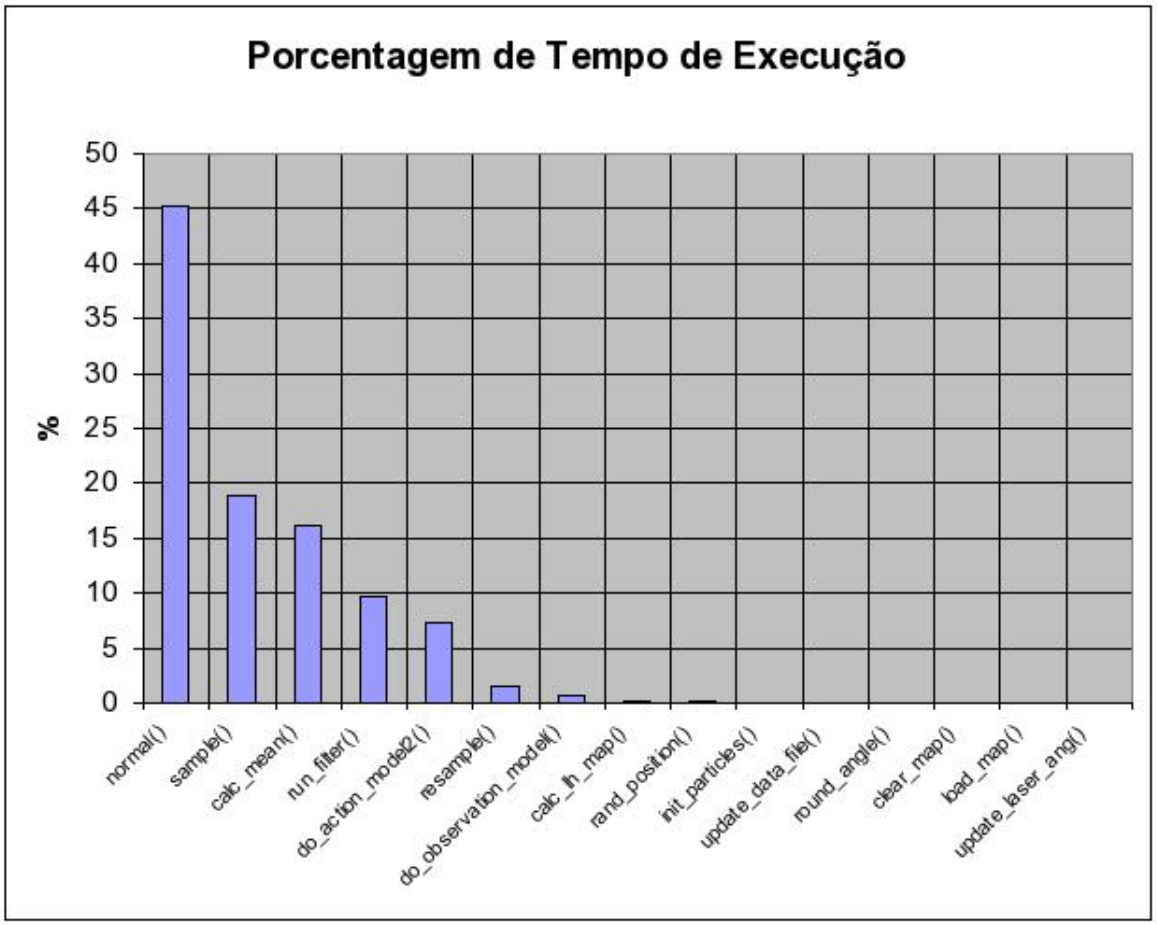

Figura 9.14: Porcentagem de execução de cada função do programa

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity FPMU is

	-- Generic declarations of the FPMU
	generic(
		LEN_SIGNAL : INTEGER := 1;
		LENGHT_EXP : INTEGER := 8;
		LENGHT_MAN : INTEGER := 25;
		LENGHT_MAN_PLUS : INTEGER := 26;
		LENGHT_FP : INTEGER := 34
	);

	port(
		signal clk: in std_logic;                          -- CPU system clock (always required)
		signal reset: in std_logic;                        -- CPU master asynchronous active high reset
		signal clk_en: in std_logic;                       -- Clock-qualifier (always required)
		signal start: in std_logic;
		signal done: out std_logic;
		signal n: in std_logic_vector(7 downto 0);         -- N-field selector of C I operations
		signal dataa: in std_logic_vector(31 downto 0);    -- Operand A (always required)
		signal datab: in std_logic_vector(31 downto 0);    -- Operand B (optional)
		signal result: out std_logic_vector(31 downto 0)   -- result (always required)
	);
```

Figura 9.15: Módulo da FPMU que define o componente a ser utilizado pelo processador Nios II

```
process(clk)
begin

        if clk'event and clk = '1' then

                if start = '1' then
                        b_instruct <= n;
                        b_fp_a     <= dataa & "00";
                        b_fp_b     <= datab & "00";
                        done   <= '0';

                else
                        done   <= '1';
                        result <= b_fp_result;

                end if;

        end if;

end process;
```

Figura 9.16: Funcionamento da implementação da FPMU em VHDL

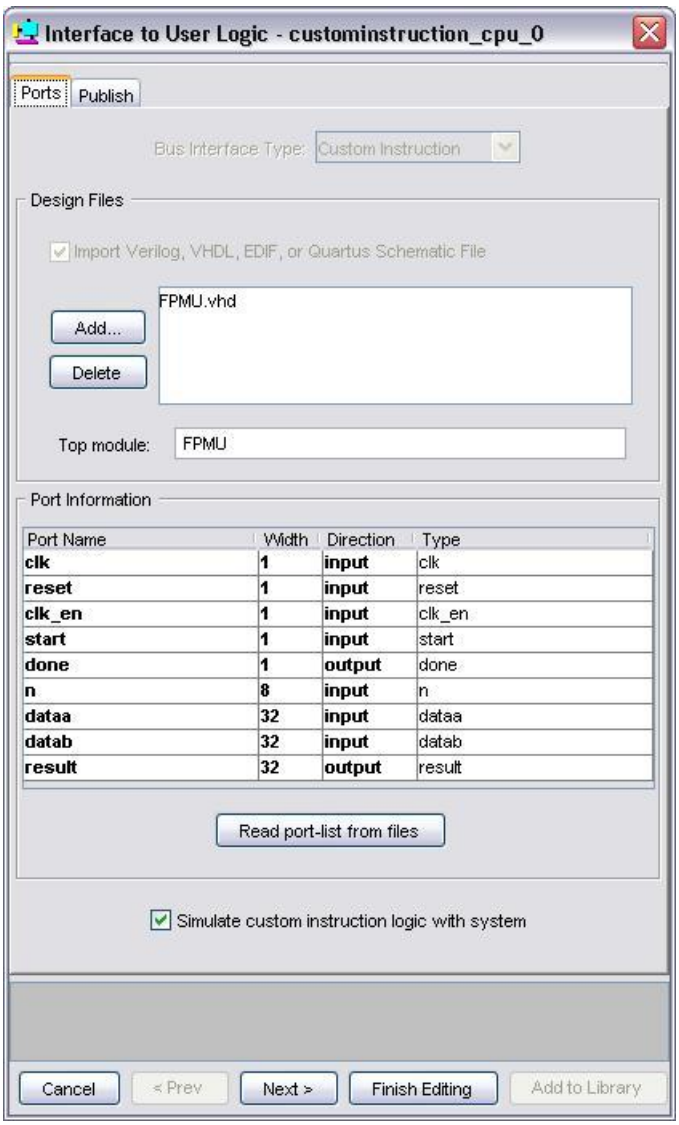

Figura 9.17: FPMU sendo integrada ao Nios II no SOPC Builder

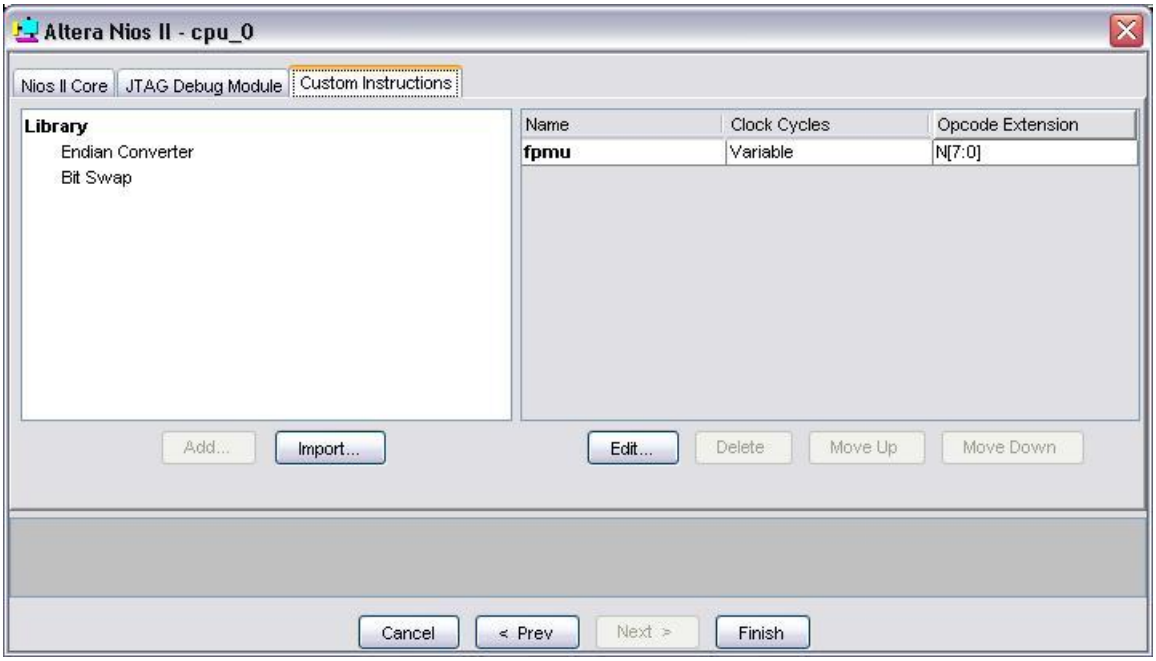

Figura 9.18: FPMU corretamente integrada ao Nios II no SOPC Builder

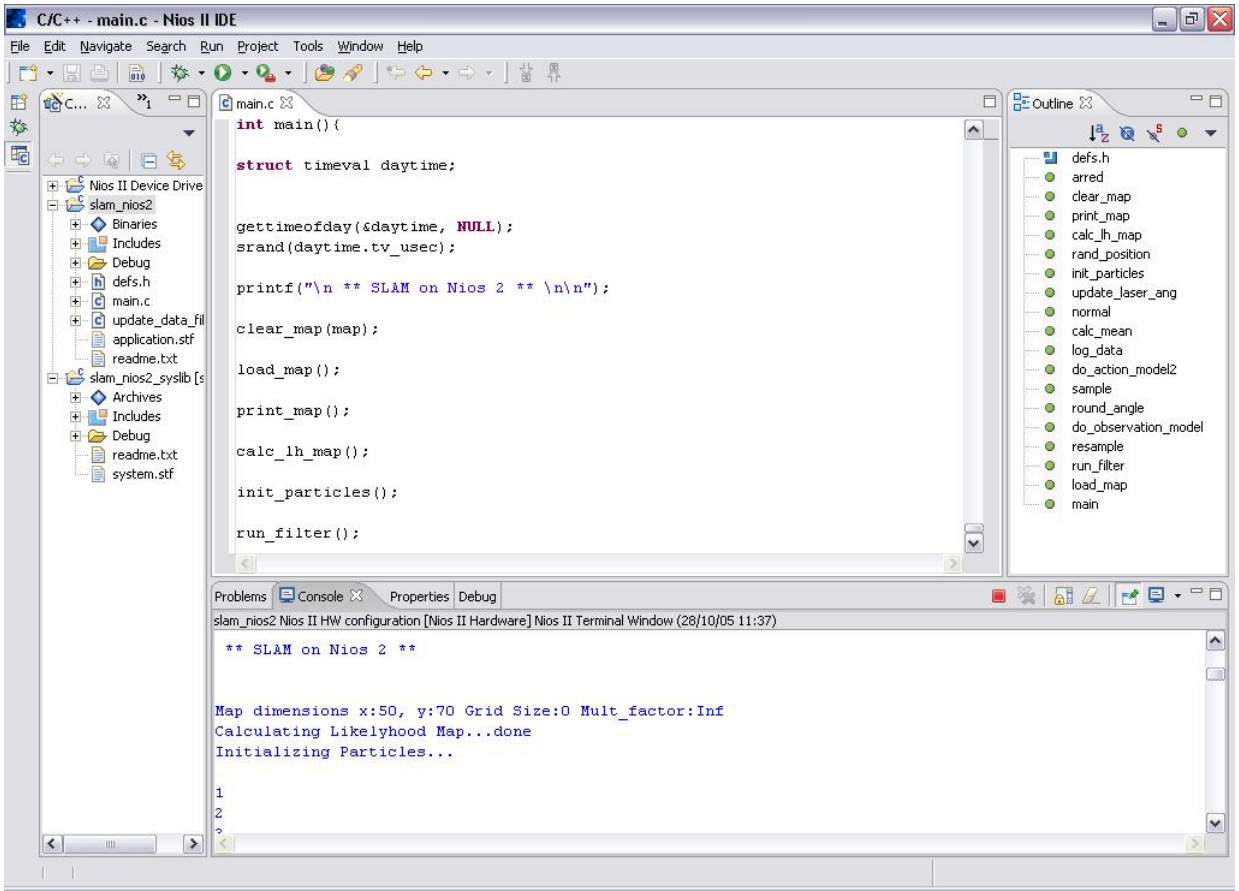

Figura 9.19: Solução embarcada em execução no Nios II