UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Proposta de uma Arquitetura Probabilística para Robótica Móvel e Aplicações Embarcadas

Bruno Franciscon Mazzotti

São Carlos / SP
Junho de 2007

Proposta de uma Arquitetura Probabilística para Robótica Móvel e Aplicações Embarcadas

Bruno Franciscon Mazzotti

Orientador: Prof. Dr. Eduardo Marques

Monografia preliminar de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação (ICMC/USP) para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração: Arquitetura de Computadores, Sistemas Embarcados, Robótica Móvel.

São Carlos / SP
Junho de 2007

Sumário

Lista de Figuras

Glossário

Lógica Combinacional – Na teoria de circuitos digitais, um circuito implementado com lógica combinacional tem sua saída definida como uma função das entradas atuais. Em contraste à lógica combinacional existe a lógica seqüencial na qual a saída do circuito não depende somente das entradas atuais mas também de todo o histórico das entradas anteriores. Em outras palavras, na lógica seqüencial há necessidade de memória de estado e na lógica combinacional não. [6]

Processador Softcore – Processador projetado com linguagens de descrição de hardware e implementado inteiramente através dos blocos lógicos de um dispositivo de hardware reconfigurável.

Unidade Lógica Aritmética (ULA) – A Unidade Lógica Aritmética é um circuito digital capaz de computar operações aritméticas (adição, subtração, multiplicação, etc.) e lógicas (e, ou, negação, etc.). A *ULA* é o bloco fundamental de qualquer tipo de processador. [6]

Resumo

O surgimento de um novo paradigma na área da robótica, a robótica probabilística, promoveu recentemente um grande avanço neste campo do conhecimento e trouxe à tona novas perspectivas para a construção de toda uma nova classe de robôs: os robôs móveis. Neste sentido, o seu grande mérito é considerar a incerteza inerente a esta classe de robôs. Os desafios enfrentados por robôs móveis encontram-se muito além das dificuldades já superadas pelos primeiros autômatos industriais e a busca por soluções destes problemas concentra atualmente os esforços de muitos cientistas. O próximo passo lógico na cadeia de evolução da robótica é o controle embarcado: o constante fluxo de novas tecnologias dá aos pesquisadores de sistemas embarcados diversas oportunidades para participar do desenvolvimento de sistemas robóticos. Este trabalho também se enquadra na conjuntura previamente descrita e propõe um projeto de arquitetura de computação probabilística a ser implementada em robôs movéis com o intuito de viabilizar o controle completamente embarcado.

1. Introdução

Nas últimas décadas tem-se observado um rápido crescimento no potencial de aplicação de sistemas robóticos. Em um primeiro momento, houve o desenvolvimento da robótica industrial. A utilização de robôs no chão de fábrica aumentou vertiginosamente a eficiência dos processos produtivos além de reduzir seus custos. [21] Atualmente os pesquisadores em robótica vêm dirigindo seus esforços para a construção de robôs móveis e introduzindo novas capacidades nos robôs a fim de habilitá-los a agir de forma autônoma. O robô móvel, o próximo passo na evolução da robótica, deve ser capaz de perceber o estado de seu ambiente e agir de forma apropriada combinando informações de diferentes fontes. Os problemas enfrentados no desenvolvimento de robôs móveis geralmente estão relacionados com a necessidade de interação adequada com os objetos físicos e entidades de um ambiente desestruturado, desconhecido ou dinâmico. A robótica móvel é um vasto campo a ser explorado e possibilita que uma ampla gama de novas aplicações seja desenvolvida. [29]

A construção de robôs autônomos tem sido um dos objetivos centrais de diversas áreas da ciência da computação. Diferentes metodologias foram elaboradas para o desenvolvimento de software de controle robótico, indo desde paradigmas baseados em modelos extremamente formais (nos meados da década de 70 do século XX) até tratamentos puramente reativos fortemente baseados em sensoriamento (nos meados da década de 80). A partir dos anos 90 surgiram modelos híbridos que mesclavam modelagem formal em níveis mais altos com um certo grau de reatividade em níveis mais baixos. [21] Uma das abordagens desenvolvidas no campo do controle robótico, a robótica probabilística, promoveu recentemente um grande avanço na área.

A robótica probabilística, cuja origem pode ser traçada até o início dos anos 60, proporcionou aos robôs móveis graus de autonomia e robustez nunca antes observados e soluções implementadas segundo seus preceitos demonstraram grande escalabilidade em aplicações complexas. Recentemente a abordagem probabilística tem-se tornado dominante na resolução de vários problemas da robótica. É fundamental também destacar que as melhores soluções conhecidas, até

o presente momento, para alguns problemas considerados difíceis no campo da robótica foram encontradas graças a métodos probabilísticos (o problema do robô seqüestrado, por exemplo). O recente sucesso da robótica probabilística pode ser atribuído a pelo menos dois fatores: o desenvolvimento teórico/algorítmico da área e a crescente disponibilidade de recursos computacionais até mesmo nos computadores pessoais mais modestos. [30][32]

No entanto, há críticas à abordagem probabilística. Tradicionalmente, as limitações mais freqüentemente citadas a respeito de algoritmos probabilísticos são: [29]

- Algoritmos probabilísticos geralmente são ineficientes se comparados a algoritmos tradicionais. A constatação de certa ineficiência nos métodos probabilísticos origina-se no fato de que a demanda por recursos computacionais para executar suas implementações é muito mais intensa. A computação deve ser realizada sobre distribuições de probabilidade inteiras e não sobre variáveis determinísticas convencionais.

- Algoritmos probabilísticos têm necessidade de realizar aproximações e não trabalham com valores exatos. Na prática, as aproximações são realizadas discretizando o ambiente, contínuo por natureza, ou através de análises por amostragem.

As limitações citadas, até certo ponto, não representam sérios obstáculos para os modernos computadores pessoais baseados em arquiteturas de propósito geral. Contudo, computadores de propósito específico, dotados de recursos computacionais mais modestos, podem ser incapazes de atender os requisitos de performance de certas aplicações implementadas através de métodos probabilísticos. Sistemas projetados para possuir características de uso geral podem executar um grande número de funções e assim atender a um maior número de usuários que fazem uso de computadores pessoais para suas atividades. Porém, o uso de arquiteturas de propósito geral não é ideal para todo e qualquer tipo de aplicação a ser executada por um computador de propósito específico. [4]

Sistemas dedicados executam uma ou poucas funções pré-definidas em tempo de projeto para atender requisitos próprios. Para a maioria das aplicações,

incluindo as da robótica probabilística, mudanças e personalizações na arquitetura do sistema que melhorariam significativamente o desempenho podem ser propostas. As mudanças na arquitetura diferem enormemente de acordo com a aplicação a ser considerada, mas essa técnica é viável ao se levar em conta os sistemas de propósito específico.

Tradicionalmente, as personalizações na arquitetura de sistemas computacionais dedicados é realizada através da utilização de circuitos digitais integrados especialmente projetados para tal finalidade, os *ASICs* (*Application Specific Integrated Circuits*). Como qualquer circuito integrado, um *ASIC* tem sua lógica interna invariável fundida na superfície de um substrato de material semicondutor. Essa tecnologia supera o processador de propósito geral em performance, porém restringe a flexibilidade da arquitetura e exclui totalmente qualquer tipo de otimização ou atualização em tempo de pós-projeto.

Um novo e emergente paradigma, a computação reconfigurável, mune os desenvolvedores com uma opção intermediária entre a alta performance dos *ASICs* e a adaptabilidade do fluxo de instruções dos processadores de propósito geral. A tecnologia de computação reconfigurável faz uso de hardware que pode ser adaptado em tempo de execução com o objetivo de promover flexibilidade sem comprometer seriamente o desempenho. Basicamente, a computação reconfigurável consegue combinar a adaptabilidade do software com a velocidade do hardware. [10]

Os sistemas baseados em computação reconfigurável são plataformas nas quais a arquitetura pode ser modificada em tempo real para melhor se adequar à aplicação que será executada, deste modo, o processador reconfigurável passa a trabalhar com uma arquitetura desenvolvida exclusivamente para a aplicação, permitindo uma eficiência muito maior do que a normalmente encontrada em processadores de uso geral. Isto ocorre pois o hardware é otimizado para executar os algoritmos necessários para a aplicação específica; ou seja, para se obter um melhor desempenho de um algoritmo, este deve ser executado num hardware específico para ele. Significativos ganhos de desempenho nas aplicações podem ser obtidos com a utilização desta tecnologia.

Atualmente, o dispositivo preferencial para o desenvolvimento de arquiteturas reconfiguráveis é o *FPGA* (*Field Programable Gate Array*), uma evolução natural dos primeiros dispositivos lógicos programáveis como o *PAL* (*Programmable Array Logic*) e o *GAL* (*Generic Array Logic*). Desde sua introdução, os *FPGA's* têm recebido grande destaque dentro da área de computação reconfigurável não somente pela sua capacidade de implementar blocos lógicos arbitrários, mas também por poderem ser reprogramados ilimitadamente. Basicamente, um *FPGA* pode ser visto como uma matriz de elementos lógicos e uma rede de interconexão configurável via software.

Hoje em dia, a tecnologia do *FPGA* já é dominada por diversos fabricantes (Xilinx, Altera, Lattice Semiconductor, Actel, Atmel, QuickLogic, Achronix Semiconductor) e seu uso já é difundido em diversas áreas (aeroespacial, militar, visão computacional, prototipação de *ASIC's,* etc.). Graças aos avanços da microeletrônica, os *FPGAs* estão se tornando cada vez mais densos em termos de elementos lógicos por área do *chip* - os modelos mais modernos de *FPGA* contêm cerca de 300.000 elementos lógicos. Isto torna o conceito de *SoPC* (*System on Programmable Chip*) cada vez mais viável na prática dos projetistas de sistemas.

Motivado pelos desafios proporcionados pelas tecnologias brevemente descritas, o presente trabalho defende a utilização de computação reconfigurável, especificamente *FPGA's*, para a construção de robôs móveis com controle totalmente embarcado implementado com abordagens probabilísticas. Os gargalos de performance dos algoritmos probabilísticos e as limitações do poder computacional dos sistemas embarcados em geral seriam, em tese, superados graças à aplicação de técnicas de co-projeto de hardware/software no desenvolvimento do sistema de controle robótico. Baseando-se neste argumento, a monografia traz a proposta de arquitetura de um *SoPC* projetado exclusivamente para realizar computação probabilística voltada para robótica móvel e analisa a viabilidade de sua implementação.

No Capítulo 2 faz-se uma revisão bibliográfica sobre robótica probabilística e sobre computação reconfigurável, mais especificamente sobre *FPGA's*. A primeira seção discorre sucintamente sobre certos fundamentos da robótica móvel e como alguns problemas da robótica podem ser solucionados probabilisticamente. A seção

seguinte versa sobre as técnicas, ferramentas e princípios relacionados à computação reconfigurável e sua realização com *FPGA's*. Concluindo o capítulo há uma análise crítica que levanta os requisitos necessários para a implementação em *FPGA* de uma arquitetura projetada especialmente para robótica probabilística embarcada e a verificação de sua viabilidade. O Capítulo 3 descreve a proposta de arquitetura para robótica móvel e suas funcionalidades. Finalmente, o Capítulo 4 expõe as conclusões a respeito do projeto desenvolvido e sugere os caminhos futuros que o trabalho poderá trilhar.

2. Revisão Bibliográfica

2.1. Síntese das Técnicas Envolvidas

Nesta seção são descritos, de forma resumida, os principais princípios, características, terminologias, técnicas e métodos relacionados ao domínio do trabalho. Primeiramente a robótica probabilística é apresentada, em seguida a computação reconfigurável e os *FPGA's*.

2.1.1. Robótica Probabilística

> *"C'est une vérité très certaine que, lorsqu'il n'est pas en notre pouvoir de discerner les plus vraies opinions, nous devons suivre les plus probables."*
>
> René Descartes

> *"I personally believe that probability is at the foundation of any intelligence in an uncertain world where you can't know everything."*
>
> Eric Horvitz

Em passado recente a robótica se encontrava restrita ao chão de fábrica das linhas de montagem desempenhando tarefas limitadas e repetitivas. Hoje em dia, graças a uma série de esforços de pesquisa bem sucedidos, a robótica está apta a penetrar em novos domínios mais complexos do que a automação industrial. Os novos domínios da robótica são inerentemente dinâmicos e incertos - lidar adequadamente com estes fatores é um dos principais desafios atualmente enfrentados pelos pesquisadores em robótica.

A capacidade de lidar com incerteza é um ponto chave para os robôs contemporâneos - um robô que carrega consigo a noção de sua própria incerteza e age de acordo com ela desempenhará melhor sua função. Esta conjectura levanta alguns questionamentos sobre quais seriam os mecanismos convenientes para tratar a incerteza. Como as leituras dos sensores devem ser integradas aos modelos internos dos robôs? Como os robôs devem tomar suas decisões uma vez que existe incerteza até mesmo a respeito das variáveis mais básicas envolvidas?

A robótica probabilística, cujas bases remontam solidamente à teoria das probabilidades, responde as questões levantadas com uma única idéia central: representar a informação utilizada na computação de forma probabilística. A teoria das probabilidades expressa elegante e explicitamente, ou seja, de maneira matemática e formal, a incerteza encontrada nas aplicações da robótica através de variáveis aleatórias e distribuições de probabilidade. [32]

Existem benefícios em se empregar técnicas probabilísticas à robótica se as contrapormos com abordagens mais clássicas. Robôs probabilísticos são mais robustos em face das limitações dos sensores e da dinâmica do ambiente e possuem requisitos menores em relação à precisão dos modelos aplicados. Além do que, se encarados de forma probabilística, muitos problemas da robótica são reduzidos a problemas de estimação de parâmetros, o que caracteriza a metodologia como geral o suficiente para ser largamente aplicada a toda uma classe de problemas envolvendo percepção e ação no mundo real. [30]

2.1.1.1. Modelagem de Sistemas Robóticos

A modelagem de sistemas robóticos possui o objetivo de arquitetar e expressar formalmente o mecanismo de percepção e controle dos robôs. Prover percepção a um robô consiste em muni-lo dos meios necessários para estimar seu próprio estado a partir da combinação das evidências de seus sensores. Odômetros e instrumentos de medição de distância (fundamentados nas tecnologias do sonar do *LASER*) são sensores tipicamente presentes em robôs móveis. Baseado em seu estado previamente inferido o robô está apto a desempenhar ações. A capacidade de decidir realizar ou não estas ações e incorporar seus resultados, tanto perante o ambiente e quanto ao próprio robô, caracteriza o controle robótico. Podem-se citar como ações características de um robô móvel a ativação de manipuladores e motores acoplados a rodas.

Tradicionalmente, a cinemática e a dinâmica de dispositivos robóticos são descritas de maneira determinística. Este tipo de modelagem caracteriza robôs idealizados livres de qualquer imprecisão e ruídos, sendo adequada somente aos casos mais simples. Ao se valer desta abordagem assume-se que o estado do robô

pode ser recuperado a qualquer momento, sem erro algum, a partir das medições de seus sensores e que uma determinada ação de controle sempre irá conduzir o robô a um determinado estado. [29]

Do ponto de vista matemático, o modelo clássico de sistemas robóticos pode ser descrito como uma quíntupla ordenada (Z, X, U, f, g), na qual:

- Z é um conjunto das possíveis medições captadas pelos sensores do robô.
- X é um conjunto dos possíveis estados válidos do robô.
- U é um conjunto das possíveis ações de controle a serem desempenhadas pelo robô.
- $f : X \times U \longrightarrow X$ é uma função determinística de transição de estado. O efeito de uma ação de controle sobre o estado do robô é governado pela relação $x_k = f(x_{k-1}, u)$ (sendo $x_{k-1}, x_k \in X$ e $u \in U$).
- $g : Z \longrightarrow X$ é uma função determinística de recuperação de estado. O estado do robô é recuperado mediante as medições de seus sensores, ou seja, $x_k = g(z_k)$ (sendo $x_k \in X$ e $z_k \in Z$).

Contrapondo-se à abordagem tradicional, o controle robótico probabilístico tem como fundamento a antecipação de diversas contingências que podem surgir em um ambiente real e incerto por natureza, mesclando assim a captura de informações (*exploration*) com a tomada de decisões orientada à performance (*exploitation*). O modelo probabilístico também pode ser visto como uma generalização do modelo clássico. [29]

Os sensores dos robôs são modelados de acordo com suas limitações (por exemplo, câmeras não podem ver através das paredes) e a presença de ruído. As ações desempenhadas pelos robôs nunca são encaradas como produtoras de resultados previamente esperados; pelo contrário, ações aumentam a incerteza do ambiente (ver figura 1). Um ambiente modelado de acordo com a abordagem probabilística é geralmente representado através de distribuições de probabilidade que descrevem a dependência de certas variáveis em relação a outras em termos

probabilísticos. O estado do robô também é representado utilizando-se distribuições de probabilidade que são deduzidas integrando-se as informações dos sensores ao modelo probabilístico do ambiente.

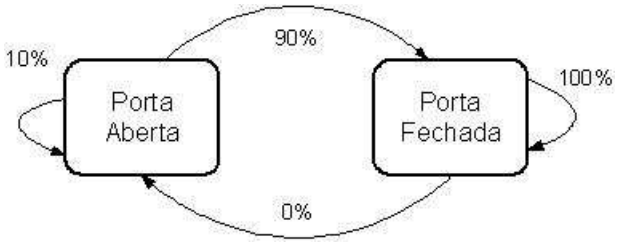


Figura 1: Modelagem probabilística da ação de controle "fechar a porta".

Matematicamente, o modelo probabilístico para sistemas robóticos é uma quíntupla ordenada (Z, X, U, f, g), na qual:

- Z, X e U têm a mesma semântica do modelo clássico.

- $f : X \times X \times U \longrightarrow [0,1]$ é uma função de distribuição de probabilidade que descreve a transição de estado do robô causada por uma ação de controle. Dado o estado anterior x_{k-1}, a aplicação de uma ação u leva a um novo estado x_k com certa probabilidade expressa por $f(x_k, x_{k-1}, u) = p(x_k \mid x_{k-1}, u)$ (sendo $x_{k-1}, x_k \in X$ e $u \in U$). Dados u e x_{k-1} quaisquer tem-se que $\int f(x_k, x_{k-1}, u) dx_k = 1$. A título de exemplo, o gráfico de uma certa f (fixado u) pode ser observado na figura 2.

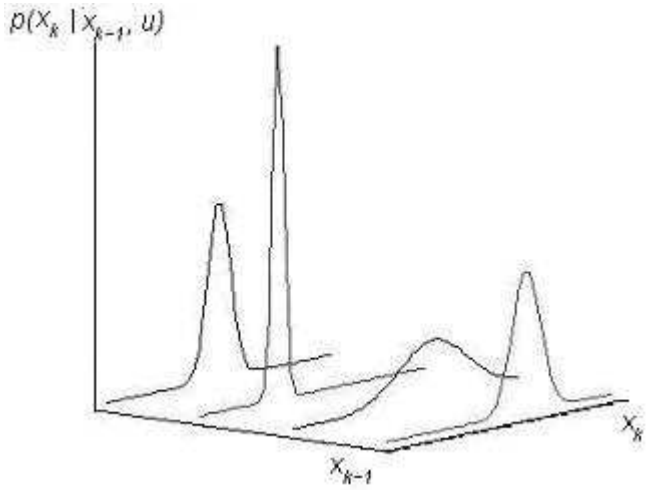


Figura 2: Probabilidade de transição de estados para uma dada ação de controle.

- $g : X \times Z \longrightarrow [0,1]$ é uma função de distribuição de probabilidade para recuperação de estado. Dadas certas medições dos sensores z_k, a probabilidade do estado do robô ser x_k é expressa por $g(x_k, z_k) = p(x_k \mid z_k)$ (sendo $x_k \in X$ e $z_k \in Z$). O sensoriamento também pode ser modelado de maneira inversa, ou seja, $g(x_k, z_k) = p(z_k \mid x_k)$ dependendo dos detalhes do algoritmo que será utilizado. Geralmente, esta modelagem é utilizada quando o algoritmo vale-se de métodos de inferência bayesiana e aproveita as evidências observadas para atualizar a probabilidade de que uma certa hipótese seja verdadeira.

Os parâmetros que descrevem as distribuições de probabilidade f e g foram omitidos anteriormente com o intuito de contribuir para uma melhor compreensão do modelo matemático. Eles podem ser arbitrários e são especificados de acordo com o problema a ser solucionado. Normalmente, o sensoriamento (função g) é modelado combinando-se uma distribuição normal para lidar com ruído e uma distribuição exponencial para tratar obstáculos inesperados. A função de transição de estados f usualmente descreve o movimento do robô e tem distribuição normal ou triangular.

2.1.1.2. Estimação Probabilística de Estado

Técnicas de estimação probabilística de estados de sistemas dinâmicos já são amplamente difundidas e a literatura sobre o assunto é vasta. [32] Esses procedimentos são aplicados à robótica com o objetivo de recuperar o estado do robô, x, a partir dos dados dos oriundos de seus sensores z. Os parâmetros que constituem o estado do sistema robótico variam de acordo com o problema a ser a ser solucionado.

Estimar parâmetros que descrevem a posição do robô em relação a um sistema de coordenadas externo caracteriza um problema de localização (figura 3). Estimar parâmetros que descrevem a posição de entidades do ambiente, como paredes e portas, define um problema de mapeamento. Problemas de mapeamento são considerados difíceis e computacionalmente intensivos devido à alta dimensionalidade dos parâmetros a serem estimados; quanto mais entidades

presentes no ambiente maior a complexidade. O mapeamento pode ainda ser mais árduo se os objetos alvo mudarem suas posições em função do tempo. [29]

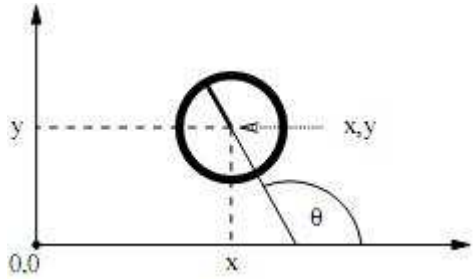


Figura 3: Estado utilizado em problemas de localização como vetor $[x, y, \theta]^T$.

O procedimento mais utilizado para estimação probabilística de estado é denominado filtro de Bayes. Com este método é possível estimar a distribuição de probabilidade sobre o espaço de estados baseando-se nas ações de controle e sensoriamentos anteriores. Diversos modelos estatísticos podem ser classificados como instâncias do filtro de Bayes (modelos de Markov escondidos e filtros de Kalman, por exemplo). Uma implementação popular do filtro de Bayes na área da robótica é o filtro de partículas. [11] Este algoritmo aproxima o estado do robô através de um conjunto de partículas. As partículas são amostras do espaço de estados distribuídas de acordo com uma função de probabilidade. O algoritmo computa o conjunto de partículas recursivamente baseando-se na ação de controle e nos sensoriamentos mais recentes.

Em uma primeira fase, a fase de predição, um modelo de movimento é utilizado com o intuito de prever o estado do robô; assume-se que o estado atual é dependente somente do estado anterior (asserção de Markov) e da ação de controle realizada. O modelo de movimento é especificado como uma densidade condicional $p(x_k \mid x_{k-1}, u)$ e a predição é obtida pela integração:

$$p(x_k \mid z_{k-1}) = \int p(x_k \mid x_{k-1}, u).p(x_{k-1} \mid z_{k-1})dx_{k-1}$$

Em uma segunda fase, a fase de atualização, um modelo de sensoriamento é utilizado para incorporar as informações dos sensores e obter a distribuição $p(x_k \mid z_k)$. Assumindo que z_k é condicionalmente independente de z_{k-1} dado x_k, o modelo de sensoriamento é expresso em termos da verossimilhança $p(z_k \mid x_k)$. A distribuição sobre o espaço de estados é obtida pelo teorema de Bayes:

$$p(x_k \mid z_k) = \frac{p(z_k \mid x_k).p(x_k \mid z_{k-1})}{p(z_k \mid z_{k-1})}$$

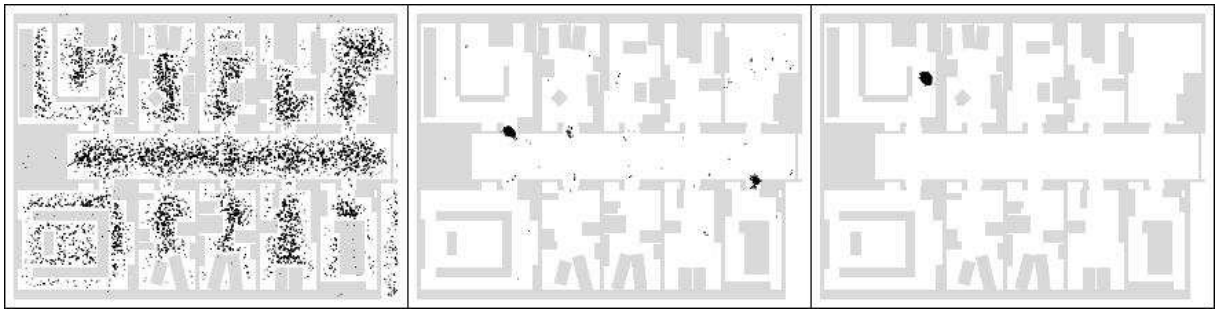
Após a fase de atualização, o processo é repetido recursivamente. Assume-se também que todas as condições iniciais são conhecidas.

Algoritmo Filtro de Partículas
  Parâmetros de entrada
    X, o conjunto de partículas computado na iteração anterior
    u, a ação realizada
    z, as medidas dos sensores
Início
   (i) Fase de Predição
   1. Faça $X' \leftarrow \{\}$
   2. **Para** i **pertencente a** $\{1 .. |X|\}$ **faça**
   3. Recupere a i-ésima partícula do conjunto X, x_i
   4. Sorteie uma amostra x'_i da distribuição $p(x_k|x_i,u)$ usando o modelo de movimento $p(x_k|x_{k-1},u)$
   5. Adicione x'_i ao conjunto X'

   (ii) Fase de Atualização
   6. Faça $X'' \leftarrow \{\}$
   7. **Para** i **pertencente a** $\{1 .. |X|\}$ **faça**
   8. Sorteie um elemento x''_i do conjunto X' com probabilidade proporcional à $p(z|x''_i)$ *usando o modelo de sensoriamento* $p(z_k|x_k)$
   9. Adicione x''_i ao conjunto X''

  10. **Retorne** o novo conjunto de partículas X''
Fim

A figura 4 ilustra a convergência do filtro de partículas em um problema de localização em duas dimensões. As partículas, que representam as possibilidades da verdadeira localização do robô, inicialmente encontram-se espalhadas pelo ambiente. Conforme o robô se move e incorpora a configuração do ambiente, as partículas tendem a se concentrar e revelar sua real posição. Isso só é possível graças às pequenas variações do ambiente; em um ambiente completamente simétrico a distribuição das partículas permanece multimodal e o algoritmo não converge completamente.



(a) Condição inicial *(b) Ambigüidade devido simetria* *(c) Localização consumada*

Figura 4: Convergência do filtro de partículas em um problema de localização.

2.1.1.3. Controle Probabilístico

Os métodos de controle robótico probabilístico, se comparados aos de estimação de estado, ainda estão pouco desenvolvidos. Uma justificativa plausível para este fato encontra-se na grande complexidade computacional própria dos processos de tomada de decisão. Entretanto, o controle probabilístico tem recebido considerável atenção recentemente. Os algoritmos existentes podem ser agrupados em duas classes principais: gulosos (*greedy*) e não-gulosos (*non-greedy*). [29]

As duas categorias de algoritmos associam uma função de custo às ações de controle. Os algoritmos gulosos maximizam o ganho local levando em conta somente o benefício que pode ser extraído da próxima ação a ser realizada. Os não gulosos atentam a um ganho mais global, considerando toda uma seqüência de controle e maximizando o seu ganho cumulativo. A utilização de métodos gulosos é interessante visto que considerar distintas possibilidades durante o planejamento de ações em longo prazo em um ambiente incerto ainda é computacionalmente complexo. Desenvolver eficientemente abordagens não-gulosas sobre múltiplos passos ainda é um problema desafiador.

Um algoritmo de planejamento de trajetória que merece destaque é o de navegação costeira (*coastal navigation*) [27]. O desenvolvimento deste algoritmo foi originalmente motivado pela observação de navios desprovidos de *GPS* que se deslocavam próximos à costa com o intuito de reduzir o risco de se perder em alto mar; o mesmo princípio pode ser aplicado aos robôs móveis.

2.1.2. Computação Reconfigurável

Tradicionalmente, a implementação de sistemas de computação sempre foi realizada através de hardware ou software executando em um processador (processador de propósito geral, processador de sinais ou microcontrolador). Ao implementar um sistema computacional, o desenvolvedor/projetista deveria optar por uma das alternativas para todo o projeto ou, em alguns casos, deliberar quais subsistemas deveriam ser implementados em hardware ou em software ponderando a respeito das vantagens e desvantagens relativas de cada abordagem. [5]

Soluções em hardware oferecem alto desempenho, pois possuem execução espacial e paralelismo intrínseco e sua lógica interna é especificamente projetada para a aplicação. Soluções em software podem ser descritas como engenhos de execução capazes de interpretar um fluxo de instruções flexível porém relativamente ineficientes em relação ao hardware. Isto se dá devido à característica temporal e seqüencial da execução do fluxo de instruções e ao fato dos operadores implementados não serem específicos para a aplicação. A figura 5 ilustra a diferença entre computação espacial e temporal. Em implementações espaciais cada operador utilizado existe em um ponto distinto do espaço possibilitando paralelismo real. Em implementações temporais um menor número de recursos computacionais mais genéricos são reutilizados com o passar do tempo. [10]

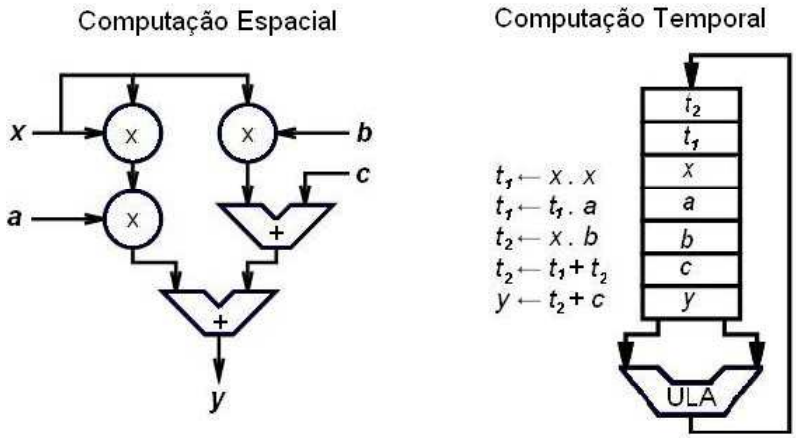


Figura 5: Computação espacial e temporal para a expressão $y = ax^2 + bx + c$.

Contudo, o desenvolvimento de tecnologias de computação reconfigurável introduziu uma nova possibilidade que, até certo nível, mescla as propriedades das alternativas convencionais. A computação reconfigurável destaca-se cada vez mais

como uma nova e importante estrutura organizacional para computação, combinando a programabilidade dos processadores convencionais com a capacidade de computação paralela/espacial do hardware de propósito específico. A emergência deste novo paradigma de computação tornou as fronteiras tradicionais entre hardware e software difusas e levou ao surgimento de toda uma nova classe de arquiteturas de computadores: as arquiteturas reconfiguráveis.

As características chave que diferenciam máquinas com arquitetura reconfigurável de máquinas convencionais são o seu alto grau de utilização de computações personalizadas, espaciais e dotadas de controle e recursos distribuídos. Arquiteturas reconfiguráveis têm se mostrado uma alternativa válida graças ao seu bem sucedido uso em diversas aplicações tais como criptografia, reconhecimento de voz, reconhecimento automático de alvos, comparação de padrões, compressão de dados, entre outras. [5]

2.1.2.1. FPGA's

Os recentes avanços da computação reconfigurável são, em grande parte, frutos do desenvolvimento dos *FPGA's*. Suas raízes históricas se encontram nos *CPLD's* (*Complex Programmable Logic Devices*) dos meados da década de 80*.* Em um primeiro momento os *FPGA's* foram utilizados como mecanismos de prototipação rápida ou como lógica de interconexão de circuitos mais complexos, porém o aumento da velocidade e capacidade destes dispositivos fez com que eles se tornassem a base da moderna computação reconfigurável.

Um *FPGA* é um dispositivo semicondutor que contém internamente um arranjo de elementos cujas funcionalidades podem ser determinadas através de múltiplos bits de configuração. Estes elementos, blocos lógicos no jargão da área, são interconectados por um conjunto de rotas também programável. Desta forma, circuitos arbitrários podem ser implementados em um *FPGA* mapeando suas funções de computação aos blocos lógicos, utilizando as rotas para conectá-los e então obter o comportamento global esperado. [8]

A reconfigurabilidade de um *FPGA* é obtida com o uso de antifusíveis ou bits de memória *SRAM* (*Static Random Access Memory*) para controlar o estado de transistores. A tecnologia dos antifusíveis utiliza-se de correntes elétricas altas e é tipicamente pouco reprogramável. *FPGAs* cuja configurabilidade é baseada em *SRAM* podem ser reprogramados ainda ligados através do download de diferentes bits de configuração para suas células de memória.

Um bloco lógico típico de um *FPGA* (figura 6) contém uma ou mais *LUT* (*look-up table*), *flip-flops* opcionais, alguns elementos extras de lógica combinacional e uma célula de *SRAM* para controle de configuração. As *LUTs* funcionam como uma tabela verdade e permitem que qualquer função booleana seja implementada, proporcionando lógica genérica. Os *flip-flops* podem ser utilizados para *pipelining*, armazenamento de estados de autômatos finitos, como um registrador ou em qualquer outra situação que um *clock* faça-se necessário. Os elementos extras são um recurso especial que provê ao bloco um aumento de velocidade em outras operações como adição e paridade. [8]

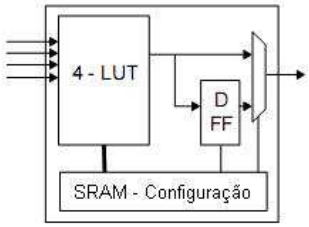


Figura 6: Arquitetura típica de um bloco lógico de FPGA.

A maioria das arquiteturas de *FPGA* organiza suas estruturas de roteamento como um mar de recursos de conexão de modo a permitir uma comunicação rápida e eficiente entre as linhas e colunas dos blocos lógicos ilhados. Como é mostrado na figura 7, os blocos lógicos se encontram imersos em uma estrutura de roteamento geral, com suas portas de entrada e saída anexadas ao meio de roteamento através de blocos de conexão. Os blocos de conexão são implementados com multiplexadores programáveis que selecionam qual sinal de qual canal de roteamento estará conectado aos terminais do bloco lógico. Os sinais gerados nos blocos lógicos seguem até os blocos de conexão e depois até os fios da rede de roteamento. Normalmente também existem comutadores inseridos entre os blocos de conexão. Os comutadores dispõem de conexões programáveis entre as trilhas

verticais e horizontais que permitem que os sinais mudem de direção caso seja necessário (figura 8). [8]

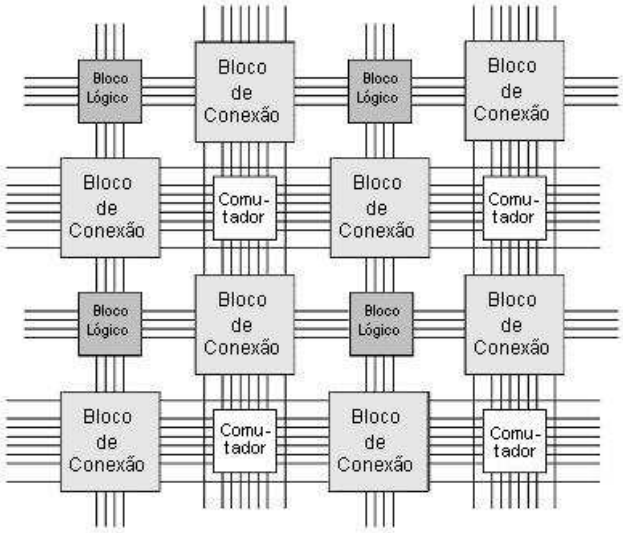


Figura 7: Arquitetura típica de um FPGA.

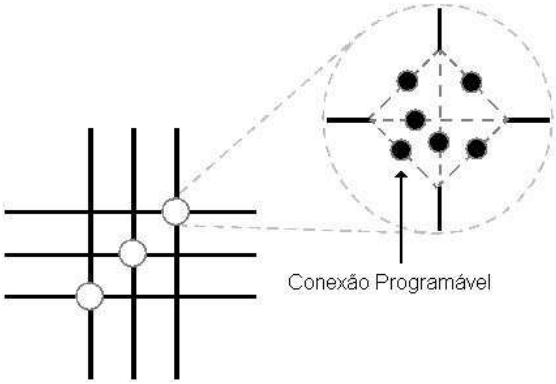


Figura 8: Detalhe de um bloco comutador.

Pode-se notar que a arquitetura de roteamento de um *FPGA* é uma estrutura bastante complexa e projetada para suportar padrões de conexão arbitrários - os blocos de conexão e comutadores ao redor de um único bloco lógico possuem milhares de pontos programáveis. Contudo, os usuários de *FPGA's* não necessitam se preocupar com detalhes arquiteturais uma vez que as ferramentas de projeto realizam o roteamento automaticamente.

2.2. Trabalhos Relacionados

Nesta seção são descritos alguns trabalhos relacionados ao projeto e que motivaram sua realização.

2.2.1. Convênio Bilateral de Cooperação Internacional Brasil - Protugal (CNPq - Grices)

Os laboratórios de Computação Reconfigurável (LCR) do ICMC/USP e do INESC-ID (Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento) de Lisboa estão desenvolvendo em conjunto um projeto em longo

prazo que visa a construção de um robô móvel totalmente baseado em computação reconfigurável. O intuito principal desta parceria é criar primeiramente uma biblioteca de algoritmos de robótica implementados em hardware e, em futuro próximo, um núcleo inteligente que controle a execução dos mesmos.

A implementação dos algoritmos está sendo efetuada em *VHDL*, uma poderosa linguagem de descrição de hardware de alto nível comumente usada em projetos de circuitos digitais. [26] O uso da tecnologia de computação reconfigurável tornará o sistema altamente flexível. O robô, utilizando um número fixo de circuitos integrados, estará apto a executar todos os algoritmos disponíveis na biblioteca reprogramando-se automaticamente em tempo de execução sem a necessidade de desligamento. O núcleo de controle, valendo-se de técnicas avançadas de inteligência artificial, será responsável por escolher a programação dos dispositivos reconfiguráveis adequada a cada situação enfrentada pelo robô. O LCR também está desenvolvendo uma ferramenta específica para auxiliar o desenvolvimento dos algoritmos robóticos em hardware, o ARCHITECT. [13] Em estado mais avançado este projeto viabilizará a construção de robôs móveis capacitados a realizar tarefas de missão crítica em tempo real graças à alta performance proporcionada pelas otimizações em hardware.

2.2.2. *FPGA's* Aplicados à Robótica

FPGA's, e computação reconfigurável no geral, são aplicáveis à grande maioria problemas que podem ser solucionados com o uso de técnicas digitais. LEONG e TSOI conduziram um estudo descrevendo alguns casos nos quais os *FPGAs* auxiliaram o desenvolvimento de aplicações em robótica [19]. No presente trabalho citam-se os qualificados como mais interessantes.

- Fórmula 1: Apesar de não ser exatamente um robô, um carro de Formula 1 possui muitas características similares, o que o torna um caso válido para estudo. O carro de 2003 da equipe BMW Williams utilizou uma *VCMU* (*Vehicle Control and Monitoring Unit*) fabricada com um *DSP* (*Digital Signal Processor*) da Texas Instruments e um *FPGA* Virtex XVC600-E da Xilinx. A *VCMU* controla diversos aspectos do veículo como mudanças de marcha, controle de tração e telemetria.

A utilização de um *FPGA* foi benéfica por dois motivos principais. Primeiramente, as funções críticas cujos requisitos não eram atendidos pelo *DSP* foram mapeadas à base reconfigurável com ganho de performance. Em segundo lugar, o *FPGA* reduziu o tempo necessário para incorporar mudanças e tornou a equipe mais competitiva.

- Controlador robótico f*uzzy*: Um *FPGA* modesto da Altera, o EPF6024ACT144-3 com apenas 24.000 portas lógicas, foi utilizado com sucesso na implementação de um controlador de comportamento autônomo baseado em lógica *fuzzy* para um robô móvel. Todas as etapas do controle (processos de *fuzzyficação*, inferência e *desfuzzyficação*) são realizadas por módulos de hardware no próprio *FPGA*.

- Sondas de exploração de Marte: *FPGA's* foram intensivamente usadas nas sondas de exploração de Marte *Mars Pathfinder*, *Mars Surveyor'98* e *Mars Surveyor'01*. As sondas eram dotadas de diversas câmeras e os *FPGAs* se encarregaram de realizar tarefas relacionadas com estes instrumentos incluindo: a decodificação de comandos do computador central das sondas, armazenamento das imagens capturadas e controle dos motores de passos para movimentar as câmeras. É importante destacar que os dispositivos reconfiguráveis mostraram-se aptos a funcionar corretamente em um ambiente inóspito como o espaço.

2.2.3. Robôs Reconfiguráveis

Robôs reconfiguráveis são aqueles que têm a capacidade de modificar sua estrutura para melhor se adequar à realização de uma determinada tarefa. A grande motivação para pesquisas a respeito de robôs reconfiguráveis é a necessidade de se criar robôs móveis que possam de adaptar às dificuldades de locomoção existentes em tipos de terreno distintos. Geralmente os robôs reconfiguráveis são compostos de módulos simples que interagem entre si a fim de habilitar o comportamento emergente do sistema robótico como um todo. Robôs modulares se comparados aos monolíticos apresentam algumas vantagens (maior robustez e

reparos facilitados), mas introduzem novas problemáticas relacionadas ao controle da configuração [13].

A seguir são descritos de forma breve alguns projetos de robôs reconfiguráveis modulares [13].

– Polypod (figura 9): Um robô reconfigurável bimodular desenvolvido na Universidade de Stanford sob patrocínio da XEROX. O Polypod é composto por apenas dois tipos de módulos: nós e seguimentos de conexão.

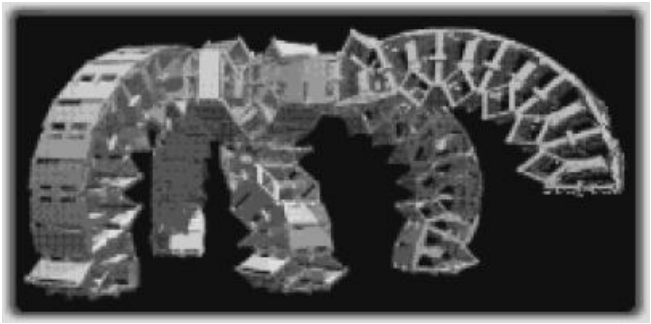


Figura 9: O robô modular Polypod.

– I-Cubes: São robôs bipartidos desenvolvidos na Universidade de Carnegie Mellon. A estrutura dos I-Cubes é formada por cubos, que podem conter sensores, e módulos de ligação que se conectam às faces dos cubos. Os elementos de ligação podem se conectar e desconectar dos cubos, mover cubos de lugar e locomover-se entre eles. Todos os módulos de ligação dos robôs são idênticos mas podem ser controlada de forma independente e distribuída.

2.2.4. *CES*: Uma Ferramenta de Programação para Robótica Probabilística

Os recentes sucessos da robótica probabilística motivaram o projeto de uma ferramenta de programação especificamente voltada para a área, a *CES* (acrônimo para *C++ for Embedded Systems*). A *CES* é uma extensão da linguagem de programação C++ que incorpora nativamente mecanismos de aprendizado baseado em exemplos e computação probabilística provendo ao programador tipos de dados, operadores e comandos para manipular distribuições de probabilidade; estes recursos possibilitam que o programador lide de forma natural com as imprecisões dos sensores robóticos e do ambiente. O objetivo desta ferramenta, é promover uma

maior agilidade no desenvolvimento de software de controle de robôs móveis. A utilização da *CES* pode dinamizar o processo de desenvolvimento uma vez que a criação de bibliotecas de rotinas probabilísticas pode ser tediosa, sujeita a erros e consumir tempo precioso dos programadores. [28][31]

2.3. Análise Crítica

> *"If you round off the fractions, embedded systems consume 100% of the worldwide production of microprocessors."*
>
> *Jim Turley*

A lei de Moore (ver gráfico na figura 10) atesta que a densidade de transistores dos circuitos integrados digitais dobra aproximadamente a cada dezoito meses. Ao estender as projeções da lei de Moore por um prazo longo, chega-se logicamente à assustadora conclusão de que a capacidade dos circuitos aumenta em mil vezes a cada quinze anos. Não coincidentemente, a computação passa por uma mudança de paradigma a cada quinze anos. Atualmente, vive-se uma mudança de paradigma: a computação move-se para um quadro caracterizado pela presença de múltiplos dispositivos computacionais por indivíduo, a era pós-PC. [12]

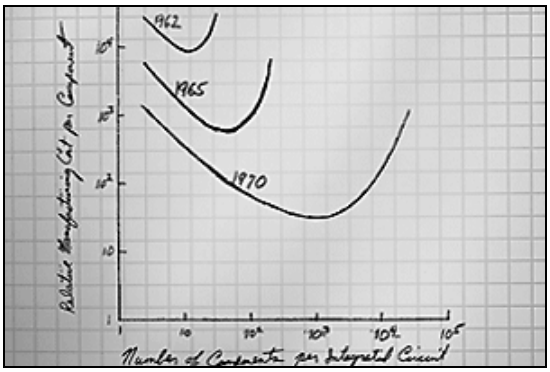


Figura 10: Gráfico original de Gordon Moore, 1965.

A nova onda tecnológica já alterou consideravelmente a relação custo/volume de produção. Hoje em dia, o custo de fabricação de um produto eletrônico (peças, componentes e montagem) não se compara ao custo de engenharia *(NRE*, *Non-Recurring Engeneering*). Os processos da indústria de semicondutores visam produtividade máxima valendo-se de avançadas ferramentas de software *EDA* (Eletronic Design Automation). A nova geração de computadores é a mais barata e

mais poderosa que já se viu - poder computacional tornou-se uma "*commodity* agrícola" do século XXI comercializada a centavos de dólar.

Um dos efeitos da revolução embarcada é a pervasividade dos nascentes frutos da robótica. Robôs começam a entrar na vida cotidiana das pessoas como brinquedos, eletrodomésticos e ferramentas de trabalho. [21] O robô móvel autônomo é uma forte necessidade do mercado e um lucrativo campo a ser explorado comercialmente.

Conforme apresentado nos capítulos anteriores, fica claro que o caminho mais favorável para o surgimento de robôs móveis realmente inteligentes passa pela robótica probabilística. A tecnologia do *FPGA* já é amplamente consolidada dentro da própria robótica e demonstrou na prática sua potencialidade para resolução de problemas computacionalmente intensivos. Contudo, mesmo com extensa pesquisa, não se encontrou na literatura especializada nenhum relato da combinação de *FPGAs* e robótica probabilística. Técnicas estatísticas já foram implementadas com sucesso em *FGPAs* e utilizadas em problemas de processamento de sinais e simulação computacional.

Tomando como fundamento tudo o que foi discutido até agora, pode-se assegurar que a implementação de uma arquitetura reconfigurável otimizada para controle robótico probabilístico é promissora em termos de resultados esperados. Ademais, os modernos *FPGA's* certamente atendem os requisitos necessários para concretizar a proposta deste trabalho.

3. Estado Atual do Trabalho

Tendo em vista que todo o processo de personalização de uma arquitetura de hardware para um dado algoritmo é extremamente específico e sensível às condições exatas do problema decidiu-se por, em um primeiro momento, trabalhar com um único algoritmo: o filtro de partículas. Esta decisão pode ser justificada com base nos seguintes argumentos:

- A subárea de estimação de estados foi escolhida, pois o controle probabilístico ainda não está plenamente consolidado.

- O filtro de partículas é genérico o suficiente para solucionar várias instâncias do problema de estimação de estados de sistemas dinâmicos, estando apto a resolver problemas de localização e mapeamento. Ademais, a aceleração via hardware poderá auxiliar o tratamento de espaços de alta dimensionalidade em problemas de mapeamento.

- Um trabalho mais extenso não se encaixaria no escopo de uma monografia de conclusão de um curso de graduação.

Esta monografia é somente o primeiro passo rumo a um controle embarcado pleno para robôs móveis. Trabalhos futuros certamente farão uso da experiência adquirida com os esforços correntes e tratarão de abordar outras possibilidades que contemplem mais benefícios ao campo da robótica probabilística.

3.1. Proposta

A monografia descreve uma arquitetura reconfigurável projetada especificamente para otimizar o algoritmo de filtro de partículas utilizado para estimação de estados em robótica móvel. Esta arquitetura baseia-se no flexível processador embarcado da Altera, o Nios II. O Nios II é um *RISC* (*Reduced Instruction Set Computing)* de 32 bits altamente personalizável e um dos processadores *softcore* mais utilizados no mundo todo para se implementar *SoPCs* em *FPGAs*. [2] Com a adição de novas instruções personalizadas ao conjunto pré-definido de instruções do Nios II pretende-se que o filtro de partículas tenha seu

desempenho melhorado, superando assim a relativa ineficiência atribuída às técnicas probabilísticas. [1] Futuramente, a arquitetura será de fato implementada com modernas linguagens de descrição de hardware e testada em um *FPGA* da Altera. Este processo será totalmente apoiado em ferramentas *EDA* de síntese, roteamento e simulação e provavelmente a família *Cyclone* de *FPGAs* será o alvo do projeto devido ao seu baixo consumo de energia e principalmente seu baixo custo.

O filtro de partículas é uma sofisticada técnica de estimação baseada em simulação probabilística, e como tal exige grandes quantidades de números aleatórios que atendam alguns critérios estatísticos de qualidade (alta equidistribuição, por exemplo). A representação das distribuições de probabilidade utilizadas por meio de um conjunto de amostras atualizado continuamente faz com que o sorteio de variáveis aleatórias seja extremamente intensivo. A geração de números aleatórios é um processo complexo e, para a aplicação em questão, certamente é o ponto crítico em termos de exigências computacionais. Dessa forma, mapear um gerador de números pseudo-aleatórios (*PRNG*) ao hardware da base reconfigurável solucionaria os problemas de desempenho do filtro de partículas em espaços de alta dimensionalidade.

A seção seguinte trata mais detalhadamente sobre *PRNGs* e mais especificamente sobre o gerador escolhido para o projeto, o MT19937 [24].

3.1.1. Geradores de Números Pseudo-aleatórios

> *"The generation of random numbers is too important to be left to chance."*
>
> *Robert Coveyou*

> *"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."*
>
> *John von Neumann*

A geração de números aleatórios é um problema fundamental da computação que vem sendo seriamente pesquisado há décadas. Um gerador de números pseudo-aleatórios (*PRNG*) é um algoritmo determinístico que se vale de métodos

aritméticos para gerar uma seqüência de números aproximadamente não correlacionados e que tem propriedades semelhantes a números aleatórios reais. A seqüência gerada por um *PRNG* não é realmente aleatória, visto que ela baseia-se em alguns valores iniciais chamados de sementes. [15]

Atualmente, existem diversos métodos para se gerar números aleatórios e vários testes estatísticos podem ser aplicados a uma seqüência pseudo-aleatória para determinar a qualidade de sua aparente aleatoriedade. A escolha do *PRNG* a ser utilizado por uma aplicação é um fator muito importante para seu sucesso. Os *LCGs* (*Linear Congruential Generators*), a classe mais antiga e mais conhecida entre os *PRNGs,* definidos por uma relação de recorrência do tipo $x_{k+1} = ax_k + b$ (com $k \geq 0$) tendem a demonstrar alguns defeitos: são muito sensíveis em relação a escolha de seus parâmetros e não são indicados para aplicações nas quais uma alta aleatoriedade da seqüência gerada é requerida. [15] Usar um *LCG* em um jogo eletrônico é aceitável, em um filtro de partículas não.

Após um levantamento intensivo acerca dos *PRNGs* existentes chegou-se à conclusão a melhor opção para o projeto era o gerador Mersenne Twister. Este gerador tem sua implementação baseada em profundos fundamentos teóricos da matemática discreta e da álgebra abstrata e funciona através de uma relação de recorrência linear matricial sobre o campo de Galois F_2. Seus criadores o projetaram para superar as deficiências dos seus antecessores e foram bem sucedidos na tentativa: o Mersenne Twister gera números aleatórios de alta qualidade muito eficientemente. [24]

Existem diversas variantes do Mersenne Twister capazes de gerar números aleatórios de tamanhos distintos em termos de número de bits; as mais utilizadas geram números de 32 e 64 bits. O MT19937, a variante mais popular do Mersenne Twister para números de 32 bits, foi escolhida como alvo de estudo da presente monografia. Esta escolha pode ser justificada pelo fato de que o tamanho da palavra do processador Nios II também é 32 bits, tornando assim a integração com *PRNG* mais facilitada.

O Mersenne Twister é uma generalização de uma classe de geradores conhecidos como *Lagged Fibonacci* (*LFG*), mais propriamente dos *GFSR*

(*Generalised Feedback Shift Register*). O *LFG* é uma extensão da seqüência de Fibonacci, $x_k = x_{k-1} + x_{k-2}$, com recorrência expressa por $x_{k+n} = x_{k+m} * x_k$ (com $n > m > 0$ e $k \geq 0$), na qual o símbolo $*$ denota uma operação binária. No caso dos *GFSR*, a operação binária é ou exclusivo bit a bit: $x_{k+n} = x_{k+m} \oplus x_k$. [23]

Os *GFSR* são rápidos e sua implementação independe do tamanho da palavra da máquina alvo w, porém existem algumas ressalvas quanto a sua utilização: [23]

- A seleção das sementes é muito crítica e influencia profundamente a aleatoriedade da seqüência, tornando uma boa inicialização é difícil e custosa.
- O período da seqüência, $2^n - 1$, é muito menor que o limite superior teórico, 2^{nw}.
- O algoritmo requer uma área de trabalho de n palavras. Muita memória pode ser consumida se vários geradores forem implementados simultaneamente
- A relação de recorrência utilizada baseia-se no trinômio $t^n + t^m + 1$, que deve ser primitivo sobre os inteiros módulo 2 - uma condição rara.

O Mersenne Twister por sua vez apresenta as seguintes propriedades que o qualificam para ser utilizado em simulações distribuídas como o filtro de partículas: [23][24]

- Inicialização das sementes livre de restrições severas.
- Período da seqüência é colossal, $2^{19937} - 1$ para o MT19937.
- Área de trabalho equivalente a $1/w$ de um *GFSR* de mesmo período.
- A seqüência gerada é estatisticamente distribuída da melhor maneira possível.
- O gerador passou por uma bateria de rigorosos testes estatísticos, incluindo os testes *diehard* propostos por MARSAGLIA. [20]

Qualquer variante do Mersenne Twister, respeitando a condição de que $2^{nw-r} - 1$ seja um primo de Mersenne, pode ser descrito pelas seguintes características: [24]

- w: Tamanho da palavra a ser gerada (em número de bits).

- n: Grau de recorrência do gerador.
- m: Número de seqüências aleatórias que podem ser geradas paralelamente por um único gerador ($1 \le m \le n$).
- r: Ponto de separação de uma palavra. Divide a palavra gerada em duas partes, inferior e superior. Também pode ser interpretado como o número de bits da parte inferior ($0 \le r \le w-1$).
- a: Coeficientes da matriz de *twisting* na forma racional normal. Este parâmetro é expresso como um número hexadecimal e deve ser encarado como um vetor de bits.
- b, c: Máscaras de bits para o *tempering*. Também expressos em hexadecimal.
- s, t: Bit *shifts* de *tempering*.
- u, l: Bit *shifts* adicionais.

A relação de recorrência linear matricial do gerador é dada por $x_{k+n} = x_{k+m} \oplus (x_k^u \mid x_{k+1}^l)A$, com $k \ge 0$. x_k^u denota os $w-r$ bits mais significativos de x_k (parte superior de x_k), x_{k+1}^l os r bits menos significativos de x_{k+1} (parte inferior de x_{k+1}) e o operador $\mid$ representa a concatenação de bits que forma uma palavra de comprimento w. [24] A é a matriz de *twisting*. Ela é constante, de tamanho $w \times w$, tem elementos em F_2 e está na forma racional normal:

$$A = \begin{pmatrix} 0 & I_{w-1} \\ a_0 & (a_1 \cdots a_{w-1}) \end{pmatrix}$$

O produto de um vetor x pela matriz A pode ser computado de maneira extremamente eficientemente devido ao fato de A estar na forma racional normal: [23]

$$xA = \begin{cases} x >> 1, & x_0 = 0, \\ (x >> 1) \oplus a, & x_0 = 1 \end{cases}$$

Antes de serem efetivamente considerados como saída do gerador, os números da seqüência pseudo-aleatória passam por um processo denominado *tempering* para aumentar sua qualidade. O *tempering* é uma multiplicação por uma matriz T, a matriz da transformada de tempering: $x \to z = xT$. [22] A computação do *tempering* é eficientemente efetuada pela aplicação seqüencial das seguintes

operações bit a bit: $y = x \oplus (x >> u)$, $y = y \oplus ((y << s) \& b)$, $y = y \oplus ((y << t) \& c)$ e $z = y \oplus (y >> l)$. Só então z pode ser considerado como um valor de saída do gerador. [22][24]

O gerador MT19937, a variante do Mersenne Twister a ser implementada, possui os coeficientes $(w, n, m, r, a) = (32, 624, 397, 31, 9908B0DF_{16})$ para a relação de recorrência e $(u, s, b, t, c, l) = (11, 7, 9D2C5680_{16}, 15, EFC60000_{16}, 18)$ para o *tempering*.

3.2. Atividades Realizadas

A maior parte do tempo dedicado ao projeto foi gasto com estudos teóricos, sendo o restante utilizado para projetar a arquitetura probabilística e o hardware do MT19937. As seguintes áreas da computação e da matemática foram investigadas a fim de se obter a bagagem teórica tão necessária para a realização do projeto: teoria das probabilidades [14][25], robótica probabilística [11][27][28][29][30][31][32] e robótica em geral [21], matemática discreta/álgebra [22][23][24], geração de variáveis aleatórias [7][15][17][18][22][23][24] e computação reconfigurável. [5][8][10]

3.3. Resultados Obtidos

Uma proposta inicial de uma implementação paralela do Mersenne Twister MT19937 é apresentada a seguir de maneira *bottom-up. Os* primeiros blocos de hardware que seguem representam as operações mais elementares; conforme os blocos forem se sucedendo o nível de abstração aumenta e o sistema se aproxima mais do comportamento seu global. Para efeito de clareza, nenhum sinal de *clock* ou *reset* foi incorporado aos diagramas de bloco.

A relação de recorrência do gerador (figura 11) e o processo de *tempering* (figura 12) foram projetados com lógica puramente combinacional. Utilizando-se destes blocos e de uma memória de múltiplas portas (três leituras e uma escrita) para armazenas o estado do gerador faz-se um MT19937 capaz de gerar um número aleatório por pulso de *clock* (figura 13), conforme proposto por KONUMA e ICHIKAWA. [16] Se uma memória convencional fosse utilizada o gerador

necessitaria de quatro pulsos de *clock* para produzir relação de recorrência. A implementação paralela (figura 14) baseou-se na independência de certos termos da relação de recorrência e no princípio do paralelismo por replicação (computação espacial) para gerar seis números aleatórios com um único *clock*. Decidiu-se que a implementação paralela deveria gerar seis números aleatórios, nem mais ou menos, pois qualquer movimento em três dimensões pode ser descrito com exatamente seis variáveis (*6DoF*): coordenadas *x, y e z* para posição e os ângulos de *yaw*, *pitch* e *roll*. [21]

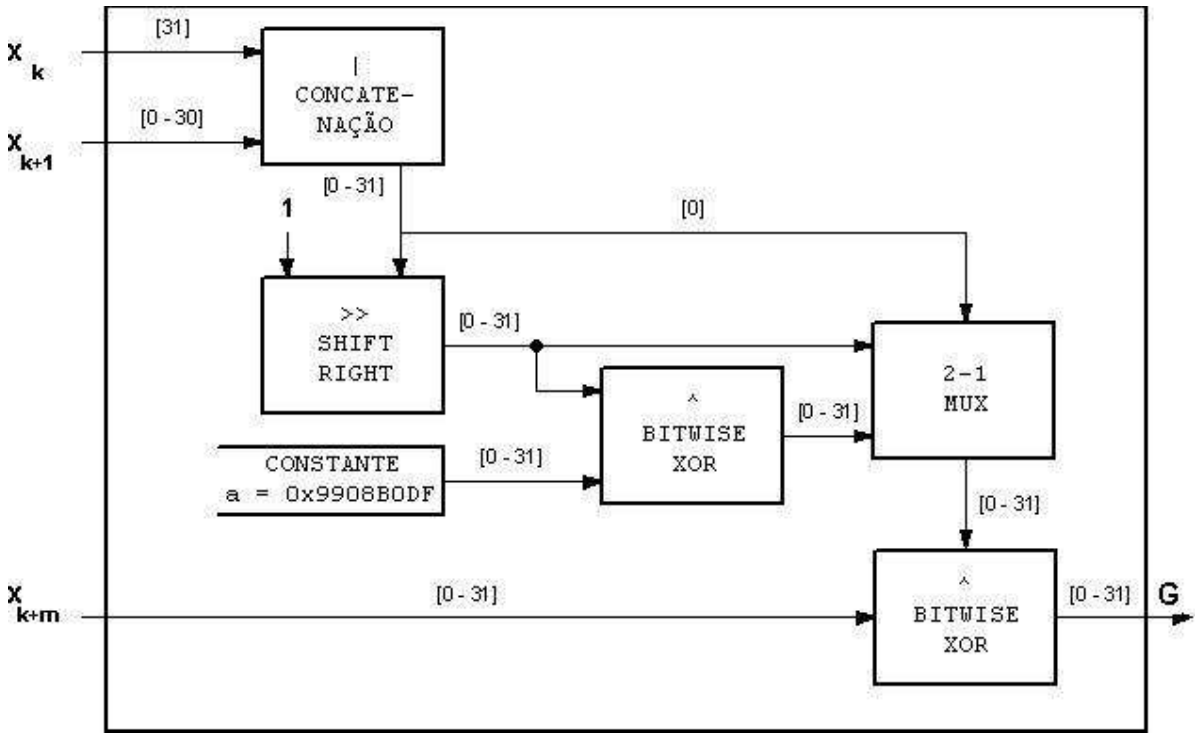


Figura 11: Relação de recorrência do MT 19937.

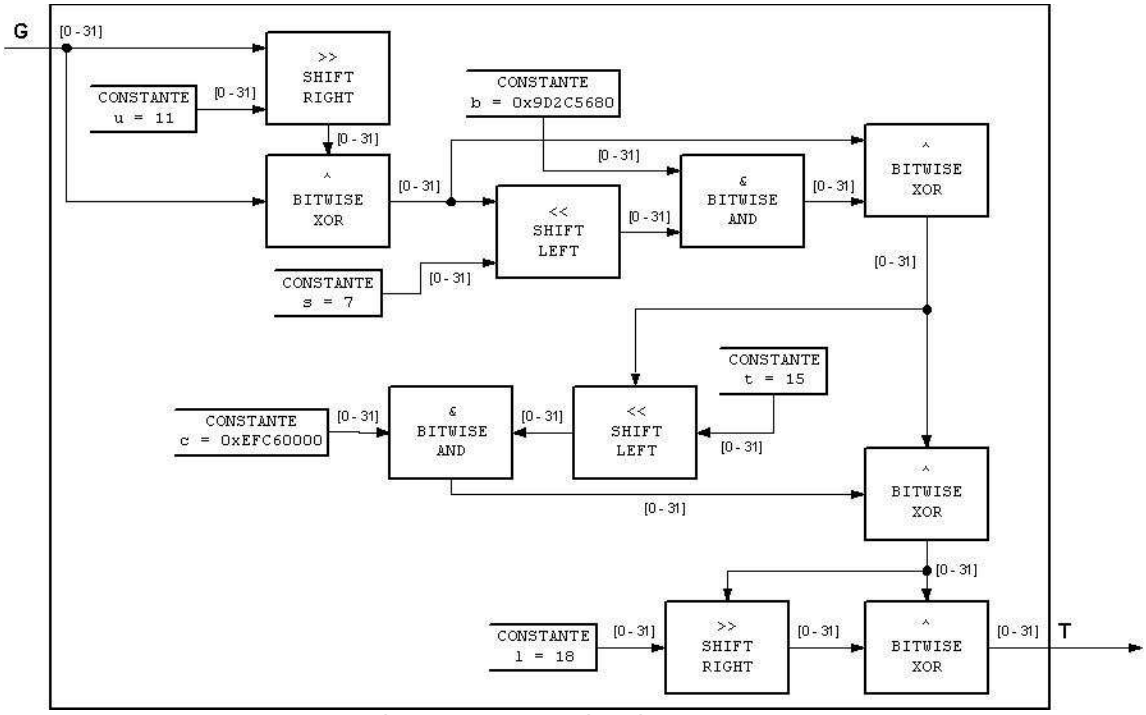


Figura 12: Tempering do MT19937.

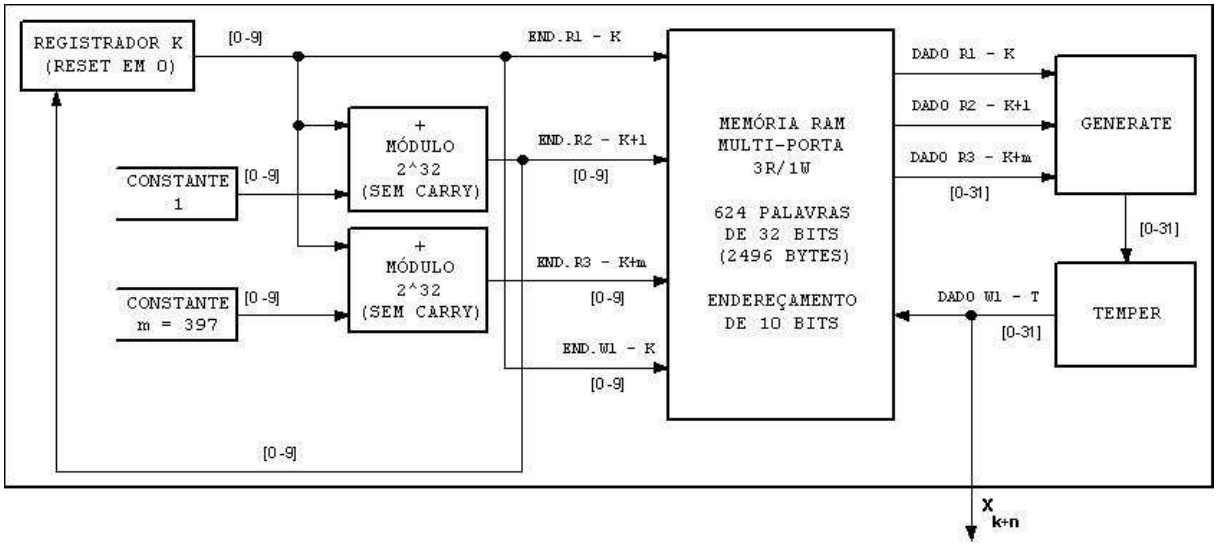


Figura 13: Implementação seqüencial do MT19937

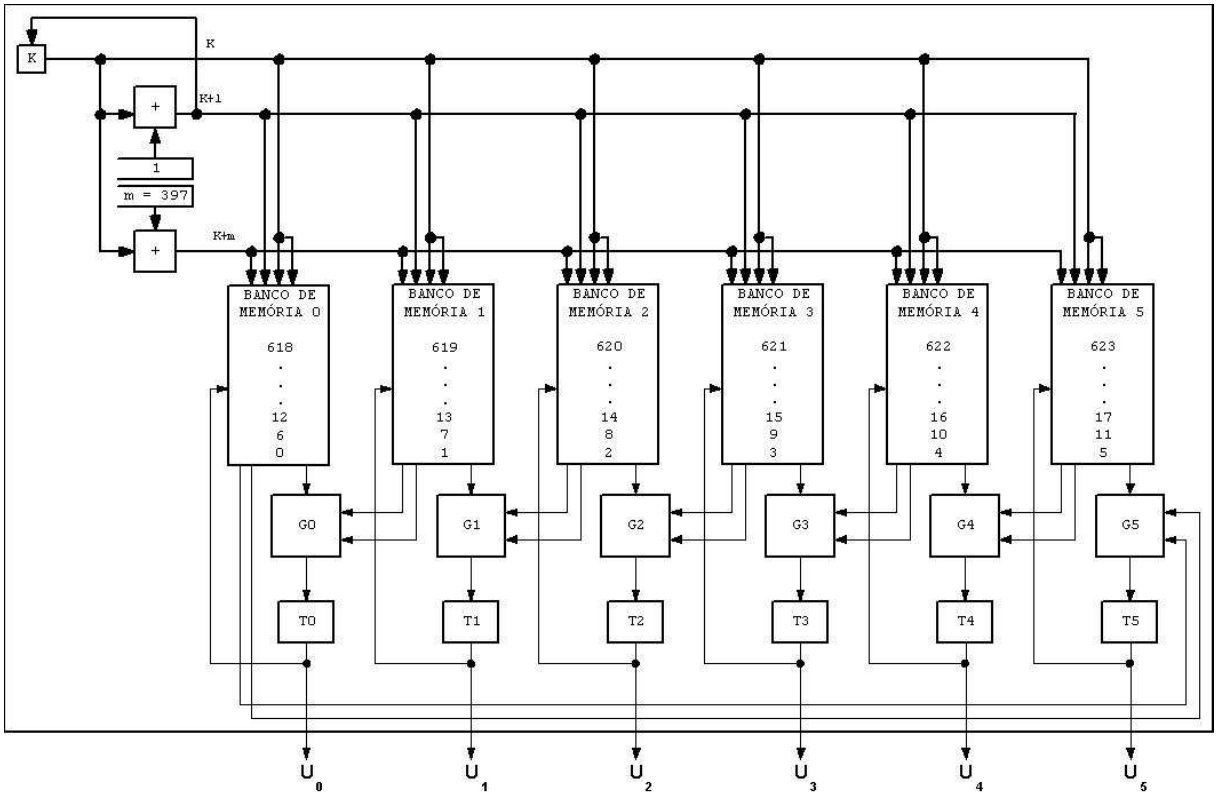


Figura 14:Implementação paralela do MT19937.

Considerando x um número gerado pelo Mersenne Twister e a transformação $x/2^w$, tem-se que a seqüência produzida segue distribuição uniforme padrão, isto é, uniforme no intervalo $[0,1]$. [17][18] Geralmente, as variáveis aleatórias utilizadas pelos algoritmos da robótica probabilística são modeladas segundo distribuições gaussianas [32]. CUI *et al* sugerem um gerador de números aleatórios universal (*URNG*) especificamente projetado para executar em *FPGAs* capaz de gerar variáveis aleatórias com as distribuições normal, exponencial e de Rayleigh a partir de distribuições uniformes. [9] O *URNG* computa as demais distribuições através de relações matemáticas em cascata partindo da distribuição uniforme. Se u_1 e u_2 são variáveis aleatórias com distribuição $U(a=0,b=1)$, tem-se que: [9]

$$\begin{cases} n_1 = \sqrt{-2\ln(u_1)}.\cos(2\pi u_2) \sim N(\mu=0,\sigma^2=1) \\ n_1 = \sqrt{-2\ln(u_1)}.\cos(2\pi u_2) \sim N(\mu=0,\sigma^2=1) \\ e = -\ln(u_1) \sim Exp(\lambda=1) \\ r = \sqrt{-2\ln(u_1)} \sim Ray(2\sigma^2=1) \end{cases}$$

As funções trigonométricas, raiz quadrada e logaritmo são calculados pelo algoritmo *CORDIC* (*COordinate Rotation DIgital Computer*). [3] Este algoritmo é simples, eficiente e normalmente usado com *FPGAs*. Não é do escopo desta monografia descrever o *CORDIC*, mas é importante citar que o LCR atualmente o está implementando totalmente em hardware para ser utilizado em outros trabalhos. Ao incorporar o *URNG* a arquitetura probabilística capturando a saída do MT19937 paralelo (figura 15) torna-se possível gerar simultaneamente seis distribuições gaussianas padrão, uma para cada grau de liberdade de um robô movimentando-se em um ambiente tridimensional.

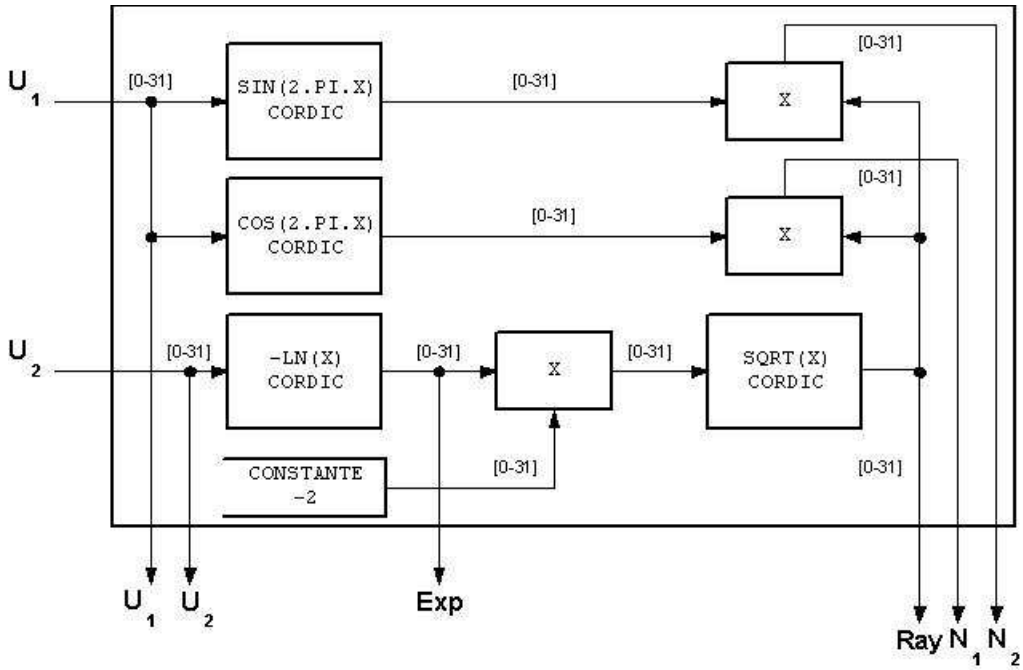


Figura 15: Gerador universal de números aleatórios (URNG).

A capacidade de armazenar os números gerados para uso posterior pode ser conseguida simplesmente acoplando-se uma memória ao sistema de geração descrito até o momento (figura 16). Isso também possibilita que grandes quantidades de números aleatórios possam ser transmitidas para outros sistemas via conexão paralela ou serial. Diferentes instruções personalizadas especificamente projetadas para manipular os números aleatórios armazenados na memória podem ser inseridas na *ULA* do Nios II (figura 17), provendo o programador de sistemas robóticos embarcados com muitos recursos úteis. [1]

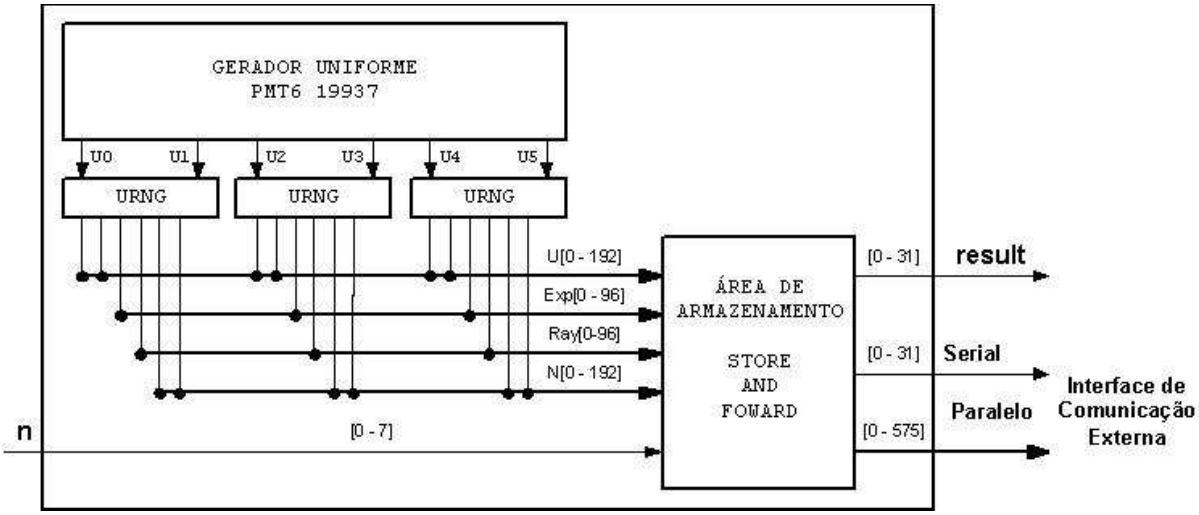


Figura 16: Bloco de instrução personalizada para o processador Nios II.

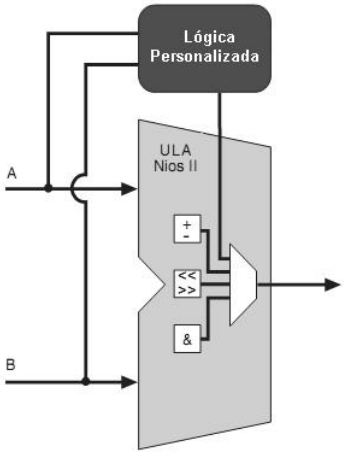
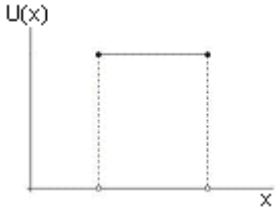


Figura 17: Integração da instrução personalizada à ULA do Nios II.

Segue um quadro resumo com das distribuições de probabilidade geradas com a combinação do *URNG* e o MT19937 paralelo:

Uniforme



$$U_n(x) = U(a=0, b=1, x) =$$

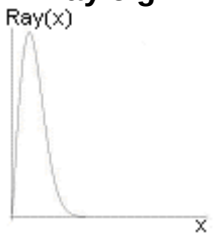
$$\begin{cases} 1, & 0 < x < 1, \\ 0, & c.c. \end{cases}$$

Exponencial

$$Exp_n(x) = Exp(\lambda = 1, x) =$$

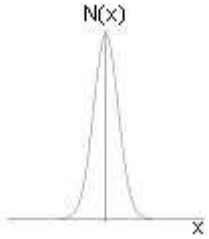
$$\begin{cases} e^{-x}, & x \geq 0, \\ 0, & c.c. \end{cases}$$

Rayleigh



$$Ray_n(x) = Ray(\sigma = \sqrt{1/2}, x) = \begin{cases} 2x\exp\left(-x^2\right), & x \geq 0, \\ 0, & c.c. \end{cases}$$

Normal



$$N_n(x) = N(\mu = 0, \sigma^2 = 1, x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right)$$

3.4. Dificuldades e Limitações

A maior dificuldade encontrada durante o desenvolvimento do projeto certamente foi a obtenção do entendimento pleno de alguns conceitos matemáticos, sem os quais não seria possível desenvolver o trabalho. Outra barreira significativa foi a grande extensão do conteúdo relacionado ao projeto; muitas concepções tiveram que ser compreendidas antes que qualquer resultado pudesse de fato se concretizar. De certa maneira, o caráter inovador do projeto em unir robótica probabilística e computação reconfigurável tornou-o mais desafiador porém não menos interessante.

4. Conclusões e Trabalhos Futuros

Notou-se que para se implementar um controle robótico probabilístico embarcado eficiente é fundamental partir de um bom gerador de números aleatórios implementado em hardware. O gerador proposto na monografia parte do estado da arte da área (em software) e mapeia o algoritmo utilizado para hardware fazendo uso intensivo de processamento paralelo e computação espacialmente distribuída. Apesar da proposta não ter sido implementada, ela se demonstra adequada e promissora para computação embarcada de alto desempenho.

Há muitos trabalhos que podem partir desta monografia e continuar a desenvolver o tema aqui apresentado. São apresentadas algumas propostas em ordem de relevância:

- Implementar e testar a arquitetura proposta.
- Realizar integração com o Nios II através da ferramenta *SoPC Builder* da Altera.
- Possibilitar que o usuário do sistema parametrize livremente as distribuições de probabilidade das quais são sorteados os números aleatórios.
- Implementar novas distribuições de probabilidade no *URNG*.
- Incorporar características da *CES* ao hardware descrito para que no futuro um compilador da linguagem possa ser desenvolvido para a arquitetura reconfigurável apresentada.

Referências Bibliográficas

[1] ALTERA CORPORATION. Nios II Custom Instruction User Guide, 2007. Disponível na Web em http://www.altera.com/literature/lit-nio2.jsp, Junho 2007.

[2] ALTERA CORPORATION. Nios II Processor Reference Handbook, 2007. Disponível na Web em http://www.altera.com/literature/lit-nio2.jsp, Junho 2007.

[3] ANDRAKA, Ray. A Survey of CORDIC Algorithms for FPGA Based Computers. Proceedings of the Sixth ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'98), 1998.

[4] ARAGÃO, Antônio; ROMERO, Roseli; MARQUES, Eduardo. Computação Reconfigurável Aplicada à Robótica. In CORE-2000 Workshop em Computação Reconfigurável, Marília. Livro Computação Reconfigurável: Experiências e Perspectivas, Rio de Janeiro/RJ, Editora Brasport, v.1. p.184-188, 2000.

[5] BONDALAPATI, Kiran; PRASANNA, Viktor. Reconfigurable Computing Systems. Proceedings of the IEEE v.90, n.7, p.1201-1217, July 2002.

[6] BROWN, Stephen; VRANESIC, Zvonko. Fundamentals of Digital Logic with VHDL Design, McGraw Hill, 2000.

[7] CHU, Pong; JONES, Robert. Design Techniques of FPGA Based Random Number Generator. MAPLD 1999 - Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference, 2nd, Johns Hopkins University, APL, Laurel, MD, United States, p.28-30, September 1999.

[8] COMPTON, Katherine; HAUCK, Scott. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, v.34, n.2, p.171-210, June 2002.

[9] CUI, Wei; LI, Chengshu; SUN, Xin. FPGA Implementation of Universal Random Number Generator. Proceedings of the 7th International Conference on Signal Processing, v.1, p.495-498, September 2004.

[10] DEHON, André; WAWRZYNEK, John. Reconfigurable Computing: What, Why and Implications for Design Automation. Design Automation Conference, Proceedings 36th, p.610-615, June 1999.

[11] DELLAERT, Frank; FOX, Dieter; BURGARD, Wolfram; THRUN, Sebastian. Monte Carlo Localization for Mobile Robots. Proceedings of IEEE on Robotics and Automation, v.2, p.1322-1328, 1999.

[12] FISHER, Joseph; FARABOSCHI, Paolo; YOUNG, Cliff. Embedded Computing - A VLIW Approach to Architecture, Compilers and Tools. Morgan Kaufmann Publishers, 2005.

[13] GONÇALVES, Richard. Architect-R: Uma Ferramenta para o Desenvolvimento de Robôs Móveis Reconfiguráveis, 2001. Exame de Qualificação de Mestrado Ciências de Computação e Matemática Computacional, Universidade de São Paulo.

[14] JAYNES, Edwin. Probability Theory: The Logic of Science. Disponível na Web em http://omega.albany.edu:8080/JaynesBook.html, Junho 2007.

[15] KNUTH, Donald. The Art of Computer Programming Vol. 2: Seminumerical Algorithms, 2nd Edition. Addison-Wesley, Reading, 1981.

[16] KONUMA, Shiro; ICHIKAWA, Shuichi. Design and Evaluation of Hardware Pseudo-random Number Generator MT19937. IEICE Transactions on Information and Systems, v.E88-D, n.12, p.2876-2879, 2005.

[17] L'ECUYER, Pierre. A Tutorial on Uniform Variate Generation. Proceedings of the 21st Conference on Winter Simulation, Washington, DC, United States, p.40-49, 1989.

[18] L'ECUYER, Pierre. Uniform Random Number Generators. Proceedings of the 30th Conference on Winter Simulation, Washington, DC, United States, p.97-104, 1998.

[19] LEONG, P.H.W.; TSOI, K.H. Field Programmable Gate Array Technology for Robotics Applications. IEEE International Conference on Robotics and Biomimetics (ROBIO), p.295-298, July 2005.

[20] MARSAGLIA, George. The Marsaglia Random Number CD-ROM with the Diehard Battery of Tests of Randomness. Supercomputer Computations Research Institute and Department of Statistics, Florida State University. Disponível na Web em http://www.csis.hku.hk/~diehard/, Junho 2007.

[21] MATARIĆ, Maja. The Robotics Primer. MIT Press, Cambridge MA, London – England, 2007.

[22] MATSUMOTO, Makoto; KURITA, Yoshiharu. Twisted GFSR Generators II. ACM Transactions on Modeling and Computer Simulation (TOMACS) v.4, p.254-266, 1994.

[23] MATSUMOTO, Makoto; KURITA, Yoshiharu. Twisted GFSR Generators. ACM Transactions on Modeling and Computer Simulation (TOMACS), v.2, n.3, p.179-194, 1992.

[24] MATSUMOTO, Makoto; NISHIMURA, Takuji. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudorandom Number Generator. ACM Transactions on Modeling and Computer Simulation (TOMACS), v.8, n.1 Special Issue on Uniform Random Number Generation, p.3-30, 1998.

[25] NIST/SEMATECH e-Handbook of Statiscal Methods. Disponível na Web em http://www.itl.nist.gov/div898/handbook/, Junho 2007.

[26] PEDRONI, Volnei. Circuit Design with VHDL. MIT Press, 2004.

[27] ROY, Nicholas; THRUN, Sebastian. Coastal Navigation with a Mobile Robot. Advances in Neural Information Processing Systems (NIPS'99), 1999.

[28] THRUN, Sebastian. A Framework for Programming Embedded Systems: Initial Design and Results. Technical Report CMU-CS-98-142, Carnegie Mellon University, Pittsburgh, PA, 21, 1998.

[29] THRUN, Sebastian. Is Robotics Going Statistics? The Field of Probabilistic Robotics. Communications of the ACM, March 2001.

[30] THRUN, Sebastian. Probabilistic Algorithms in Robotics. AI Magazine, v.21, n.4, p.93-109, 2000.

[31] THRUN, Sebastian. Towards Programming Tools for Robotics that Integrate Probabilistic Computation and Learning. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA, United States, v.1, p.306-312, 2000.

[32] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. Probabilistic Robotics. MIT Press, Cambridge MA, 2005.

Lista de Siglas

6DoF – Six Degrees of Freedom

ASIC – Application Specific Integrated Circuit

CES – C++ for Embedded Systems

CORDIC – COordinate Rotation DIgital Computer

CPLD – Complex Programmable Logic Device

DSP – Digital Signal Processor

EDA – Eletronic Design Automation

FPGA – Field Programmable Gate Array

GAL – Generic Array Logic

GFSR – Generalised Feedback Shift Register

GPS – Global Positioning System

LASER – Light Amplification by Stimulated Emission of Radiation

LCG – Linear Congruential Generator

LFG – Lagged Fibonacci Generator

LUT – Look-up Table

MT – Mersenne Twister

NRE – Non-Recurring Engineering

PAL – Programmable Array Logic

PRNG – Pseudo Random Number Generator

RISC – Reduced Instruction Set Computing

SoPC – System on Programmable Chip

SRAM – Static Random Access Memory

ULA – Unidade Lógica Aritmética

URNG – Universal Random Number Generator.

VCMU – Vehicle Control and Monitoring Unit

VHDL – VHSIC Hardware Description Language

VHSIC – Very High Speed Integrated Circuits

VLIW – Very Large Instruction Word