UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Departamento de Ciências da Computação e Estatísticas

Implementação de Mapas Topológicos para Navegação de Robôs Móveis baseadas em Computação Reconfigurável

Jean Miler Scatena

Orientador: Prof. Dr. Eduardo Marques

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

São Carlos, janeiro de 2003.

Agradecimento

Agradeço a todos que estiveram presentes e que me ajudaram direta e indiretamente neste grande passo de minha vida. Agradeço a Deus, à minha família, ao meu segundo pai, dentro da USP, (meu Orientador) Eduardo, ao meu Professor, Coordenador e amigo Pacheco e principalmente a uma pessoa que sempre me apoiou em tudo que construísse, a minha namorada Carla.

Resumo

O presente trabalho é vinculado a duas áreas de grande pesquisa e enfoque na comunidade cientifica, a área de navegação de robôs móveis e a área de computação reconfigurável. Este trabalho tem como principal finalidade implementar uma técnica de mapeamento para o sistema de navegação de um robô móvel, em hardware reconfigurável, objetivando a melhora do desempenho na execução da técnica chamada mapeamento topológico, além de fornecer a capacidade de um sistema robótico poder-se auto reconfigurar em tempo real.

Para que seja realizada esta tarefa, foram necessários pesquisas e estudos a estes dois assuntos, podendo ser encontrada uma explanação dos mesmos nos capítulos 3 e 4.

O primeiro tema abordado foi o sistema de navegação de robôs móveis com análise inicial sobre as formas de navegação e mapeamento associadas com o estudo dos ambientes que serão realizadas as tarefas de navegação.

O segundo tema abordado é sobre sistemas reconfiguráveis que tem como ênfase à construção, implementação, reconfiguração assim como os principais fabricantes.

Depois de realizado todo o estudo inerente à pesquisa, anteriormente citado, é implementado um sistema de navegação de robôs móveis em um hardware reconfigurável utilizando o conjunto de ferramentas de desenvolvimento de hardwares reconfiguráveis da empresa chamada Altera.

Abstract

The present work is related to two great research and focus areas at scientific community, the mobile robots navigation and reconfigurable computing area. This work has as main purpose to implement one mapping technique that was a part of a navigation system using reconfigurable hardware, with the objective of improving the execute performance of the technique called topological mapping, beyond to provide with of the capacity of a robotic system to can self-reconfiguration in real time.

So that this task is accomplished, they were necessary researches and studies of these two subjects, more information can be found in the 3rd and 4th chapter.

The first topic approached theme was the navigation system for mobile robots with an initial analysis on the navigation types and mapping, associated with the study of the environments in what the navigation tasks will be accomplished.

The second issue was about the reconfigurable of the systems with spot in the construction, implementation, reconfiguration, as well as, the main manufacturers.

After having accomplished the whole inherent study to the research, previously mentioned, a system of navigation for mobile robots is implemented in a reconfigurable hardware using the group of tools of development of reconfigurabel hardwares of the Altera Company.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1 - Introdução

1.1. Motivação

Os sistemas reconfiguráveis são hoje a revolução da área de hardware, tendo várias características de inovação, inclusive no método de roteamento dos circuitos e na velocidade desses novos sistemas.

Grande parte dos sistemas robóticos existentes hoje são ainda desenvolvidos apenas em software, tendo muitas vezes que exigir um grande trabalho computacional das máquinas que estes softwares utilizarão. Já nos sistemas transpostos para o hardware reconfigurável este desempenho aumenta e este hardware possui todas as principais vantagens das máquinas de propósito gerais.

No contexto de robótica, existem hoje duas áreas que estão repercutindo tanto no ramo industrial como no ramo de pesquisas, sendo essas áreas:

- *Projeto de robôs móveis*, o quais conseguem se locomover e interagir com o ambiente (dinâmicos) em que se encontram;
- *Computação Reconfigurável* esta se desenvolvendo e se concretizando principalmente na área espacial. Tudo isso pelo fato de sua portabilidade e flexibilidade, as quais possibilitam a troca do hardware remotamente (via rádio, internet, etc.).

1.2. Objetivo

Este projeto tem como principal finalidade aprimorar os conhecimentos de sistemas de navegação e mapeamento. Propõe-se estudar, juntamente com os sistemas de mapeamento, as técnicas de implementação do sistema de navegação de robôs móveis em hardwares reconfiguráveis. Como em qualquer área de desenvolvimento, para

aprimorar ou adquirir novos conhecimentos é necessário passar por diversas fases, sendo que é de se esperar que tais fases sejam cumpridas. Estas fases envolvem um background sobre o assunto a ser implementado, juntamente com o estudo das partes já construídas, a elaboração de uma proposta visando alcançar o objetivo esperado, o desenvolvimento do projeto e sua simulação e teste.

Mais especificamente, este trabalho se baseará na implementação de um mapa topológico baseado em detecção de espaços livres [DUC98; DUC99; DUC99A; DUC2000], conforme descrito no item 3.2 e 8.2, utilizando-se o ambiente de desenvolvimento de sistema de robôs móveis chamado Saphira e o ambiente de desenvolvimento de hardware reconfigurável chamado Excalibur (Altera).

1.3. Justificativa

Com as novas tecnologias existentes na área de desenvolvimento de hardware, principalmente o hardware reconfigurável pelo fato da necessidade de computadores mais velozes, tem se explorada maciçamente a nova tecnologia, o hardware reconfigurável, pois ele se torna um sistema viável pelo fato de poder ser reconstruído sem a modificação do hardware físico [VIL97].

Na área de hardware ocorreram grandes mudanças na sua forma de estruturação e desenvolvimento, isso se deve graças à computação reconfigurável, que revolucionou a área de circuitos integrados, facilitando a construção de hardware através de software, com uma grande vantagem, que é o acoplamento dos dados em módulos reconfiguráveis sem que haja a mudança de hardware (modificação física), com tempos de atualização e respostas muito rápida [VIL97].

Já na área de projeto de navegação de robôs móveis, as técnicas de inteligência artificial predominam na maior parte dos aspectos, evoluindo sempre suas técnicas para aperfeiçoar seus projetos. Porém a área de navegação de robôs possui um grande problema, que é o tempo de resposta, isto é, o tempo em que uma técnica ou processo demora em realizar todo o seu processamento e devolver o resultado. Este problema é

causado pela complexidade dos algoritmos utilizados para a navegação de robôs móveis, exigindo assim máquinas mais velozes [YAM98].

A pesquisa aqui apresentada tem como função para a conclusão de seu objetivo, analisar as possíveis formas de implementação de mapas topológicos em hardware reconfigurável, para melhorar o tempo de resposta e a modularidade desse sistema em hardware reconfigurável.

1.4. Delimitação

A delimitação da pesquisa realizada neste trabalho se restringe à proposta de construção de um sistema o qual irá criar um grafo que especificará a rota percorrida por um robô móvel. Mais detalhadamente o sistema apenas informará a orientação do robô, tendo como principais diretivas a direção (Centro, Esquerda ou Direita), o ângulo de conversão caso haja necessidade (30, 0 ou -30 graus) e a distância percorrida por uma direção.

Este sistema é implementado em hardware reconfigurável o qual simulará os sensores de um robô móvel e transportará estas leituras para o sistema de mapeamento topológico em hardware. Estas leituras passarão por uma rede neural artificial desenvolvida em um hardware dedicado que retornará ao sistema de mapeamento a melhor rota a ser tomada.

O sistema implementado em hardware é apenas uma parte do desenvolvimento da navegação encontrada em diversos robôs móveis, sendo que para o desenvolvimento completo do sistema de navegação deverão ser considerados detalhes sobre o robô utilizado.

1.5. Projeto ARMOSH

O projeto ARMOSH é um projeto de desenvolvimento de um robô móvel reconfigurável, tendo como principais colaboradores os integrantes do laboratório de computação reconfigurável (LCR) e do laboratório de inteligência computacional (LABIC) da Universidade de São Paulo – USP – Campi São Carlos [ARM2002].

Este projeto fundamenta-se nas pesquisas de navegação de robôs móveis, FPGA, Sistemas On Chip e reconfigurabilidade de hardware.

1.5.1. Objetivos

O principal objetivo do projeto ARMOSH propõe a análise e o desenvolvimento gradual de um sistema complexo que envolve robôs móveis, sendo em seu estado inicial, o estudo e implementação de algoritmos de aprendizado de robôs móveis tanto no nível de software quanto no nível de hardware, visando uma comparação de desempenho dos principais algoritmos de aprendizado na versão de software e hardware [ARM2002].

O projeto tem a finalidade de buscar novas alternativas na área, inicialmente através de estudos, trabalhando com implementações tanto no nível de software quanto no nível de hardware dos principais algoritmos de aprendizado de máquina, onde serão analisados [ARM2002]:

- *Os principais algoritmos para navegação*, levando-se em conta a dependência do número de sensores do robô (sonar, infravermelho, a laser ou câmera de vídeo) e a definição do ambiente (fechado, aberto, desconhecido);
- *Os algoritmos que envolvem aprendizado através a Programação Dinâmica*, as redes neurais, com aprendizado por reforço;
- *Os algoritmos que envolvem reconhecimento de objetos via Redes Neurais*, via técnicas clássicas de processamento de imagens (tais como, forma, textura, cor, etc...), em ambientes estáticos e dinâmicos.

Os estudos levantados neste projeto serão direcionados para a sua evolução futura, através da construção de uma biblioteca dos algoritmos estudados em sua versão hardware/software e na montagem de um kernel, a longo de prazo, que controlará de forma inteligente a execução destes algoritmos [ARM2002].

Com essas implementações tem-se um sistema em tempo real que poderá decidir qual algoritmo é mais adequado para realização de uma determinada tarefa, utilizando ainda, a melhor implementação (via hardware/software) do algoritmo escolhido. A Figura 1 descreve o funcionamento do projeto ARMOSH [ARM2002].

Esta pesquisa auxiliará na decisão de qual método será utilizado no desenvolvimento do projeto ARMOSH, tendo em vista demonstrar os prós e contras deste método de navegação, bem como a sua forma de interação em um sistema de hardware reconfigurável.

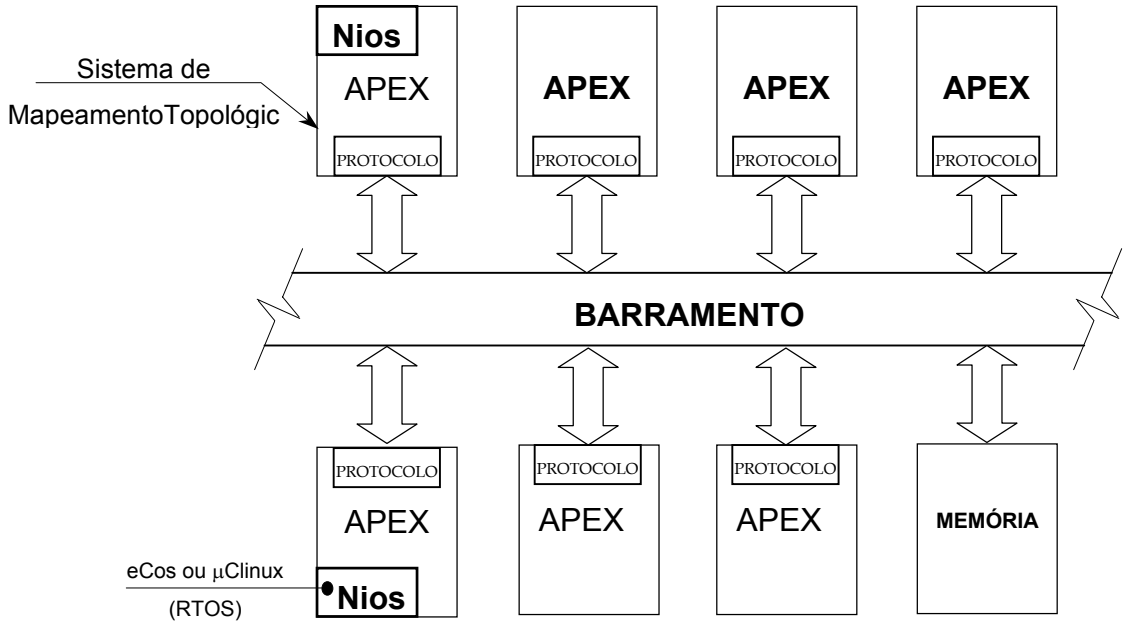


Figura 1 - O Hardware do Projeto ARMOSH [ARM2002].

1.6. Apresentação

O trabalho está distribuído em seis capítulos e três apêndices. Neste capitulo foi apresentado à motivação do trabalho, os objetivos da pesquisa, sua justificativa e a sua delimitação. Os demais capítulos estão organizados da seguinte forma:

- *Capítulo 2*: É apresentado o sistema autônomo que descreve uma breve introdução de sistema autônomo, robôs móveis e inteligência artificial.
- *Capítulo 3*: Aborda os sistemas de navegação utilizados por robôs móveis. Este capítulo descreve conceitos sobre o que é navegar, como navegar e quais as principais formas de navegação e mapeamento em robôs móveis. São apresentadas as principais características, vantagens e desvantagens dos métodos de mapeamento topológico e mapas baseados em grids (matrizes) juntamente com a descrição das pesquisas realizadas atualmente na área de mapeamento topológico.
- *Capítulo 4*: Aborda um breve histórico sobre a nova forma de desenvolvimento de hardware, que é chamado de hardware reconfigurável. Este capítulo apresenta os principais fabricantes e dispositivos de computação reconfigurável, bem como as suas formas de desenvolvimento.
- *Capítulo 5*: É apresentado o sistema Saphira que simula o ambiente de um robô móvel, ou seja, esse software simula as leituras dos sensores com suas imperfeições, o mapa de navegação com obstáculos e os problemas resultantes da navegação em um sistema real.
- *Capítulo 6*: Apresenta como foi desenvolvido o sistema de mapeamento topológico em hardware reconfigurável juntamente com todos os seus detalhamentos. Este capítulo descreve também como foram desenvolvidos a detecção de espaços livres, a rede neural artificial e o mapa topológico.
- *Capítulo 7*: Aborda as conclusões e as dificuldades encontradas neste projeto.
- *Capítulo 8*: Apresenta as previsões para trabalhos futuros.

- *Apêndices*: Apresentam os códigos de desenvolvimento do sistema de mapeamento no ambiente Excalibur.

Capítulo 2 - Robôs

Desde os primórdios dos tempos, o homem vem aprimorando idéias e ideais sobre um conjunto independente de máquinas as quais desenvolveriam tarefas que os seres humanos não conseguiriam desenvolver e tarefas que não tinham nenhum valor intelectual. A esse conjunto de máquinas independentes foi-se então chamado de robôs [BOR96].

Numa primeira etapa houve a necessidade do aperfeiçoamento dos robôs visando à área industrial, isto trouxe grandes benefícios como a melhoria da eficácia, qualidade, redução da mão-de-obra, além de mais eficiência, confiabilidade e redução de custos. Essas vantagens incluem ainda a capacidade de realizar tarefas para as quais os humanos teriam grandes dificuldades, como por exemplo, a remoção de humanos de tarefas em ambientes perigosos e tarefas repetitivas, que exigem pouca ou nenhuma utilização intelectual [ARA2000B].

Seguindo esta linha de evolução, surge um novo campo de atuação destes sistemas que é ampliado a níveis superiores ao da área industrial, que é a interação entre os robôs e os seres humanos, como por exemplo, os sistemas ALVINN, TESEO e FINALLE descritos no item 3.1.3 [MIT2001].

2.1. Robôs Móveis

Nesta nova etapa de evolução dos sistemas robóticos, muitos pesquisadores têm concentrado grandes esforços na construção de robôs móveis, introduzindo nestas máquinas as capacidades de mobilidade e autonomia para reagir adequadamente em um ambiente, o que abre um vasto campo de novas aplicações e conseqüentemente muitos desafios[ARA2000B].

Dentre algumas possibilidades de uso de robôs móveis pode-se citar o transporte de materiais, combate a incêndios, desativação de explosivos, vigilância de armazéns, viagens submarinas, aeroespaciais e prestação de serviços, onde haja interação com pessoas para o auxílio de tarefas rotineiras.

Para interagir em um ambiente desestruturado, desconhecido ou dinâmico, um robô móvel deve ser capaz de perceber as circunstâncias em sua volta e a partir disto, gerar ações que serão apropriadas para o ambiente e para os objetivos de seu sistema. As ações utilizadas neste(s) ambiente(s) são técnicas que utilizam vários recursos da computação, como por exemplo, a inteligência artificial, a qual fornece detalhes precisos do ambiente (mundo) em que o sistema esta interagindo. A partir destes dados é que o robô terá condição de realizar a(s) sua(s) tarefa(s) [ARA2000B].

2.2. Redes Neurais Artificiais

O grande desenvolvimento na área de inteligência artificial (IA) possibilita a criação de sistemas que possuem a capacidade de tomar decisões baseadas em um conhecimento pré-estabelecido.

Dentro do vasto conjunto de aplicações de inteligência artificial, têm-se as redes neurais artificiais, as quais constitui-se de um sistema distribuído composto por unidades de processamento simples conhecidos como neurônios artificiais (nós), que trabalham sobre determinadas funções matemáticas, onde normalmente essas funções não são lineares. Tais unidades são dispostas em uma ou mais camadas interligadas por um grande número de conexões, geralmente unidirecionais [PON98].

Na maioria dos modelos estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado no modelo e que servem também, para ponderar a entrada recebida por uma estrutura física concebida pela natureza [PON98]. Este tipo de estrutura é muito semelhante à estrutura do sistema nervoso do ser humano composto pelo cérebro e seus neurônios e tem como base para a construção de seus nodos a base dos neurônios humanos como ilustrado na Figura 2.

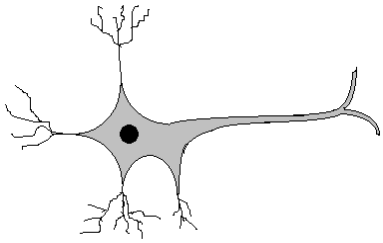


Figura 2 - Neurônio biológico humano [PON98]

A solução de problemas através de redes neurais artificiais é bastante atrativa, uma vez que possui uma semelhança com os neurônios do sistema nervoso humano, e pela representação internamente agregada ao seu paralelismo, o qual é inerente à sua arquitetura, a rede neural artificial demonstra um desempenho superior, se comparado aos modelos convencionais de softwares para a solução de problemas [PON98].

O procedimento usual na solução de problemas passa inicialmente por uma fase de aprendizagem, onde um conjunto de exemplos é apresentado para a rede, a qual extrai automaticamente as características necessárias para representar a informação fornecida. Essas características são utilizadas posteriormente para gerar respostas para o problema [PON98].

As redes neurais artificiais possuem a capacidade de aprender e de generalizar as informações obtidas através de exemplos, que são sem dúvida, um dos atrativos principais para a solução de problemas utilizando o sistema de redes neurais artificiais [PON98].

A generalização, que está associada à capacidade da rede aprender através de um conjunto reduzido de exemplos e posteriormente dar respostas coerentes para dados não conhecidos, é uma demonstração de que a capacidade das redes neurais artificiais vai muita além do que simplesmente mapear relações de entrada de saída. Elas são capazes de extrair informações não representadas de forma explícita através de exemplos. Não obstante, as redes neurais artificiais são capazes de atuar como mapeadores universais de funções multi-variáveis, com o custo computacional que cresce linearmente conforme o aumento do número de variáveis [PON98].

Outras características importantes são a capacidade de auto-organização e de processamento temporal que, aliadas àquelas citadas anteriormente, fazem das redes neurais artificiais uma ferramenta computacional extremamente poderosa e atrativa para a solução de problemas complexos.

2.2.1. Neurônio McCulloch e Pitts (MCP)

O primeiro modelo artificial de um neurônio biológico foi fruto do trabalho pioneiro de *Warren McCulloch* e *Walter Pitts* em 1943. McCulloch dedicou 20 anos tentando representar um evento no sistema nervoso. *McCulloch* atuava como psiquiatra e neuroanatomista. *Pitts*, um matemático recém graduado juntou-se a *McCulloch* em 1942. No trabalho publicado em 1943, "A Logical Calculus of the Ideas Immament in Nervous Activity", é apresentada uma discussão sofisticada de redes lógicas de nodos e novas idéias sobre máquinas de estados finitos. O trabalho de McCulloch e Pitts se concentra muito mais em descrever um modelo artificial de um neurônio e de apresentar as suas capacidades computacionais do que apresentar técnicas de aprendizado [ARA2000B, PON98].

O modelo de neurônio proposto por *McCulloch* e *Pitts* é uma simplificação do que se tinha a respeito do neurônio biológico naquela época. A sua descrição matemática resultou em um modelo com *n* terminais de entrada X_1, X_2, X_3... X_n (que representam os dendritos), e apenas um terminal de saída Y (representando o axônio). Para emular o comportamento das sinapses, os terminais de entrada do neurônio têm pesos acoplados W_1, W_2, W_3... W_n cujos valores podem ser positivos ou negativos dependendo das sinapses correspondentes serem inibitórias ou excitatórias. O efeito de uma sinapse particular *i* no neurônio pós-sináptico é dado por: X_iW_i,, onde os pesos determinam “em que grau“ o neurônio deve considerar sinais de disparo que ocorrem naquela conexão.

Para que esses valores combinados gerem um resultado esperado, todo neurônio deve possuir uma função de ativação e uma arquitetura. A função de ativação é uma função que decide se o neurônio deve ou não disparar (liberar a saída do neurônio), ela utiliza a soma ponderada dos valores de entrada (X_i) multiplicado pelos pesos

sinápticos (W_i) [ARA2000B, PON98].

Dentre os conjuntos de neurônios artificiais tem-se várias funções de ativação, as quais resultam diversas saídas, dependendo do problema a ser analisado. [ilustrado na Figura 3].

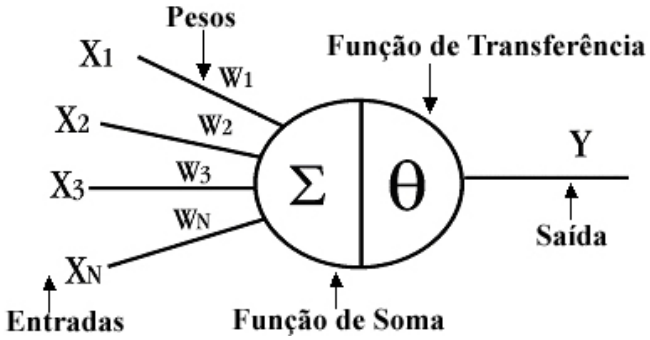


Figura 3 - Neurônio Artificial McCulloch Pitts [ARA2000B,PON98]

A arquitetura de uma rede neural é a estrutura na qual a rede neural será organizada. Deve-se prestar muita atenção na definição de qual arquitetura escolher, pois é a arquitetura que restringe o tipo de problema que pode ser tratado pela rede. Um exemplo disto, é que as redes de uma única camada de neurônios MCP (MacCulloch Pitts), só conseguem resolver problemas linearmente separáveis. Fazem parte da definição da arquitetura os seguintes parâmetros: número de camadas da rede, números de neurônios em cada camada, tipo de conexão, topologia e o aprendizado. Na definição de uma rede neural artificial o parâmetro que gera maior discussão gira em torno dos métodos de aprendizado, que treinam os nodos para que eles possam ser capazes de executar uma determinada função [ARA2000B, PON98].

O aprendizado de redes biológicas e artificiais veio a ser objeto de estudo somente alguns anos depois do trabalho de *McCulloch* e *Pitts*. O primeiro trabalho de que se tem notícia que tem ligação direta com aprendizado foi apresentado por *Donald Hebb*, em 1949. *Hebb* mostrou como a plasticidade da aprendizagem de redes neurais é conseguida através da variação dos pesos de entrada dos nós. Ele propôs uma teoria para explicar o aprendizado em nodos biológicos baseada no reforço das ligações sinápticas entre nodos excitados [ARA2000B, PON98].

Em 1958, *Frank Rosenblat*, utilizando a estrutura do neurônio MCP (*McCulloch* e *Pitts*) desenvolveu um novo modelo o qual chamou de perceptron. Este novo modelo possuía as características do neurônio MCP adicionando apenas os pesos ajustáveis. *Rosenblat* descreveu uma topologia com seu novo neurônio, uma estrutura de ligação entre esses neurônios e propôs ainda um algoritmo de treinamento para esta rede executar determinadas funções. Após seus testes com sua nova definição de neurônio, *Rosenblat* concluiu que seu perceptron era capaz de comportar-se como um classificador de padrões, comprovado mais adiante que este sistema era capaz de classificar classes que sejam linearmente separáveis [PON98].

Em 1969, *Minsky* e *Papert* realizaram um estudo sobre a rede de *Rosenblat* e descobriram que esta rede não era capaz de solucionar problemas não linearmente separáveis, pois esses problemas continham problemas na aprendizagem. Após a conclusão deste estudo, as pesquisas neste campo estão cada vez mais fracas, deixando assim o conceito de redes neurais adormecido até 1982, quando o pesquisador *John Hopfield* publicou um artigo que fazia uma relação entre redes recorrentes, as quais tinham a capacidade de aprender problemas com problemas de aprendizagem e propôs um algoritmo de treinamento para redes neurais artificiais (RNA) chamados de back-propagation. Este trabalho de *Hopfield* foi que motivou um novo interesse nas pesquisas sobre RNAs, pois este artigo trouxe uma revolução nas pesquisas de *Minsky* e *Papert*, mostrando que eles possuíam uma visão pessimista sobre o perceptron [PON98].

Dentre todos os estudos e pesquisas realizados pode-se estruturar uma rede neural através de sua função de ativação, ao seu número de camadas, ao tipo de conexão entre seus neurônios e quanto a sua conectividade.

Quanto a sua função de ativação tem-se [PON98]:

- *Função Linear* produz valores crescentes constantes conforme a variação das entradas. [Ilustrada na Figura 4.a];
- *Função Rampa* produz valores constantes em uma faixa, e neste caso a função pode ser uma rampa, visualizada na Figura 4.c;

- *Função Passo* produz saída positiva para valores maiores que zero e valores negativos para valores menores que zero. [Ilustrada na Figura 4.b];
- *Função sigmoidal* é uma função semilinear, limitada e monotônica. É possível definir várias funções sigmoidais. As funções sigmoidais são encontradas na modelagem de diversos modelos na mais variadas áreas. Esta função pode ser visualizada na Figura 4.d.

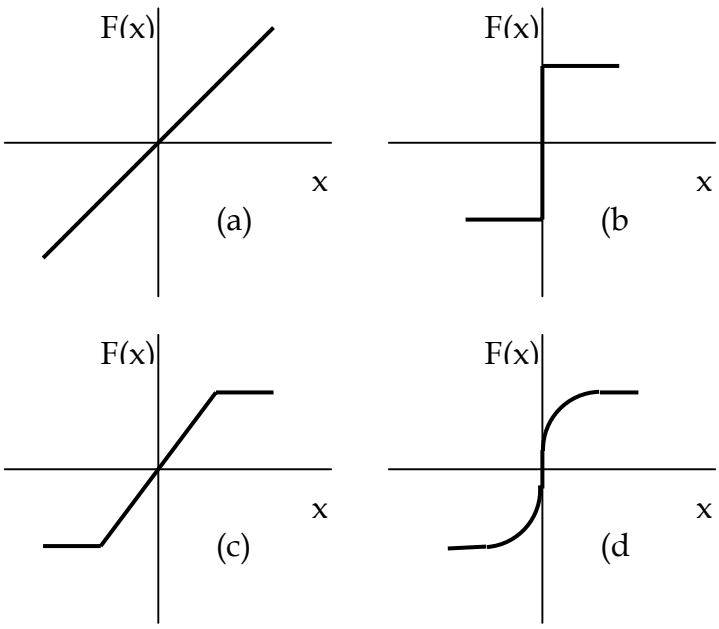


Figura 4 - Funções de Transferência [PON98]

Quanto ao número de camadas, pode-se ter:

- *Redes com uma única camada* são redes que contém um neurônio (nó) entre qualquer entrada e qualquer saída;
- *Redes de múltiplas camadas* são redes que contem mais de um neurônio entre alguma entrada e alguma saída da rede.

Quanto ao tipo de conexão de cada neurônio tem-se:

- *Feedforward*, ou acíclica à saída de um neurônio em qualquer camada não pode ser usada como entrada de outro neurônio em camadas de índice menor;
- *Feedback*, ou cíclica à saída de um neurônio em qualquer camada é usada como entrada de outro neurônio em camadas de nível menor.

E para concluir, as redes neurais artificiais podem ser classificadas quanto suas conectividades:

- *Rede fracamente conectada*, ilustrada na Figura 5;
- *Rede completamente conectada*, ilustrada na Figura 5.

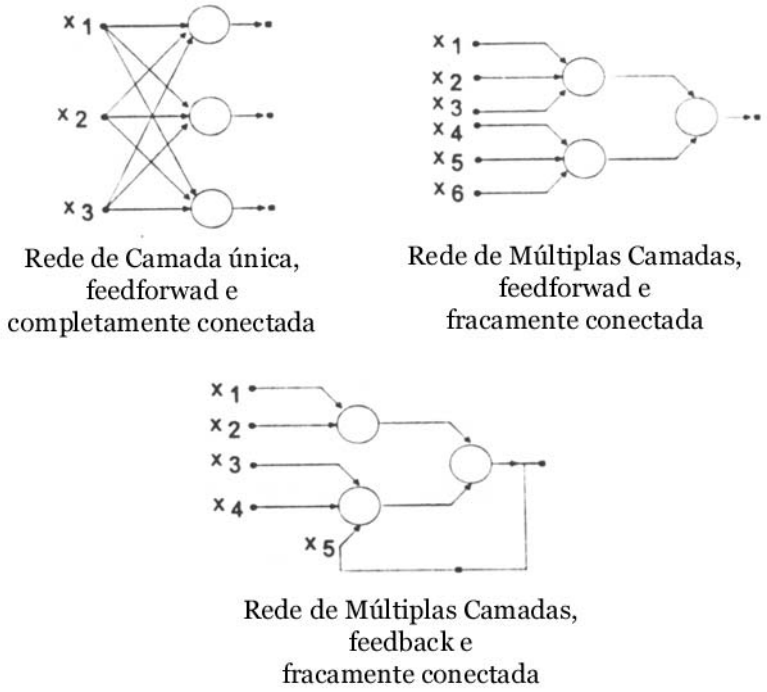


Figura 5 - Arquitetura Completa de Redes Neurais Artificiais [PON98]

Capítulo 3 - Sistema de Navegação

Navegação é a ciência ou tecnologia de encontrar a posição, o curso e a distância percorrida por um veículo qualquer. Com isso pode-se resumir o sistema de navegação em três questões: Onde estou? Para onde vou? Como eu irei até lá? Como muitos destes problemas são comuns à todas as entidades de movimentação, ou seja, homens com ou sem veículos, animais e veículos autônomos, e tendo como base que tanto homens quanto animais despendem uma grande parte de sua infância para aprender o básico sobre navegação, existem diversas formas envolvendo a solução desses problemas [BOR96; MIT2001].

No sistema de navegação, a informação é a principal chave para a tomada de decisões, pois motoristas bem informados podem tomar decisões mais eficientes.

Para a utilização de um veículo para navegação há três conceitos principais envolvidos [MIT2001]:

- *Motorista*: pode ser representado por um homem ou um módulo de decisões artificiais. Para dirigir um veículo é necessária a execução de cinco tipos de tarefas distintas como: *planejar*, *perceber*, *analisar*, *tomar decisões* e *controlar*. Algumas destas tarefas, como o planejamento, podem ser feitas antes do inicio da movimentação ("desligado" em termos de computação), porém muitas destas tarefas necessitam de uma resolução em tempo real, ou seja, enquanto o veículo se move. Percebe-se que quanto maior é a velocidade atribuída ao veículo, menor é a capacidade humana de percepção e menor o tempo para a realização das tarefas. Contudo em ambientes muito complexos (como os ambientes que possuem muitos veículos e/ou pedestres) consegue-se sobrecarregar a capacidade humana de tomar decisões, sendo que o resultado disto é um número alto de acidentes e mortes no trânsito;

- *Veículos*: equipamentos utilizados para navegar. Ele aumenta a capacidade de movimentação humana. Um veículo pode mover-se através de um motor, o qual recebe comandos vindos das atitudes tomadas pelo motorista;
- *Ambiente*: Tudo que for externo ao motorista e ao veículo, o qual pode influênciar o motorista durante a sua tomada de decisão. Particularmente as principais características do ambiente são as que contêm outras entidades de tráfego (outros veículos ou pessoas) e infra-estrutura de tráfego (objetos os quais auxiliarão o motorista, como placas, sinais, etc.).

No processo de navegação a maior parte do tempo despendido é durante a movimentação, onde o motorista tem que resolver as tarefas de navegação, sendo estas tarefas, procedimentos que tomam decisões, onde pelo menos um dos parâmetros (entrada ou saída) possui propriedades espaciais [MIT2001]. Existe ainda um grande número de tarefas em que algumas podem ser executadas independentemente, enquanto outras precisam ser sincronizadas. Dentre elas, tem-se as tarefas reativas (como parar bruscamente na frente de um objeto não esperado) que são simplesmente reações para a percepção sem qualquer planejamento e com ou sem modelo de ambiente, e as tarefas intermediárias que requerem mais detalhes sobre o ambiente ou sobre sua posição neste ambiente, para execução de funções ou comandos mais abstratos (como dirija pela rua, vire à direita, etc.) [MIT2001].

Para que a navegação funcione corretamente é necessária à interação de todas as tarefas e para isso é necessário informar a(s) posição(ões) corrente(s) referente a sua localização global para as tarefas de níveis mais altos. Após a tarefa intermediária encontram-se as tarefas de planejamento, as quais se baseiam em um modelo global de dados para planejar futuros passos de navegação, obtendo uma prévia sobre o objetivo a ser atingido [MIT2001].

A navegação é obviamente muito complexa para ser automatizada utilizando programas convencionais de computação. Por isso, hoje muitas pesquisas estão sendo direcionadas para o uso de técnicas de inteligência artificial na tentativa de solucionar alguns dos principais aspectos do processo de navegação [MIT2001].

O principal emprego da pesquisa realizada na área de inteligência artificial visando o sistema de navegação é na área de robôs móveis, pois para que os robôs sejam móveis eles necessitam explorar, perceber, mapear, navegar e realizar tarefas sobre o(s) ambiente(s) o(s) qual(is) ele se encontra [KOR93].

3.1. O que é preciso para navegar?

Mesmo a navegação sendo uma técnica antiga ela ainda é muito usada na nossa vida diária. Dentro deste século foram testemunhados dois caminhos nos quais os sistemas de navegação se desenvolveram. Eles são: o sistema de navegação inercial (utiliza sonares, infravermelho, etc.) e sistema de navegação via rádio (utiliza principalmente o Global Position System - GPS) [THRUN96; MIT2001; KOR99].

A forma de posicionamento dentro do sistema de navegação é ramificada em dois segmentos de sensores [THRUN96; MIT2001; KOR99]:

- *Posição Absoluta*: informa a localização global através de um sistema de coordenas absolutas (como receptor de GPS que fornece a posição tridimensional de um objeto na terra);
- *Posição Relativa*: informa a posição como sendo a diferença de distância entre o robô e um objeto (como o sensor de varredura a laser que fornece a distância entre um dispositivo e um objeto selecionado) ou de uma prévia posição do dispositivo (como o odômetro fornece distância percorrida por um veículo desde a última vez que o odômetro foi iniciado).

As pesquisas decorrentes dos sensores de navegação, acima citados, estão sendo vinculadas à execução de tarefas de navegação de alto nível, fornecendo assim as informações necessárias para o processo de planejamento, como por exemplo, posição, distância, etc. Porém, no conjunto de sensores descritos, sabe-se que indiferentemente do tipo de sensor de navegação, ele apresentará informações redundantes e com muitas distorções (ruídos), pois cada sensor possui limitações devido a suas características físicas apresentadas pelos materiais que os compõem. Devido a essa problemática os dados do mundo real têm que ser extraídos com a máxima fidelidade possível, pois se houver um pequeno ruído que não se consiga filtrar, toda à parte de gerenciamento de localidade estará imprecisa provocando assim a perca do foco central do robô. [THRUN96; MIT2001; KOR99].

A navegação, além da extração de dados do mundo real com uma boa fidelidade, necessita de [THRUN96]:

- *Mapa de localização*: utilizando as informações vindas dos sensores será construído um mapa que conterá o posicionamento do robô;
- *Planejador*: utilizando as informações do mapa será construído o plano de navegação, que irá traçar a rota a ser percorrida;
- *Navegador*: utilizará as informações do plano para navegar e cumprir o objetivo esperado.

Na Figura 6 é visualizado um diagrama do funcionamento de um sistema de navegação. Segundo o diagrama, todas as decisões são baseadas nas informações dos sensores que auxilia os demais módulos a realizarem suas tarefas. Isto quer dizer que se houver informações imprecisas o sistema terá problemas para navegar.

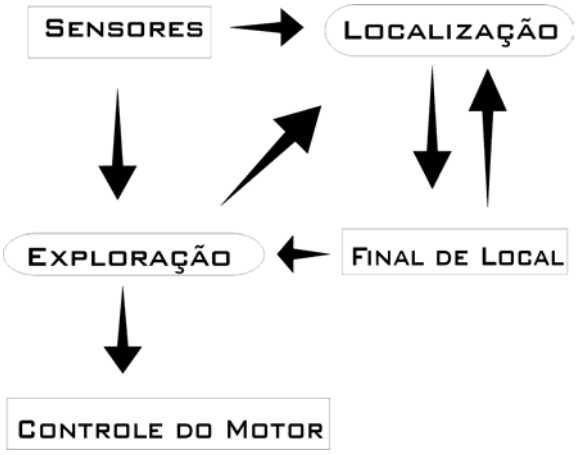


Figura 6 - Diagrama Genérico de Navegação[DUC99]

3.1.1. Navegação em Robôs Móveis

Como todo ser humano ou animal que navega, o sistema de navegação de robôs móveis necessita de um modelo de aprendizagem de direção com suporte para navegação, pois o sistema precisa saber onde se encontram os objetos para poder prever ou até mesmo planejar suas ações futuras. Este modelo de aprendizagem de direção terá como base os dados vindos dos sensores de navegação inercial, que contém as informações sobre o que há no mundo real, auxiliando no processo de navegação. Após o processo de aprendizagem do ambiente é necessária a navegação, onde navegar é um processo regular com um grande número de padrões que se repetem freqüentemente. Porém, é preciso detectar as diferenças sobre o contexto corrente de movimentação e o aprendizado de padrões, para que não haja a possibilidade da ação a ser tomado não ser a ação correta para a navegação [BOR96].

O desenvolvimento de um assistente de navegação inteligente contém [BOR96, MIT2001, PON98] :

- *Método de reconhecimento de eventos*: os dados dos sensores e suas traduções serão tratados como eventos em forma simbólica. A pesquisa terá inicio com método de reconhecimento híbrido, simbólico / não simbólico.

- *Método de aprendizado de padrões*: Coletará dados relevantes de um ou mais sensores, reconhecendo padrões e prevendo eventos futuros. Dentre as técnicas vistas até agora, tem-se várias técnicas descritas como método de aprendizado [PON98]:
    - *Técnica de aprendizado não supervisionado*, o qual assume as responsabilidades da solução da tarefa;
    - *Técnicas de medição de aprendizado*, onde é atribuído um valor para avaliar o sucesso do aprendizado. Esta técnica é utilizada para escolher qual o melhor paradigma de aprendizado.
- *Métodos de gerar predições* e avaliar as situações a serem desenvolvidas.

Para a criação de um sistema de navegação para robôs móveis é necessária uma estrutura que capture os dados e depois os transforme em informações para que haja o planejamento e a navegação dentro dos ambientes, ilustrada na Figura 7 [MIT2001].

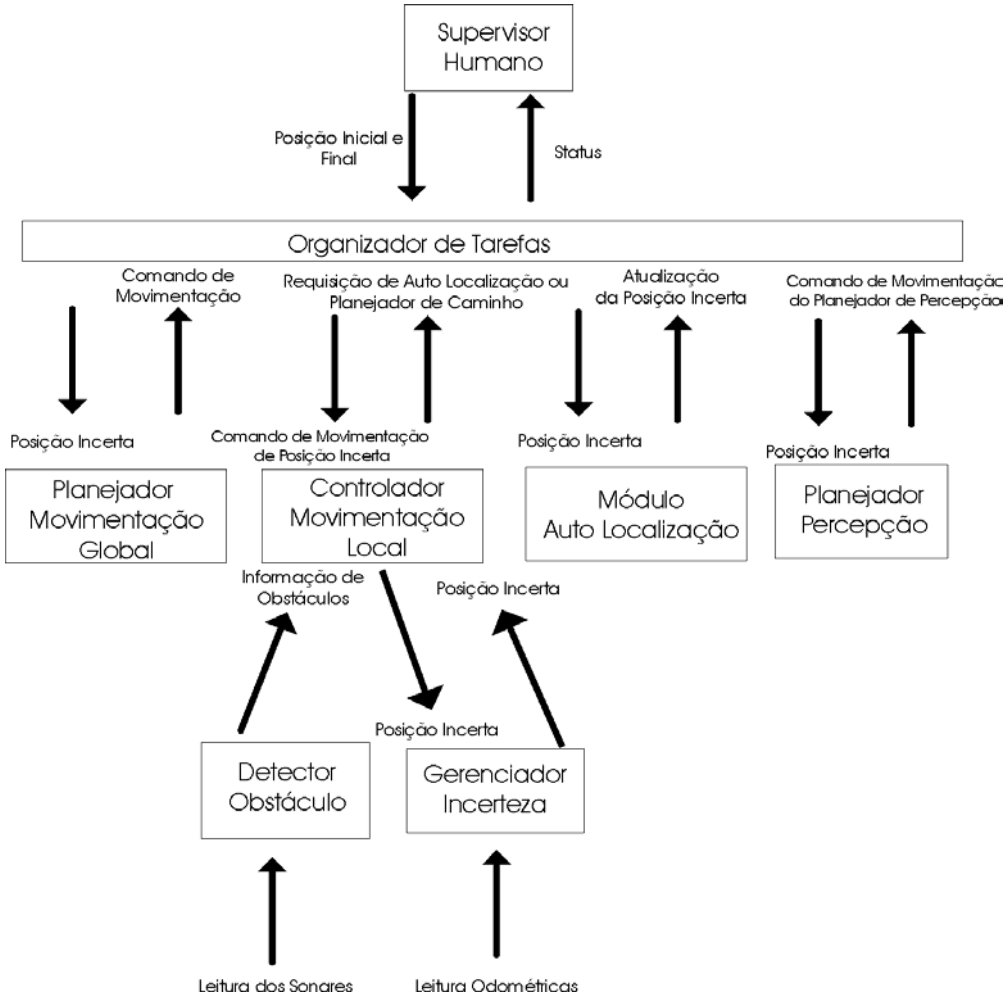


Figura 7 - Gráfico generalizado de Sistema de Navegação [MIT2001]

3.1.2. Problemática da navegação

Os problemas enfrentados no desenvolvimento de robôs móveis geralmente estão relacionados na base da criação do sistema de navegação, pois os robôs móveis possuem a necessidade de interagir com os objetos físicos e entidades do ambiente. A plataforma deve ser capaz de navegar a partir de uma posição conhecida e locomover-se para uma nova localização desejada, evitando colisões com objetos fixos e móveis durante sua rota [ARA2000].

Um dos principais problemas descritos sobre o sistema de navegação está relacionado com a filtragem dos dados referentes ao mundo real, os quais são recebidos pelos sensores externos. Esses dados são derivados de ambientes dinâmicos, onde estes ambientes sofrem varias mudanças no decorrer do tempo [DUC98, DUC99, DUC99A].

O segundo ponto a ser analisado é a transformação destes dados em informações úteis para que o robô possa tomar decisões. Estas informações serão agrupadas em mapas locais que classificarão o mundo real. Com estes mapas o robô poderá analisar qual a melhor forma para se locomover dentro deste ambiente [DUC98, DUC99, DUC99A].

Contudo estes mapas possuem um problema, que é a sua precisão dentro do contexto global. Para que o robô saiba que seu mapa local está correto, é necessário fazer um ajuste através da correlação da posição absoluta com a posição relativa, informando qual resultará na posição correta dentro do ambiente. Em vista disto, segundo [THRUN96], se estes problemas forem analisados individualmente, é possível desenvolver uma solução mais eficiente e precisa, tendo em vista que apesar de sua interdependência eles tratam de problemas distintos.

3.1.3. Exemplos de Sistema de Navegação

Atualmente existem várias formas de desenvolver o Sistema de Navegação de ambiente(s), sendo que o sistema é dividido em mapeamento, planejamento e navegação (descrito no item 3.1). Dentro destas subdivisões, elas podem ser implementadas de diversas maneiras, pois não se tem um paradigma padrão para nenhuma destas tarefas [MIT2001].

A seguir alguns exemplos de sistema de navegação e aprendizado de ambientes:

- *Sistema ALVINN*: que controla automóveis em estradas e ruas sem a problemática de velocidade e distância. Os dados de seu sistema vêm de câmeras de vídeo que projetam esses dados como se fosse uma retina. Ao serem transformados em informações, esses dados passarão por um software de inteligência artificial que mostra qual a direção a ser tomada [JOC96];

- *Sistema TESEO*: utiliza uma rede neural baseada em um paradigma de aprendizado por reforço [PON98] para resolver tarefas de navegação de alto nível. O sistema requer um mapa global do ambiente feito com o resultado de landmarks. A entrada da rede neural vem da leitura de vários infravermelhos e sonares bem com as distâncias sobre a localização atual e a localização do destino do robô. Um simples nó de saída controla diretamente a direção do robô e indiretamente às ordens do motor, corrigindo os problemas de navegação através de "tentativa e erro" [MIL95];
- *Sistema FINALE*: é um robô capaz de navegar autonomamente através de corredores a uma velocidade de 8m/min usando visão para autolocalização e sonares como detectores para evitar colisão. A atuação do robô não é debilitada pela presença da movimentação de objetos e de objetos estáticos nos corredores. Este robô simplesmente trata tudo que não é base para seu modelo geométrico como uma coleção de imagens desordenadas. Ele utiliza câmeras para se autolocalizar, como ilustrado na Figura 8 [RVL2001].

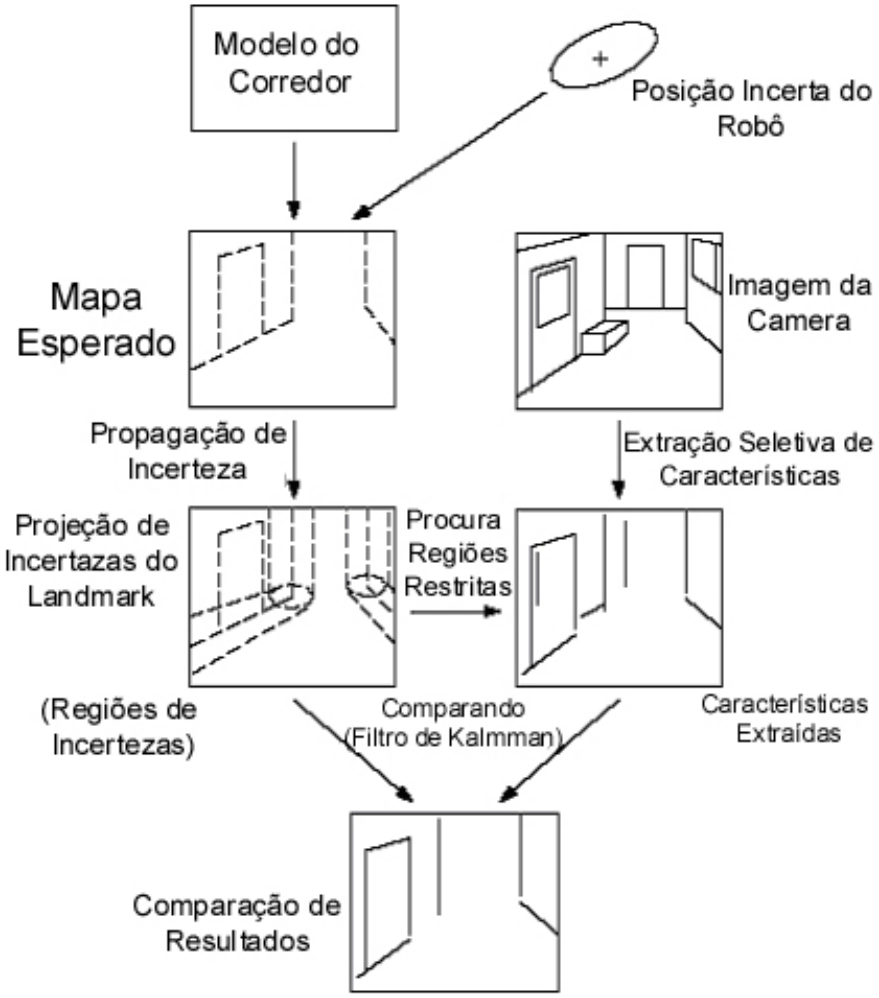


Figura 8 - Estrutura de Visão do FINALE [RVL2001]

Estes exemplos foram citados pelo fato de que eles conseguem mostrar as diversas utilizações do sistema de navegação, dando destaque à navegação de veículos e de robôs móveis.

3.2. Construindo Mapas em Sistemas de Navegação de Robôs Móveis

No conjunto de navegação de robôs móveis, os mapas são essenciais para o controle de robôs em ambientes não estruturados, sendo que ele precisa se localizar em relação ao ambiente em que se encontra, planejar o melhor caminho e interagir com o esse ambiente [DUC2000]. Sem essa habilidade de identificar lugares com confiança, o robô móvel inevitavelmente estaria perdido, e então ele passaria por cima dos objetos ou até mesmo danificaria sua estrutura caso não fosse tomada a decisão correta[NEH99].

Dentro dos tipos mais consagrados de mapeamento tem-se:

- *Mapas baseados em grids (matrizes)*: São mapas construídos em cima de matrizes de células, aonde cada célula contém as medidas de certeza e coordenadas cartesianas correspondentes às regiões ocupadas por objetos, descrita na Figura 9.
- *Mapas topológicos*: O lugar reconhecido é representado com um grafo de lugares conectados que não possuem objetos. Neste mapeamento não é preciso haver um prévio conhecimento das coordenadas cartesianas, ilustrado na Figura 10.

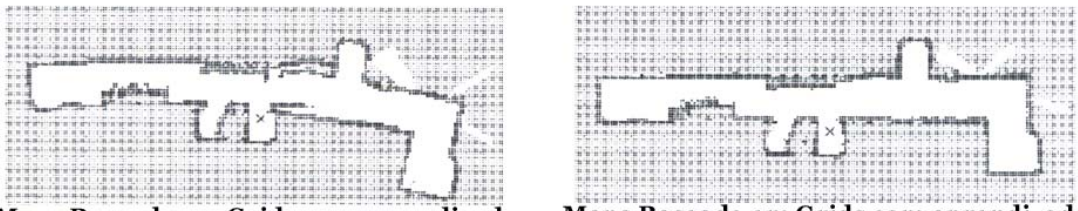


Figura 9 - Mapa baseado em Grids[DUC99,DUC99A,DUC2000]

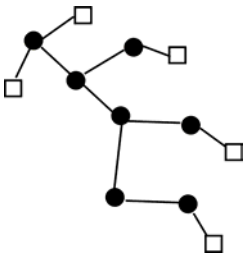


Figura 10 - Mapa Topológico[DUC99,DUC99A,DUC2000]

Estas duas formas de mapear, juntamente com qualquer outra forma de mapear contêm seus problemas e vantagens, como no caso do mapeamento baseado em grids, que possui uma interpretação geométrica detalhada do ambiente, porém o seu custo computacional é muito grande. Já o caso do mapeamento topológico, possui um baixo custo computacional, só que infelizmente ele não apresenta uma interpretação geométrica do ambiente [DUC2000].

3.2.1. Mapas Topológicos

Em relação aos mapas baseados em grids, os mapas topológicos possuem uma descrição mais abstrata do ambiente em que se está interagindo. Estes mapas são construídos através de dados com baixo nível de abstração, que são coletados pelos sensores que se encontram posicionados no robô [ULR2000].

Os mapas topológicos são baseados nos dados referentes às características do mundo real que estão sendo informadas por uma posição absoluta (lugar onde se encontra o robô) [DUC99; DUC99A; ULR2000].

Para se construir um mapa topológico é preciso:

- Pesquisar a leituras dos sensores do robô, pois não existe nenhuma prévia do mapa a ser visitado, sendo que a posição em que o robô é ligado, é identificado como posição inicial;

- Explorar o mapa, que consiste na tentativa contínua de expandir o território já representado pelo robô na forma de um grafo. Este trabalho utiliza uma rede neural artificial para adicionar um novo lugar "previsto" no mapa;

Este tipo de mapa contém dois diferentes tipos de lugares:

- *Os lugares previstos* são lugares que se presume existir, porém ainda não foram visitados pelo robô;
- *Os lugares confirmados* são lugares que o robô visitou atualmente.

O movimento subseqüente feito pelo robô é usado para verificar se a região (lugar) "prevista" atualmente existe ou não. Das regiões (lugares) iniciais, o robô adiciona o primeiro conjunto de lugares "previstos" no mapa, e então tenta navegar pelos lugares "previstos" próximos a ele. Se o robô estiver habilitado a mover-se fisicamente para a nova localização sem encontrar nenhum obstáculo, o lugar "previsto" é então transformado em lugar "confirmado", caso contrário ele é excluído do mapa [DUC98, DUC99, DUC99A, DUC2000].

A todo o momento outro lugar "confirmado" é adicionado no mapa, e a rede neural artificial é usada novamente para prever novos lugares dentro do ambiente a ser explorado. Este processo é repetido até que todos os lugares "previstos" no mapa tenham sido transformados em "confirmados" ou excluídos do grafo.

Para a implementação da estratégia de exploração baseada em mapas topológicos, os mecanismos necessários a serem analisados são [AYM98, DUC98, DUC99, DUC99A, DUC2000]:

- *Reconhecimento de localização (autolocalização)*. Assume-se que o robô tem a habilidade de se localizar dentro de um mapa. O algoritmo de autolocalização é capaz de determinar a posição mais provável ocupada pelo robô, e, além disso, pode mostrar o maior deslocamento do robô no interior de um possível lugar;

- *Detecção de espaço livre*. Para que um lugar "previsto" seja incluído no mapa topológico, o robô tem que ter a habilidade de determinar sua orientação. Em adição, alguns mecanismos são requeridos para a adição de novos lugares "previstos" (como detectar áreas inexploradas em uma particular direção). Uma rede neural foi treinada para aprender o conceito de "espaço livre", a qual detecta se o espaço que está combinando as informações, das quais apresentam ruídos, obtidos das diversas leituras feitas pelos sensores, possuem objetos ou está livre para a navegação;
- *Procurando um caminho*. Após a marcação dos lugares "confirmados" e lugares "previstos", é necessário encontrar um caminho para que o robô navegue dentro do ambiente. Para isso, utiliza-se o algoritmo de Dijkstra para encontrar caminhos a partir do grafo criado;
- *Cálculo de posição final*. Para determinar um novo lugar "confirmado" adicionado no mapa, uma estratégia de cálculo de posição final é utilizada. Se o robô conseguiu atravessar uma distância limite pré-estabelecida (1m) em sua próxima posição do mapa sem encontrar obstáculos, então um novo local "confirmado" é adicionado no mapa;
- *Gerenciando a consistência*. A posição final não pode ser usada para estimar o posicionamento global durante a construção do mapa, devido ao acúmulo de erros de movimentação causados pelas derrapagens das rodas. Então, alguns outros mecanismos são necessários para determinar às coordenadas globais dos lugares no mapa do robô, usando somente relações métricas locais no meio dos lugares (posições).

3.3. Pesquisas sobre Mapas Topológicos

Antes de discutir as pesquisas atuais na área de navegação será feito um breve resumo sobre os sistemas de navegação.

No início das pesquisas sobre os sistemas de navegação de robôs móveis, eram necessárias formas de controle de posições as quais identificavam a rota percorrida e a posição inicial de partida dos sistemas robóticos.

Para que esse controle fosse realizado era necessária a construção de mapas para a armazenagem dessas posições. Este mapa auxilia o sistema de navegação, pois uma vez guardado o percurso do robô, o sistema de navegação consegue identificar a melhor rota para a sua posterior navegação. As primeiras e mais utilizadas metodologias de desenvolvimento de mapas são os mapas topológicos e mapas baseados em grids, as quais estão descritas no item 3.2.

Como as pesquisas não cessaram, consegui-se com a evolução dos sistemas de navegação desenvolver uma vasta gama de variações desses sistemas, sendo alguns deles vindos de métodos híbridos.

No conjunto de sistemas híbridos, tem-se como exemplo, o sistema de navegação que utiliza o mapa topológico como um sistema primário na navegação sendo que há um sistema auxiliar utilizando o mapa baseado em grids. O funcionamento desse sistema se comporta da seguinte forma: o mapa topológico contém as informações das possíveis rotas, com isso o mapa topológico auxilia a criação de um mapa global baseado em grids o qual tem a capacidade de armazenar informações mais precisas do ambiente. Este sistema auxilia na localização exata do robô em relação ao ambiente e uma melhor detecção de objetos presentes nesse ambiente [DUC2000].

Como visto anteriormente, as pesquisas atuais estão desenvolvendo mapas topológicos associados a diversas formas de validação desse tipo de mapa. Essa validação é feita através de varias formas, sendo elas muitas vezes como meio de verificação de objetos encontrados no ambiente e validação de posicionamento local e global do robô.

Um dos projetos que explora o fundamento acima é citado em [VAL2001] o qual utiliza um robô NOMAD do SUPER SCOUT II que possui acoplado em seu sistema um laser scanner da SICK que tem por objetivo fazer varreduras de objetos e suas distâncias com uma abertura de 180^{o}.

Esse sistema de navegação constrói o mapa topológico, contudo esse mapeamento não possui uma boa odometria e para melhorar o aproveitamento do sistema de navegação o laser scanner auxiliará na questão do posicionamento local e global corrigindo a odometria e auxiliando na identificação de corredores e portas das salas. Isso é feito através da comparação das leituras dos sensores, ou seja, o algoritmo de navegação captura as informações de todos os sensores do robô, neste caso o laser scanner, que possui uma alta fidelidade, valida estas leituras e cria um mapa topológico com uma maior fidelidade ao ambiente em que o sistema esta navegando.

Inclusive neste sistema de navegação, a navegação é dividida em duas partes:

- *Navegação de baixo nível*: controlam os motores, rodas, encoders e realiza a parametrização;
- *Navegação de alto nível*: é construído e utilizado o mapa topológico;

Já no projeto de [KÜN2001], o sistema de navegação é utilizado em ambientes estáticos, tendo como principal base à utilização de máquinas de estado como mapas topológicos para o sistema de navegação reconhecer as características da região que o sistema de navegação esta percorrendo, ou seja, para cada região existente no mapa topológico existe uma característica peculiar que identifica esta região. Isto auxilia o sistema na tomada decisão de qual o melhor caminho a ser percorrido para alcançar um destino.

Em uma terceira fase há o desenvolvimento híbrido de sistemas, onde se tem a associação do reconhecimento de imagens com a construção dos mapas topológicos [GAS2000]. Esse sistema tem como base à navegação interna do ambiente utilizando duas formas de navegação:

- *Navegação Topológica*. Sistema utilizado para a travessia de longas distâncias sem necessitar da posição exata do robô. Porém o mapa utilizado (Topológico) deve armazenar algumas características peculiares da região que foi adicionada no mapa;
- *Navegação baseada em caminhos pré-definidos visualmente* é utilizada localmente, pois sua navegação é muito precisa e isto auxilia na mudança de ambientes, como por exemplo, travessia de portas. O robô é controlado para seguir um caminho apurado pelas características visuais acrescidas ao mapa, descritas na Figura 11.

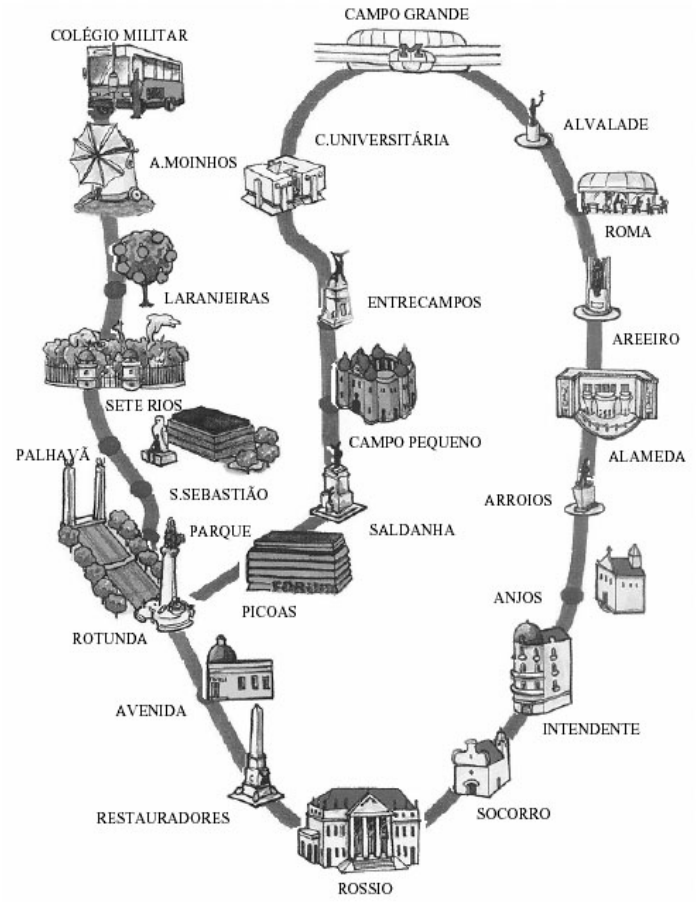


Figura 11 - Mapa baseado na Visão Omni-direcional [GAS2000]

Nesse sistema de navegação, um modelo complementa o outro, isto quer dizer que, enquanto a navegação topológica possui apenas as rotas e suas características, a navegação baseada na visualização consegue melhorar a topológica devido ao fato de capturar informações visuais do ambiente.

Existem várias outras aplicações para o sistema de mapeamento topológico, contudo têm-se apenas as aplicações com maior proximidade ao assunto deste estudo.

Capítulo 4 - Sistemas Reconfiguráveis

Ao grande avanço das mais diversas tecnologias no mundo de hoje, os circuitos digitais tem sofrido uma grande evolução, sendo que essas mudanças tiveram um fator muito importante e radical em todo o processo de projeto de hardware que influência diretamente a criação de novas soluções para as mais diversas áreas existentes [ARA2000, ARA2000B].

No conceito de componentes digitais houve uma revolução que foi evolução dos transistores individuais para circuitos integrados VLSI (very large scale integration). Acompanhando esta evolução, veio à utilização de ferramentas EDA (eletronic design automation) para simplificar e acelerar todo o ciclo de projeto, deixando com isso, de haver a necessidade de desenhar portas lógicas individuais e planejar todas suas interconexões [ARA2000, ARA2000B].

O projeto de desenvolvimento de ferramentas para auxiliar a elaboração de circuitos tem adquirido grande importância, pois todas as mudanças que ocorreram na tecnologia de circuitos digitais exigem uma prototipação cada vez mais rápida, pois o ciclo de vida dos produtos modernos está tornando-se cada vez mais curto em relação ao tempo necessário para o projeto e desenvolvimento dos mesmos [ARA2000B].

É importante ressaltar que os Circuitos integrados (CIs) digitais podem ser construídos utilizando-se diversas tecnologias diferentes, a escolha da tecnologia adequada deve ser realizada com base no tipo de projeto que se pretende executar.

Observando as implementações de circuitos, podem-se agrupá-las em diversas categorias [ARA2000]:

- *CIs customizados ou ASICs (Application Specific Integrated Circuits)*: São circuitos que necessitam de um processo de fabricação especial, pois requerem máscaras (moldes) específicas para cada projeto, tendo com isso um tempo de desenvolvimento longo e os custos extremamente altos. Em aplicações que requerem um grande volume de produção, o alto custo,

tanto do projeto quanto dos testes, são amortizados;

- *MPGAs (Mask-Programmable Gate Arrays)*: Nesse tipo de implementação, o processo de fabricação é agilizado pelo uso de máscaras genéricas de módulos pré-projetados, mas ainda necessita de máscaras específicas para a interconexão dos módulos. O projeto é normalmente facilitado por uma biblioteca de células, proporcionando um tempo de desenvolvimento mais curto e custos mais baixos em relação aos CIs customizados;
- *Standard Cells*: Essa tecnologia se assemelha muito à das MPGAs; o projeto também é facilitado pelo uso de módulos pré-projetados, sendo os módulos (standard cells) geralmente salvos em bancos de dados para futuras utilizações. Os projetistas selecionam as células desejadas (nesses bancos de dados) para realizar seus projetos. Em comparação aos CIs customizados, os circuitos implementados em standard cells são menos eficientes em tamanho e desempenho, entretanto, seu custo de desenvolvimento é mais baixo;
- *PLDs (Programmable Logic Devices)*: Essa tecnologia possui como principal característica à capacidade de programação (configuração) pelo usuário, eliminando o processo de fabricação e facilitando assim as mudanças de projetos. Em comparação com outras tecnologias, os PLDs apresentam um ciclo de projeto muito curto e com baixo custo.

O mercado de PLDs encontra-se em plena expansão, de forma que atualmente existem diversos fabricantes e modelos de dispositivos. Uma das principais tarefas do projetista hoje é pesquisar e selecionar, dentre as opções disponíveis no mercado, qual a que melhor atende suas necessidades [ARA2000].

Já a tecnologia dos MPGAs motivou o projeto de dispositivos programáveis equivalentes, conhecidos como FPGAs (Field-Programmable Gate Arrays). A principal diferença é que no MPGA, a sua interconexão é feita durante o processo de fabricação, como em circuitos integrados, enquanto os FPGAs são programados via comutadores programáveis eletricamente, assim como nos PLDs [ARA2000].

A programação de um FPGA é realizada pelo próprio usuário final. Este novo tipo de tecnologia fez surgir à computação reconfigurável, pois o usuário cria uma solução para suas necessidades e programa esta solução em uma FPGA. Em termos básicos, a computação reconfigurável combina a velocidade do hardware com a flexibilidade do software.

A tecnologia da computação reconfigurável consiste na habilidade de se modificar o hardware da arquitetura para esse se adequar à aplicação [DEH; SBP; VIL97]. Essa reconfiguração do hardware pode ser realizada de duas maneiras:

- *Reconfiguração Estática*: onde a utilização do chip deve ser suspensa, até que o processo de reprogramação seja concluído;
- *Reconfiguração Dinâmica*: onde uma parte do chip pode continuar sendo utilizada durante a reprogramação. São reprogramadas as regiões do chip que não estiverem sendo usadas naquele determinado momento, e as regiões em uso se mantêm inalteradas.

Isso poderia ocorrer também de forma dinâmica, de tal forma que a Reconfiguração ocorra entre tarefas sendo executadas por aquela FPGA.

O hardware reconfigurável pode ser classificado em três categorias:

- *Hardware Puro* (hardware para aplicações específicas);
- *Coprocessadores* (chips específicos para determinadas tarefas);
- *Computadores* (plataformas de computação completamente reconfiguráveis).

Muitos trabalhos recentes têm sido realizados na construção de arquiteturas reconfiguráveis, destacando-se PRISC [SMI94], DISC [WIR95], GARP [HAU] [HAU97], RAW [TAY96], entre vários outros. Atualmente, a área de computação reconfigurável apresenta-se como uma tecnologia inovadora, pelas suas características de desenvolvimento e construção, tanto no campo industrial, como no campo acadêmico. São inúmeras as atuais aplicações de computação reconfigurável como, por exemplo: telefones celulares, controladores de dispositivos presentes em carros e aviões,

hardwares específicos de alto desempenho para as mais diversas aplicações, como por exemplo, sistemas robóticos, sistemas de aviação e etc.

A computação reconfigurável ainda é uma área em desenvolvimento, diferente de arquitetura de computadores, aonde os conceitos e mecanismos vêm sendo longamente testados e comprovados. Portanto, definir e validar esses tópicos em computação reconfigurável se torna um desafio para os pesquisadores atualmente, contudo há um grande interesse e planejamento para tornar esta tecnologia um padrão mundial, e para isso se vê grandes empresas utilizando essas tecnologia e aprovando-a em sua totalidade motivando assim a sua padronização [ARA2000].

4.1. Field Programmable Gate Array (F.P.G.A)

Nas últimas décadas a maioria dos hardwares bem como suas funções lógicas eram fixas e não podiam ser modificadas. Porém, com a tecnologia de FPGAs, descrita anteriormente, tanto as funções lógicas quanto os blocos lógicos podem ser alterados (incluindo também as conexões entre esses blocos), enviando sinais de configuração para os chips. Com isso os blocos lógicos de um FPGA podem ser re-escritos e re-programados repetidamente, muito depois do chip ter sido produzido em uma fábrica.

Esta tecnologia vem desenvolvendo um novo conceito para circuitos integrados, onde ela oferece uma nova opção com grande capacidade, velocidade e modularidade, para que os circuitos de hardware possam ser modificados em qualquer momento durante o seu uso.

A arquitetura básica de um FPGA consiste de um arranjo 2-D de blocos lógicos, onde a comunicação entre os blocos é realizada através de recursos de interconexão, sendo a borda externa desse arranjo blocos especiais capazes de realizar operações de entrada e saída (I/O). Uma arquitetura típica de um FPGA pode ser visualizada na Figura 12.

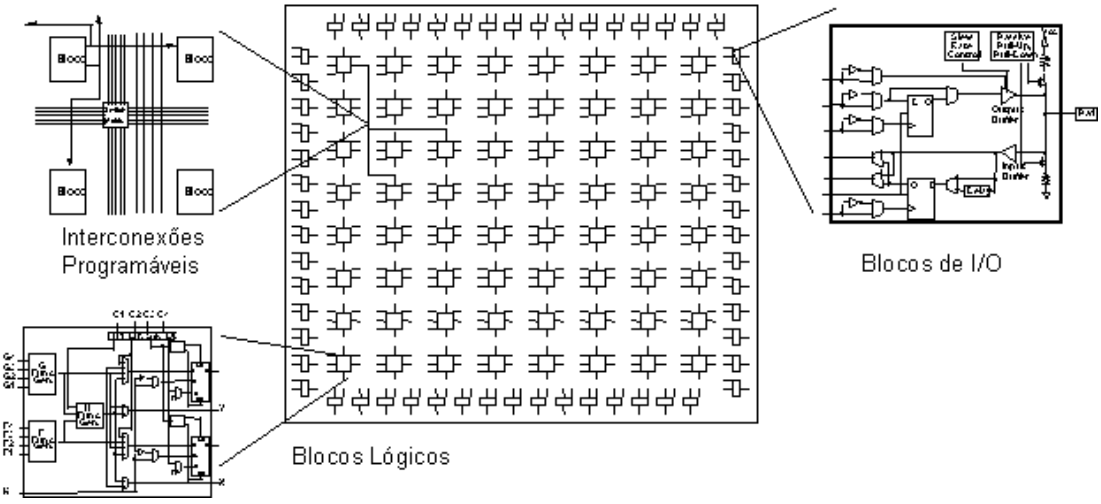


Figura 12 - Estrutura básica de um FPGA [DUC2000]

Os FPGAs combinam o desempenho dos gate arrays com a versatilidade dos PLDs, sendo responsáveis pelas principais mudanças no modo em que os circuitos digitais são projetados.

Os principais fabricantes de FPGAs são:

- *Altera*, a mais moderna família de FPGAs desenvolvida pela Altera é a família APEX. Os chips APEX apresentam um altíssimo nível de integração, o que permite que sistemas completos sejam implementados em um único dispositivo e possuem um acentuado desempenho, possibilitando assim, uma alta flexibilidade de projeto e eficiência para aplicações SOC (System on a Chip) de alto desempenho;
- *Xilinx*, a tradicional fabricante de FPGAs Xilinx atualmente trabalha com a família Virtex II. Essa família é especialmente desenvolvida para possibilitar um rápido desenvolvimento em duas das mais desafiadoras áreas da tecnologia: a comunicação de dados e o processamento de sinais digitais;
- *Actel*, a mais moderna família de FPGAs desenvolvida pela Actel é a família eX. Os chips eX são baseados na tecnologia antifuse e, têm como características básicas o alto desempenho, o baixo custo e consumo de energia.

Com a implantação dessa nova tecnologia, surgem novos termos e conceitos, como por exemplo, o termo reconfiguração dinâmica (auto-reconfiguração), que é a possibilidade de mudar, totalmente ou parcialmente, a funcionalidade de um sistema usando um mecanismo transparente, o qual o sistema não precisa parar a sua execução enquanto o FPGA é reconfigurado [MOR2001].

4.1.1. Reconfigurabilidade de FPGAs

Segundo sua reconfigurabilidade, os FPGAs podem ser classificados conforme mostra a Figura 13 [LYS93]. Todos os dispositivos FPGAs são por definição programáveis, ou seja, configurável pelo menos uma única vez, como qualquer PLD. Um pequeno subconjunto destes dispositivos pode ser reconfigurado várias vezes por uma operação que carrega completamente a configuração do dispositivo.

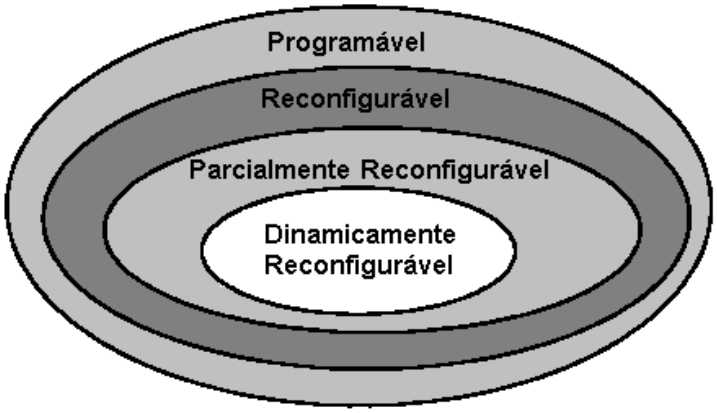


Figura 13 - Classificação de FPGAs de acordo com sua configurabilidade [ARA2000B].

Conforme a figura acima se tem:

- *Reconfiguração Parcial*: essa reconfiguração é caracterizada como parcial se um dispositivo aceita uma reconfiguração seletivamente, ou seja, uma parte do dispositivo é reconfigurado enquanto o resto permanece inativo, porém à parte que não será reconfigurada permanece intacta sem nenhuma modificação;

- *Reconfiguração Dinâmica*: os FPGAs são classificados como dinamicamente reconfiguráveis se seus circuitos internos de armazenamento podem ser atualizados seletivamente sem prejudicar o funcionamento da lógica restante que pode estar em operação (execução). Estes dispositivos podem assim ser reconfigurados seletivamente enquanto estiverem ativos.

Exemplos destes dispositivos dinamicamente reconfiguráveis são os da família APEX da Altera, XC6200 e Virtex da Xilinx.

Para um FPGA ser reconfigurável dinamicamente, implica-se que ele deve ser capaz de se reconfigurar parcialmente enquanto ativo (energizado e em operação). Em nível de sistema, um módulo que contenha múltiplos FPGAs pode ser classificado como reconfigurável dinamicamente se os componentes FPGAs são reconfigurados individualmente [WIR95].

Como os tempos de configuração não são desprezíveis, a habilidade de intercalar execução e reconfiguração, sem prejuízo do desempenho, é uma questão que ainda merece atenção e esforços de pesquisas [WIR95].

Um sistema de reconfiguração dinâmica inclui pelo menos uma área de reconfiguração onde blocos lógicos podem ser carregados em tempo de execução, como visto na Figura 14 que ilustra a reconfiguração dinâmica de um sistema composto de cinco circuitos ou tarefas, onde as tarefas de entrada e saída são permanentemente residentes no FPGA enquanto as três tarefas dinâmicas alternam-se sob o controle de um sinal de Swap [ARA2000].

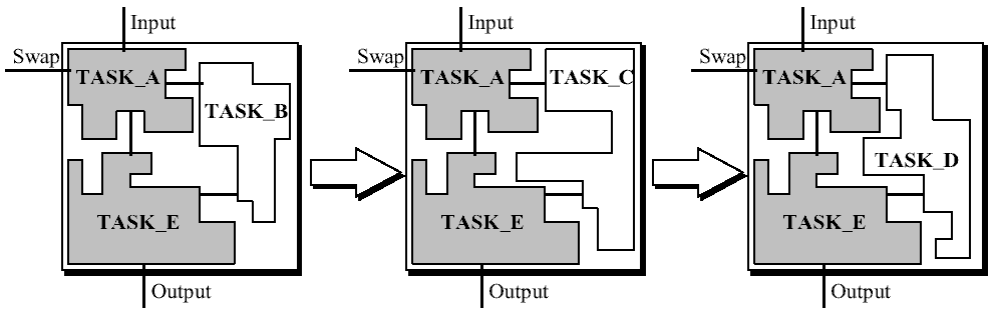


Figura 14 - Exemplo de Reconfiguração dinâmica [ARA2000B].

A capacidade de reconfiguração dinâmica permite o compartilhamento ao longo de tempo de tarefas diferentes, o que pode reduzir significativamente a área de silício exigida. Esta tecnologia torna possível o conceito de hardware ilimitado ou "Hardware Virtual" [HAU97B; CAR99].

A principal questão nesta abordagem é a implementação de um controlador capaz de manipular a reconfiguração de todas essas tarefas. Este controlador deve oferecer serviços como o carregamento e remoção de tarefas, escalonamento de tarefas e gerenciamento dos recursos.

4.2. System On Chip (SOC)

Impulsionados pelo avanço da eletrônica, e visando sempre em diminuir o tamanho dos circuitos, os projetistas conseguem colocar mais dispositivos eletrônicos no mesmo espaço de um chip, criando assim novas características e funcionalidades aos sistemas desenvolvidos atualmente [D&R2001].

Hoje, os circuitos integrados (CI) fabricados são compostos de milhões de dispositivos, comportando muitas vezes sistemas eletrônicos inteiros integrados em um único circuito, tornando-se assim um completo sistema no chip (system on chip) [TAN2001].

De fato, alguns grupos de CIs possuem chips que realizam operações complexas de diversos sistemas, como memória, unidade central de processamento, controle de entrada/saída e muito mais, em um simples projeto de SOC. No mundo real, placas mães completas podem ser contidas em um simples chip, como componentes eletrônicos como DVD e VCRs, que precisariam de vários chips, e agora necessita de somente um [TAN2001].

Segundo a lei de Moore que diz que a cada 18 ou 24 meses o número de componentes de um chip dobra. O crescimento da densidade esta habilitando novos desenvolvimentos no projeto de automação eletrônica unindo juntamente com as novas capacidades de fabricação de chips. Essa idéia tem sido amplamente adotada pela

grande maioria de fabricantes da área de hardware, principalmente pelo fato do excelente desempenho e versatilidade que a tecnologia FPGA tem se mostrado para implementar os sistemas SOC [RON2002]. Isso forçará muitas mudanças nas atitudes dos projetistas de CIs [RON2002].

4.3. Robôs Reconfiguráveis

Os primeiros sistemas robóticos tinham como objetivo realizar tarefas simples e repetitivas. Para que isso fosse possível era necessário que o ambiente de interação do robô fosse estruturado, ou seja, ambiente deveria possuir uma organização a qual não poderia mudar durante a interação do robô, pois se o ambiente fosse modificado o robô não conseguiria se localizar e tão menos se locomover [KEI98].

Com o passar dos anos houve a evolução e esses sistemas tomaram uma nova direção, que é a realização de tarefas interagindo com um ambiente que se modifica constantemente (ambiente dinâmico), tendo a capacidade de interagir, inclusive, entre diversos ambientes dinâmicos [KEI97; KEI98].

Devido a esses aspectos, o projeto de um robô com a habilidade de se adaptar ao ambiente e ainda realizar as funções requeridas, deverão apresentar muito mais do que uma simples arquitetura fixa, pois este sistema deverá se reajustar (adaptar) ao seu novo ambiente e a sua nova função conforme sua interação com este ambiente [KEI97; KEI98;NAB98].

Para que esse sistema funcione, ele deve se auto-reconfigurar, pois essa auto-reconfiguração consiste em compilar e executar programa(s), o qual forneça a união entre o software de controle de baixo nível e o software planejador de alto nível, para que o robô se auto-ajuste conforme a necessidade [KEI97; KEI98;NAB98].

A Reconfigurabilidade do robô consiste então em um conjunto de módulos idênticos que autonomamente e dinamicamente se reconfigura em uma variedade de formas a qual melhor se adapte aos novos requisitos, como a melhor forma de navegar em um tipo terreno especifico, a melhor forma de interagir com um ambiente, e a

realização da tarefa adequada. Sendo assim, teremos robôs mais extensíveis e versáteis com múltiplas modalidades de locomoção e manipulação [NAB98].

Um simples módulo autônomo de uma arquitetura pode agregar uma variedade de estruturas com módulos idênticos. Se o módulo é robusto e o protocolo de agregação esta provavelmente correta, o resultado final é um confiável robô, como por exemplo, o módulo de reconfigurabilidade que estrutura o robô para atravessar túneis, e depois o reconfigura novamente, com o objetivo de manipular objetos através de um braço [SBP2001].

Como descrito anteriormente este dispositivos possuem a capacidade de se reconfigurar e desta forma, estes dispositivos são muito úteis no desenvolvimento de robôs móveis reconfiguráveis, pois quando houver a necessidade de reconfigurar o robô para a realização de uma nova tarefa, o robô não perderá as suas configurações originais, mudando assim somente os módulos que não serão mais necessários [SBP2001].

Para que esses sistemas sejam desenvolvidos com base em FPGAs deve-se antes de mais nada analisar a forma de desenvolvimento de sistemas em FPGAs. Como visto anteriormente, existem várias empresas que desenvolvem FPGAs, cada uma com sua arquitetura e suas ferramentas de desenvolvimento. Cada ferramenta analisa e configura a FPGA da melhor forma possível, isto quer dizer que as aplicações são construídas em software e depois a ferramenta da FPGA integra esse software a FPGA, utilizando assim os componentes da FPGA [SBP2001].

4.4. Ambiente Excalibur

A ferramenta de desenvolvimento utilizada neste projeto é chamada Excalibur desenvolvida pela empresa Altera.

O ambiente Excalibur é uma ferramenta de desenvolvimento de SOPC (system-on-a-programmable-chip) que contém um processador embutido (embarcado) chamado NIOS. Este processador é um processador totalmente integrado com um softcore para

trabalhar com arquiteturas PLDs da Altera.

O kit de desenvolvimento Excalibur™ contém todas as ferramentas que os projetistas de hardware precisam para criar sistemas de alto desempenho em dispositivos lógicos programáveis. O kit provê o *soft-core* de um processador otimizado que pode ser imediatamente implementado, diminuindo o tempo de desenvolvimento de sistemas de hardware [ALT2002].

A ferramenta de desenvolvimento Excalibur contém os seguintes itens:

- Processador Nios RISC configurável;
- Compilador GNUPro® da Cygnus®, uma companhia Red Hat®;
- O Software de desenvolvimento Quartus™;
- Cabo ByteBlaster™;
- Placa de desenvolvimento equipada com o FPGA APEX™ EP20K200E.

4.4.1. O Processador Nios

O processador embutido Nios é um *soft-core* de um processador RISC configurável, desenvolvido especialmente para a arquitetura dos PLDs. Otimizado para uma área reduzida do PLD, este processador provê um desempenho de até 50MIPS. Inicialmente projetado para a família APEX, o Nios ocupa apenas 12% da área de um FPGA EP20K200E, permitindo ao projetista utilizar, de maneira conveniente, o restante da área disponível. O diagrama de blocos do processador Nios (configurado para 32 bits) é ilustrado na Figura 15.

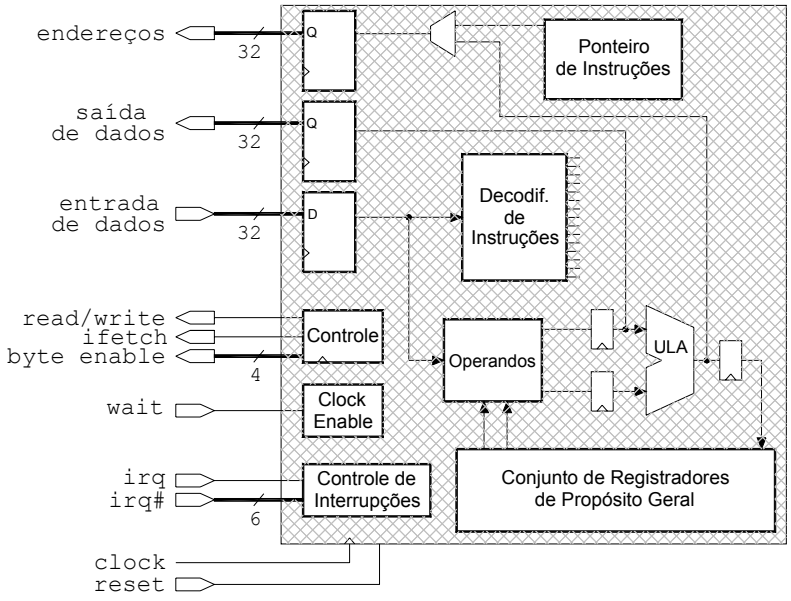


Figura 15 - Diagrama de Blocos do Processador Embutido Nios [ALT3W].

Características do processador embutido Nios:

- Conjunto de instruções de 16 bits;
- Barramento de dados de 16 ou 32 bits;
- Uma instrução por ciclo de *clock*;
- Suporte para memória *on-chip* ou *of-chip*;
- Desempenho de mais de 50 milhões de instruções por segundo (MIPS);
- Registrador para rápido manuseio de interrupções;
- Conjunto de 512 registradores de 32 bits;
- Registradores de acesso a periféricos.

Uma interface *MegaWizard* presente no ambiente de desenvolvimento Quartus, permite ao usuário especificar as conexões entre o processador Nios e o restante do sistema. Através desta interface, o usuário pode gerar um conjunto de periféricos e inseri-los ao *soft-core* final [ALT3W].

Vários periféricos são disponíveis para utilização com o Nios. Dentre eles, pode-se citar:

- Receptor / transmissor assíncrono (UART);
- Entrada / saída paralela (PIO);
- Temporizador;
- Controlador de disco IDE;
- Interface para memórias SRAM e FLASH;

O esquema utilizado na comunicação entre o processador embutido Nios e os periféricos definidos pelo usuário é ilustrado na Figura 16.

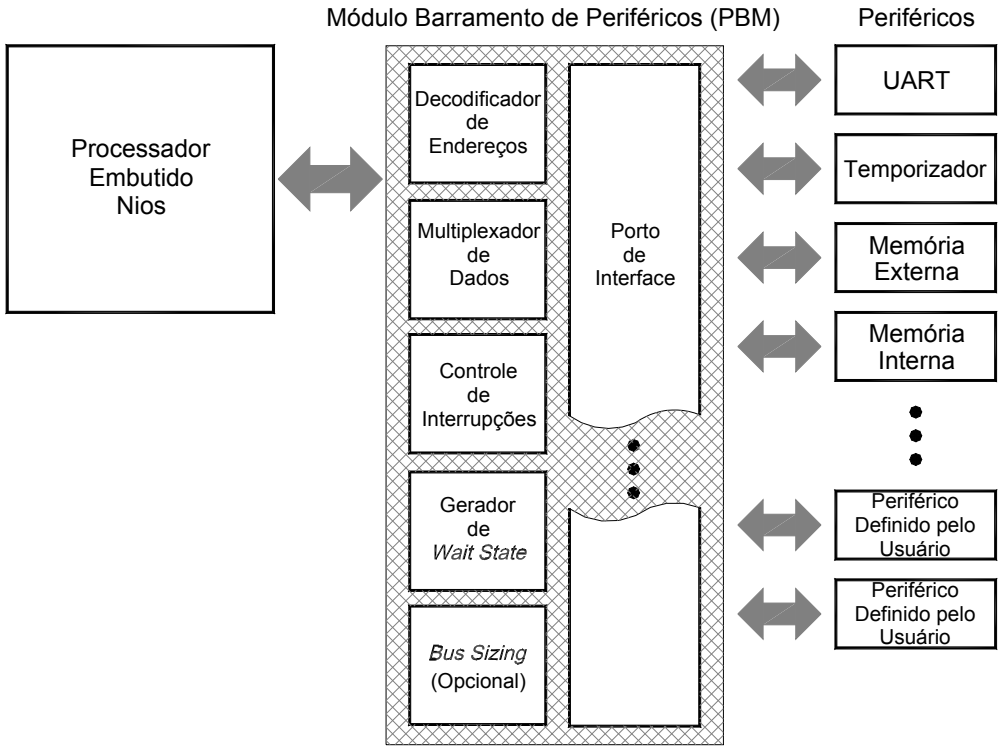


Figura 16 - Comunicação entre o Processador Nios e seus Periféricos [ALT3W].

A interface *MegaWizard* cria o Módulo de Barramento de Periféricos (PBM) de acordo com a configuração especificada. Características como endereço base, número de portos, largura de barramento de dados, *wait-states* e prioridade de IRQ são automaticamente customizadas.

A eficiência da utilização do processador Nios permite que múltiplas instâncias do dispositivo possam ser implementadas num mesmo FPGA, satisfazendo assim às necessidades de aplicações que requerem desempenho intensivo. A Figura 17 ilustra esta possibilidade.

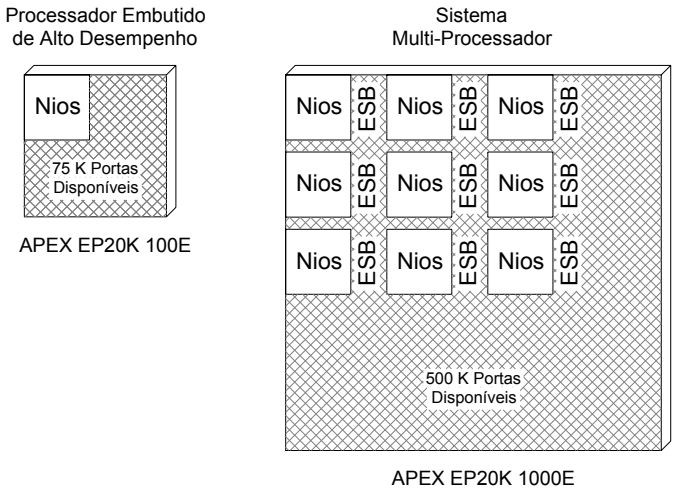


Figura 17 - Flexibilidade e Escalabilidade dos Processadores Embutidos Nios [ALT3W] .

4.4.2. Compilador GNUPro

O compilador e *debugger* GNUPro da Cygnus, uma companhia da Red Hat, é uma ferramenta padrão, usada por fabricantes de software de todo o mundo. Ele é uma ferramenta de desenvolvimento C/C++. Otimizado para o processador embutido Nios, o GNUPro proporciona um ambiente de desenvolvimento familiar aos engenheiros e projetistas, incluindo:

- Compilador otimizado C/C++;
- Assembler GNU;
- *Debugger* interno;
- Utilitários binários.

4.4.3. A Ferramenta de Desenvolvimento Quartus

O software de desenvolvimento Quartus permite aos projetistas, o processamento de milhões de portas lógicas, com vantagens nunca vistas antes nas ferramentas de desenvolvimento de PLDs. O software Quartus suporta soluções no nível de sistema com editoração de blocos, trabalho em grupo e um avançado suporte para megafunções. Além disso, um sistema de análise lógica embutido, permite aos usuários verificar a funcionalidade e a temporização do *chip*, observando os valores de sinais internos à velocidade de clock do sistema. O software Quartus é um ambiente completo para o desenvolvimento de projetos SOPC (*system-on-a-programmable-chip*) [ALT3W].

4.4.4. Placa de Desenvolvimento

O kit de desenvolvimento Excalibur inclui também uma placa de desenvolvimento onde os projetistas podem implementar seus sistemas sobre um *chip* programável [ALT2002]. O *lay-out* desta placa pode ser visualizado na Figura 18.

O *chip* reconfigurável que acompanha a placa de desenvolvimento, como pode ser visto na figura, é o APEX EP20K200E, com 526.000 portas lógicas e 106.496 bits de memória RAM.

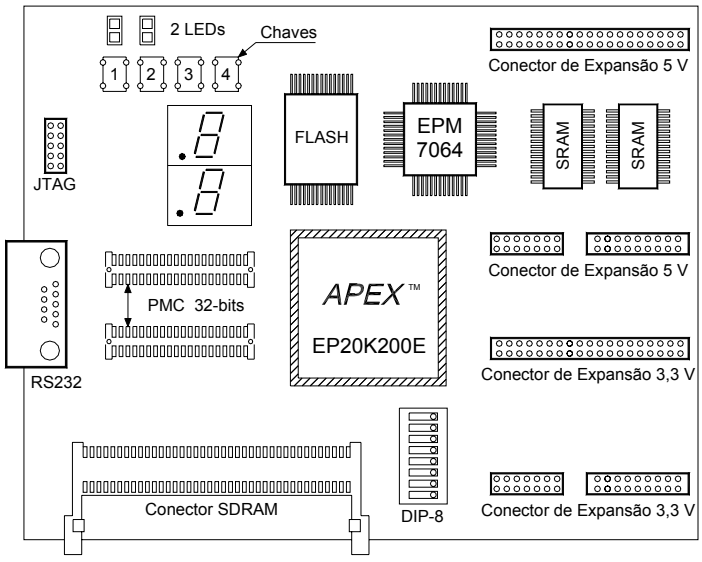


Figura 18 - Placa de Desenvolvimento Excalibur [ALT2002].

O conjunto de ferramentas agregadas ao Excalibur possibilita o desenvolvimento de aplicações e o próprio gerenciamento dos erros e das analises de desempenho. Além de prover desenvolvimento através de blocos lógicos (portas lógicas) e linguagem de programação C Ansi. Incluso ao modulo de desenvolvimento através da linguagem C Ansi se tem à simulação de operações aritméticas utilizando ponto flutuante e utilização de alocação dinâmica de variáveis sem interferir nos demais processos alocados no NIOS.

O kit de desenvolvimento que se encontra no Laboratório de Computação Reconfigurável (LCR), no qual esta sendo desenvolvido este projeto é o kit Excalibur com uma FPGA da Altera da família APEXII 20KE [ALT3W]. Atualmente o LCR esta adquirindo um conjunto de ferramentas de desenvolvimento Excalibur / ARM que possui uma FPGA do tipo Stratix.

4.5. Kit Excalibur / ARM

A Altera, empresa que desenvolve kits de System On a Programmable Chip, o qual faz parte o Excalibur, desenvolveu um novo kit de SOPC o qual possui as seguintes características [ALT2001,ALT3W]:

- Processador ARM922T com velocidade de 200MHz;
- De 4.160 a 38.400 Elementos Lógicos (LEs);
- Periféricos de teclado;
- Portas duplas e simples de memória externa;
- Interfaces para rede do tipo Ethernet com conexão externa do tipo RJ45;
- Suporta programação externa através da linguagem C/C++;
- Possui desenvolvimento do Hardware através de um Kit de Software de Desenvolvimento, o qual faz parte o Quartus II utilizado neste projeto.

Este novo kit o qual foi apresentado acima, é conhecido como Excalibur com processador ARM e FPGA EPXA10. Este novo kit foi criado para auxiliar nas diversas aplicações existentes hoje, como por exemplo, TVs digitais, switchs de redes do tipo frame-relay, redes DSL e outras aplicações de redes, aplicações real-time, soluções linux. Com esse kit é possível customizar e integrar diversos tipos de aplicações com um grande volume de produção.

O processador ARM utilizado neste kit é baseado em um processador de 32 bits com 8 Kbytes para cada instrução e para dados de cachê, um gerenciador de memória (MMU), e um módulo de capacidade embutido (ETM). Em adição a esse processador, a família Excalibur com o processador ARM integra portas simples e duplas de SRAM com 256 Kbytes de porta simples e 128 Kbytes de portas duplas de SRAM avaliada na FPGA EPXA10. Este kit ainda inclui uma interface para memória externa e uma interface de barramento de expansão (EBI) [ALT3W].

Capítulo 5 - Sistema Saphira

O sistema Saphira é uma arquitetura desenvolvida para controle de robôs móveis, sendo também, uma ferramenta para desenvolvimento de ambientes para vários tipos de plataforma de robôs. Ele foi originalmente desenvolvido para as pesquisas do robô Flakey 2 do SRI Internacional que é um instituto sem fins lucrativos, tendo, depois de 10 anos, sua arquitetura completamente modificada passando a suportar uma vasta variedade de aplicações para a programação de robôs móveis e em diversas plataformas [SAP1997].

O Saphira é uma integração de um conjunto de rotinas de comunicação que controla um robô através de um computador hospedeiro (host). Este sistema é projetado para facilitar a definição das aplicações do robô, a qual esta agregada ao programa cliente. Por causa disto, esta arquitetura é uma arquitetura aberta onde os usuários podem escrever seu próprio sistema de controle de robôs móveis e reutilizar as vantagens das micro-tarefas e propriedades do refletor de estados, o qual gerencia as informações internas do robô (sensores, motores e garras).

A forma de operação do Saphira é através de uma plataforma cliente / servidor, sendo que suas bibliotecas são um conjunto de rotinas para construção de clientes realizando assim todo o trabalho de comunicação e gerenciamento dos dados para o robô servidor.

O cliente Saphira conecta-se a um robô servidor com os componentes básicos para o robô obter as informações necessárias do ambiente e navegar, ou seja, as informações da posição dos encoders, a leitura dos sensores e manipulação do motor e das rodas. Já o servidor manipula as rotinas de mais baixo nível, como enviar e responder aos comandos do cliente, através de um protocolo de comunicação, gerenciar a direção e os sensores do robô.

Este software suporta funções de alto nível para o controle do robô e interpretação dos sensores, incluindo o sistema de planejamento reativo, navegação baseada em mapa, sistema de registro e controle de comportamento fuzzy.

5.1. Simulador do Robô

O Saphira possui um software de simulação do robô físico e de seu ambiente. Estas características foram incluídas para melhorar a análise das aplicações em um computador sem ter que utilizar o hardware do robô para validar os sistemas.

Este simulador implementa todas as principais características de um ambiente real, como por exemplo, distorções para as leituras dos sensores e dos encoders das rodas, com isso a interface de comunicação é a mesma do robô físico fazendo com que não seja necessário à modificação ou ajuste do software para funcionar no robô.

Este simulador constrói os modelos do mundo real ou do seu ambiente em modelos de duas dimensões, sendo estes modelos chamados de mundos. Os modelos de mundo são abstrações de um mundo real, com segmentos lineares representando as superfícies verticais dos corredores, passagens das portas e os objetos presentes nele.

5.2. Linguagem de Programação de Robô – COLBERT

Na versão 6.x do Saphira foi incorporada uma linguagem semelhante com a linguagem C, chamada de COLBERT, a qual é utilizada para desenvolver programas de controle de robô. Com o COLBERT, os usuários podem desenvolver e depurar rapidamente procedimentos de controle complexos, esses procedimentos são chamados de activities. Os activities possuem semânticas de estado-finito a qual conseguem produzir representações procedimentais de conhecimentos de uma seqüência de ações. Activities podem iniciar e parar ações diretas do robô, comportamento de baixo nível, e outros activities. Os activities são utilizados pelo executor de COLBERT, o qual suporta o processamento concorrente entre diversos activities.

O COLBERT vem com um ambiente de avaliação em tempo de execução o qual os usuários podem visualizar a interação de seus programas, editá-los e reexecutá-los, e agregá-los em um código padrão de linguagem C. Muitos usuários programam interativamente com o Colbert, o qual podem desenvolver todas as funções da API do Saphira avaliadas no ambiente runtime.

5.3. Compilador e Executor de Comportamentos

O Saphira usa regras de controle fuzzy para implementar programas de controle de baixo nível. Os comportamentos são definidos utilizando estruturas e funções da linguagem C padrão. Para depurar os comportamentos, deve-se utilizar o compilador de comportamento do Saphira o qual transforma um simples código fuzzy em um código de linguagem C. Na versão 6.1 do Saphira, os comportamentos são um tipo de activities, e podem ser ligados e desligados diretamente da janela de activities.

5.4. Arquitetura do Saphira

O Saphira possui uma arquitetura de um sistema operacional básico para o controle do robô. A Figura 19 mostra a estrutura de uma típica aplicação no Saphira. As rotinas do Saphira estão em azul turquesa e as rotinas do usuário em amarelo. Todas as rotinas do Saphira são executadas a cada ciclo Saphira, ou seja, a cada 100ms, sendo que estas rotinas são invocadas pelo SO do Saphira que constrói estas micro-tarefas.

Este SO também manipula os pacotes de comunicação com o robô, a construção do estado do robô, entre outras tarefas mais complexas como navegação e interpretação de sensores.

5.4.1. Arquitetura de Controle do Saphira

A arquitetura de controle do Saphira é construída no topo do refletor de estados. Ele consiste em um conjunto de micro-tarefas que implementam todas as funções requeridas para a navegação em um ambiente. Um cliente típico poderá usar alguns destes subconjuntos de suas funcionalidades. Mais especificamente, esta arquitetura de controle contém um subconjunto de rotinas que interpretam as leituras dos sensores relativos à posição geométrica do modelo de mundo, bem como um conjunto de rotinas para controlar os estados do robô enquanto ele interage com o ambiente. Pode-se visualizar esta arquitetura na Figura 19.

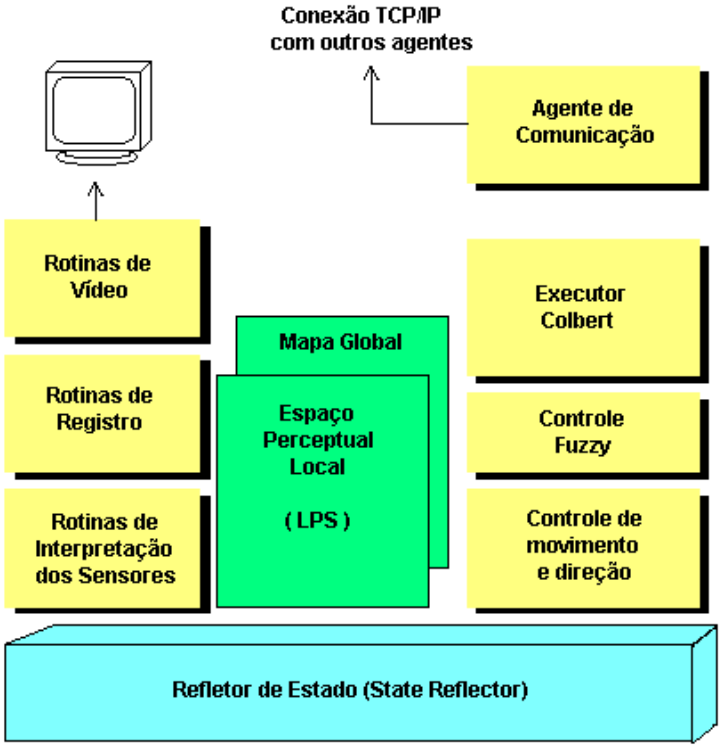


Figura 19 - Arquitetura de Controle do Saphira [SAP1997]

5.4.2. Sistema Operacional do Saphira

A arquitetura do Saphira foi construída baseada na sincronização de processos e interrupções dos sistemas operacionais. As micro-tarefas são máquinas de estado finito (Finite State Machine - FSM) que são registradas com o sistema operacional do Saphira. A cada 100 ms, que é o ciclo do SO, todas as máquinas de estado finito registradas são atualizadas e por causa deste tempo fixo, todas as FSM operam em sincronismo, pois elas podem depender dos estados anteriores, sem que eles sejam atualizados. Por causa deste tempo de 100ms de ciclo, esta arquitetura suporta controle reativo do robô, devido as suas respostas rápidas de mudança das condições do ambiente.

5.4.3. Rotinas do Usuário

As rotinas do usuário (micro-tarefas) são sincronizadas a todo ciclo do SO do Saphira, assim como as rotinas de biblioteca do Saphira. Com isso, as micro-tarefas do usuário são uma extensão das rotinas de biblioteca e podem acessar a arquitetura do sistema em qualquer nível. Tipicamente o nível mais baixo que são as rotinas do usuário trabalharão com o refletor de estados, o qual é uma visão abstrata do estado interno do robô.

As tarefas são executadas nos mesmos 100ms de ciclo como as outras micro-tarefas. Os sincronismos destas rotinas são separados por threads de execução, que compartilham um espaço de endereço comum com as rotinas de biblioteca do Saphira, mas são independentes dos 100 ms do ciclo do Saphira. O usuário pode inicializar muitos destes threads separados, desde que eles respeitem as limitações do sistema operacional do hospedeiro. O sistema do Saphira possui prioridade sobre seus threads de usuário, assim ele consegue planejar o tempo de consumo destas threads e com isso ele consegue coexistir com a arquitetura do sistema Saphira sem afetar o tempo real do controle do robô.

5.4.4. Pacote de Comunicação

O Saphira possui um protocolo de comunicação baseado em pacotes utilizado para enviar comandos para o robô servidor e receber informações sobre o refletor de estados do robô.

Tipicamente os clientes enviam em torno de 1 a 4 comandos por segundo, e todos os clientes recebem 10 pacotes por segundo do robô. Estes pacotes contêm informações da leitura dos sensores e informações sobre movimentação do robô. Estes pacotes contêm o tamanho de 30 a 50 bytes por pacote e são transmitidos a 9600 baud. O Saphira tem a capacidade de conectar ao robô servidor sobre uma linha TTY, ethernet com tcp/ip, ou link local IPC.

Existe ainda um pacote de checksum para determinar se um pacote esta corrompido ou não, isto ocorre pelo fato de que o Saphira é muito sensível a perda de informações. Porém, se o canal de comunicação tiver muita interferência na transmissão, esses pacotes de checksum e correção começarão a sobrecarregar o canal, degradando assim o desempenho do Saphira.

5.4.5. Refletor de Estado (State Reflector)

Para controlar o robô seria necessário interpretar os dados incorporados nos pacotes de comunicação para depois utilizá-los em nosso programa de controle do robô, porém o Saphira incorpora um refletor de estado interno que é um espelho dos estados do robô em um computador hospedeiro (host).

Essencialmente, o refletor de estados é uma visão abstrata dos estados internos do robô, o qual contém as informações sobre os seus sensores e movimentos. Similarmente, para controlar o robô, uma rotina é apenas um conjunto apropriado de variáveis de controle do refletor de estados, e as rotinas de comunicação irão enviar comandos apropriados para o robô.

A Figura 20 demonstra o funcionamento da conexão do robô com o cliente Saphira.

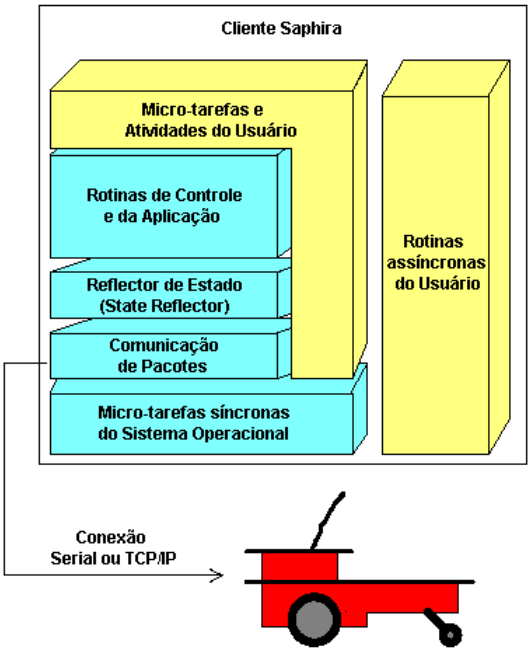


Figura 20 - Conexão do Cliente Saphira [SAP1997]

5.4.6. Representação de Espaço

Robôs móveis operam em espaços geométricos, e a representação destes espaços influência muito o seu desempenho, por isso, existem duas formas principais de representação geométrica no Saphira. Uma delas é o espaço local percebido, o qual é representado por um sistema de coordenadas egocêntricas, com muitos metros em radianos, centralizados no robô. Para uma maior perspectiva, o Saphira usa um mapa de espaço global para representar os objetos que fazem parte do ambiente do robô especificado em coordenadas absolutas (globais).

O espaço local percebido é comumente conhecido como mapa local, onde este mapa expressa os dados referentes a um perímetro do ambiente, ou seja, ele representa apenas uma parte do ambiente. Já no mapa de espaço global, tem todas as características do ambiente a ser navegado e das coordenadas globais as quais o robô se encontra referente ao ambiente de um modo geral.

5.4.7. Rotinas de Interpretação dos Sensores

As rotinas de interpretação dos sensores são processo que extraem dados dos sensores ou do mapa local, e retornam as informações para o mapa local. Os processos interpretativos de atividades do Saphira. Detecção de obstáculos, reconstrução de superfícies, e reconhecimento de objetos são alguns das rotinas que existem correntemente, todas trabalhando com dados refletidos dos sonares e dos sensores de movimento.

5.5. Registros e Mapas

No mapa global de espaço, o Saphira mantém uma estrutura de um conjunto de dados internos (artefatos) que representam o ambiente. Os artefatos incluem corredores, portas, paredes e salas. Estes mapas podem ser criados diretamente no arquivo de mapa ou extraindo informações relevantes do ambiente em que o robô está navegando.

O registro é um processo de manter a localização global do robô em um mapa interno consistente com a leitura dos sensores do ambiente local. Existem rotinas para extrair informações relevantes do mapa local e comparar com as estruturas do mapa global, para atualizar então a posição do robô.

Capítulo 6 - Mapas Topológicos em Hardware Reconfigurável

Esta pesquisa vem analisar as possíveis formas de implementação de mapas topológicos em hardware reconfigurável, para a melhoria de desempenho. A construção do mapa topológico fundamentou-se na implementação de um mapa topológico baseado em detecção de espaços livres [DUC98; DUC99; DUC99A; DUC2000], e utilizando-se o ambiente de desenvolvimento Excalibur para a conversão do software em hardware reconfigurável.

6.1. Contribuição para o projeto ARMOSH

Este trabalho vem com o intuito de ser incorporado ao projeto ARMOSH, de forma que esse trabalho auxilie a navegação do robô em um ambiente estático e fechado. Esta contribuição é feita através da implantação do mapeamento topológico em uma FPGA.

Além da contribuição para o projeto ARMOSH, esse trabalho contribuirá com a sociedade científica através do incentivo a implementação de sistemas robóticos em FPGAs, pois atualmente são poucos os sistemas baseados em robôs móveis que utilizam desta tecnologia para o seu desenvolvimento.

6.2. Detecção de Espaços Livres

O método de detecção de espaços livres (descrito no item 3.2.1) para mapas topológicos a ser implementado neste trabalho foi proposto por [DUC99]. Este método utiliza uma rede neural artificial feedforward (descrita no item 2.2.1), já treinada [Figura 21].

A rede neural artificial feedforward é composta por:

- *Função de ativação* do tipo linear para representar o grau de incerteza da saída da rede neural artificial. A função matemática referente à função de ativação será analisada tendo como ponto principal para otimização do hardware reconfigurável;
- *Arquitetura*:
  - *Rede de múltiplas camadas* para realizar a comparação dos dados dos sensores, enviando assim, uma informação mais precisa;
  - *Conexão do tipo feedforward* que auxiliará na correcao de erros da rede neural artificial no momento da aprendizagem;
  - *Rede completamente conectada* que auxiliará na detecção de padrões introduzidos no aprendizado.

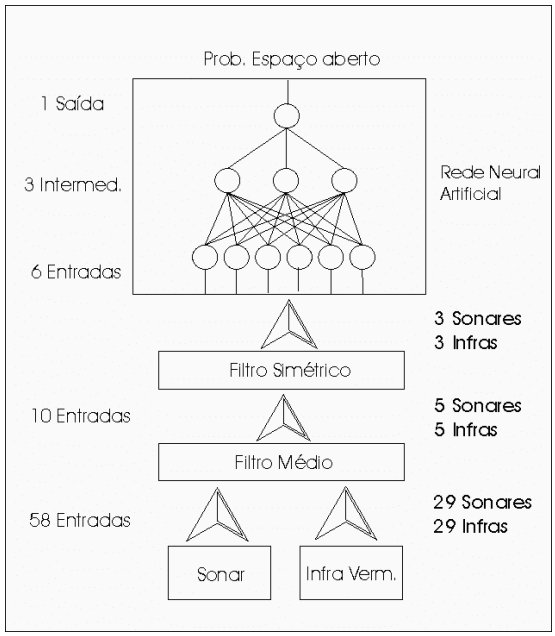


Figura 21 - Estrutura de Localização de Espaço Livre [DUC99].

Essa rede neural artificial é fundamental na fase de mapeamento topológico segundo [DUC99], pois dentro do contexto geral de mapeamento topológico, ela é a responsável pela captura dos sinais dos sensores filtrados e pela análise destes dados. Sendo à saída desta rede neural artificial feedforward considerada como a probabilidade da existência de um espaço livre para o planejamento e navegação de um robô em uma dada direção. Ela já possui um aprendizado para o reconhecimento de espaços livres.

As informações fornecidas por esta rede serão utilizadas pelo módulo de cálculo de posição final do mapa topológico, indicando ao mapa topológico onde uma região é caracterizada como lugares "previstos" ou lugares "confirmados" (descritos no item 3.2.1). Já no módulo de cálculo de posição final do mapa topológico, será analisada essa informação e criará novas rotas dentro do mapa topológico, descartando os lugares que o robô não conseguirá se locomover devido à presença de objetos. Os filtros mostrados na Figura 21 são utilizados para a remoção dos possíveis ruídos dos sensores, de modo a melhorar a confiabilidade dos dados de entrada rede neural artificial feedforward.

6.3. Hardware da Rede Neural Artificial

Ao se analisar as estruturas das redes neurais de forma genérica, nota-se que os algoritmos possuem simplicidade e um grande paralelismo em suas operações matemáticas. Com esta característica as redes neurais viabilizam a sua implementação em hardware, tendo inclusive como vantagem o seu altíssimo desempenho quando comparada com as mesmas versões em software [DEM96, MOL2000, MOE96, MOR99].

Um dos assuntos mais discutidos em todos os trabalhos citados vem sendo exatamente a exploração de paralelismo desses algoritmos de redes neurais artificiais, já que sua implementação tradicional em software permite apenas um fluxo seqüencial na execução instruções, ignorando completamente todo o paralelismo natural inerente às redes neurais artificiais. Quando essas redes são implementadas em hardware torna-se possível à exploração de seu paralelismo inerente a seus algoritmos.

Basicamente existem dois tipos diferentes de implementação de redes neurais em hardware: as que são treinadas pelo próprio hardware, e as redes que são treinadas *off-line* e depois implementadas. Este trabalho se concentra no segundo tipo de rede neural.

6.3.1. Hardware da Rede Neural para Detecção de Espaços Livres

Como visto no item 6.2 será utilizada neste trabalho uma rede neural artificial para a detecção de espaços livres para a construção do mapa topológico. Porém para se conseguir um paralelismo maior e melhorar o desempenho da rede neural artificial, essa rede será transformada de software para hardware. Para isso foram realizadas diversas modificações em sua estrutura original citada por [DUC99].

A rede neural ilustrada na Figura 22 esta estruturada assim:

- *Múltiplas Camadas* possui três camadas onde: A primeira camada com 5 neurônios; a segunda camada com 3 neurônios e a terceira camada com 1 neurônio;
- As conexões entre os neurônios é do tipo *FeedForward*;
- As conexões estão *fracamente conectadas*;
- As funções de ativação são: Na primeira camada, na segunda e na terceira camada a função é do tipo *Linear*;
- Pesos definidos conforme treinamento supervisionado, ou seja, a rede a ser implementada já esta treinada.

Na Figura 22 está apresentada a rede neural desenvolvida em software a qual foi convertido em hardware.

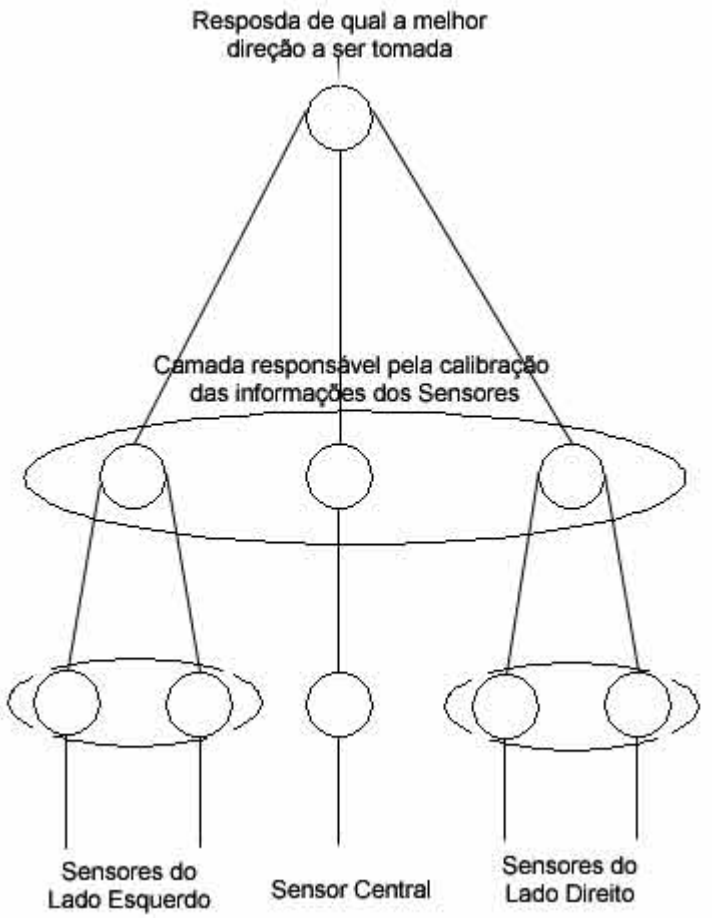


Figura 22 - Rede Neural para Detecção de Espaços Livres – Diagramação da Rede

Devido aos avanços nas ferramentas de projeto e desenvolvimento de hardware encontrado nos dias atuais, a conversão desta rede neural em hardware foi simplificada de forma que suportasse as configurações de testes realizadas no robô PIONEER I.

A estrutura atual esta descrita abaixo:

- *Primeira Camada*: conseguiu-se a simplificação da primeira camada de modo que a essa camada fosse convertida em um barramento que transmite para a segunda camada as entradas dos sensores ponderadas;
- *Segunda Camada*: transmite para a terceira camada o resultado dos filtros dos sensores refinados.

- *Terceira Camada*: o resultado da saída é um valor que identifica qual dos lados (Esquerdo, Centro, Direta) o robô poderá se locomover.

Esta rede neural artificial ficará acoplada ao sistema de locomoção primário que auxiliará este sistema na construção do mapa topológico e também na navegação primária.

A descrição do hardware da rede neural artificial, a qual detecta os espaços abertos, esta ilustrada nas Figuras 23, 24 e 25 que é o projeto desenvolvido no software Quartus.

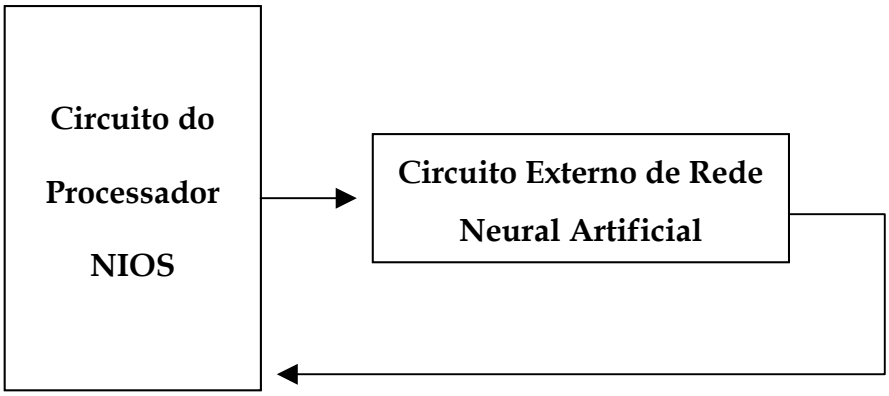


Figura 23 - Esquemático do Circuito Principal da Configuração da FPGA

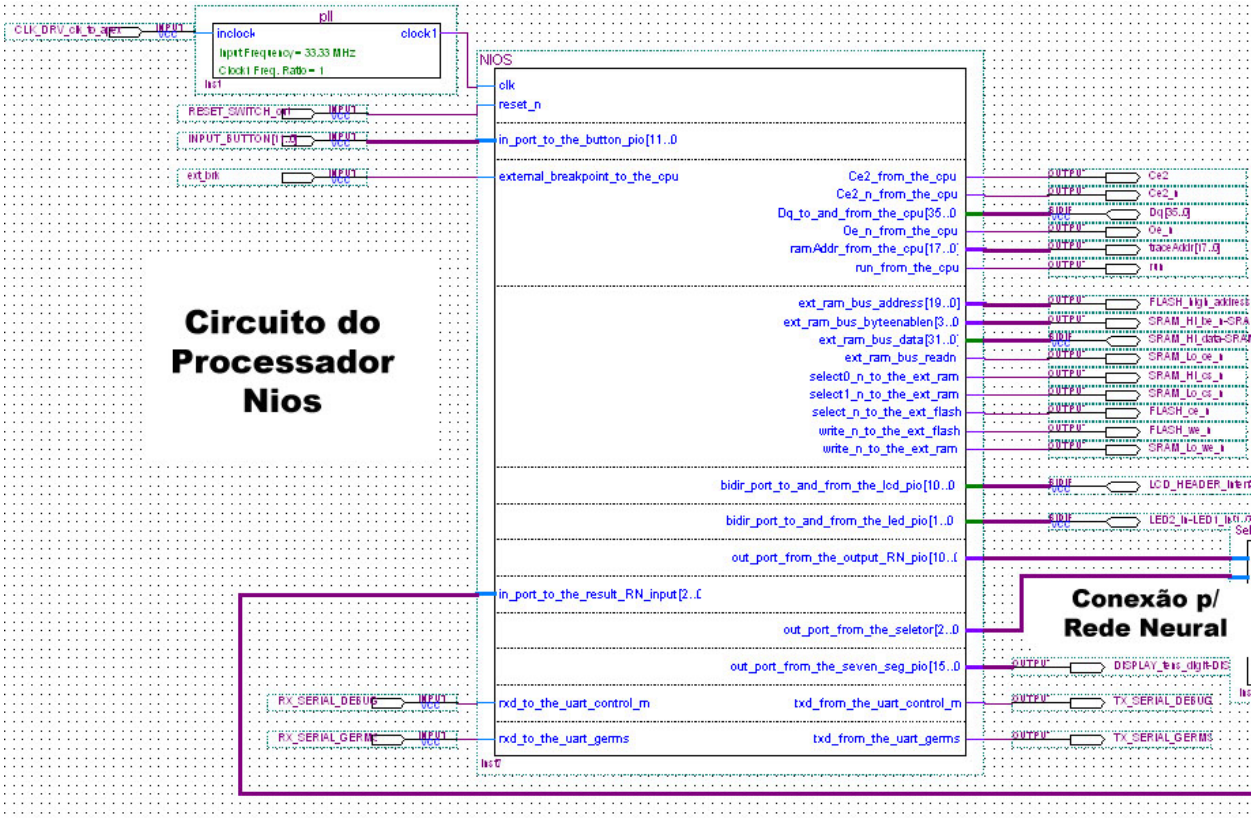


Figura 24 - Circuito do Processador NIOS

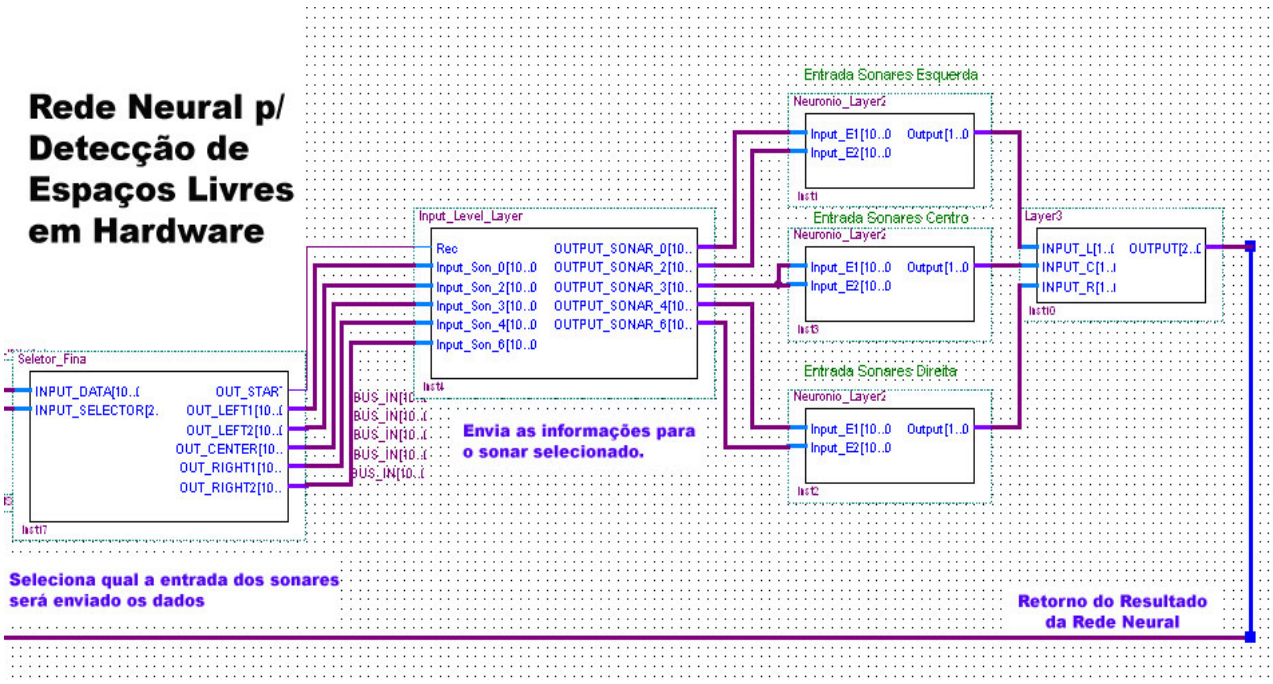


Figura 25 - Rede Neural implementada em Hardware

Este hardware de rede neural recebe as informações do processador NIOS, verifica as informações conforme aprendizado pré-definido e após a conclusão da rede neural o valor resultante destes cálculos retorna para o processador NIOS.

6.4. Construção do Mapa Topológico

Para construir mapas topológicos precisa-se, antes de qualquer coisa, filtrar a leitura dos sensores do sistema de navegação, o qual irá identificar a posição atual do robô para então ocorrer à inclusão desta posição no mapa topológico. Este registro indica o posicionamento global do robô em relação ao seu ambiente.

Como descrito no item 3.2 e 3.2.1, os mapas topológicos são mapas construídos baseados na teoria de grafos, que indicam quais as posições foram exploradas e quais não foram.

Este mapa contém informações que auxiliam no desenvolvimento de sistemas de navegação que necessita de uma referência global do robô perante seu ambiente. Ele é considerado um ótimo mapeador para ambientes estáticos, ou seja, ambientes onde não há mudanças constantes de seus objetos e/ou pessoas.

A sua grande vantagem, como descrito anteriormente, é o seu baixo custo computacional, porém, as informações contidas neste mapa não apresentam coordenadas para uma localização mais precisa e regional.

O sistema desenvolvido aqui para a criação de mapas topológicos segue as seguintes premissas:

- Para a construção do mapa é necessário um sistema de navegação simples;
- Os dados do sistema de navegação serão filtrados e analisados de forma que possa ser extraída a informação necessária para o sistema de forma geral;
- Os dados anteriormente citados servirão como alimentação de rede neural artificial a qual irá avaliar as informações de forma que o resultado final desta avaliação será qual o melhor caminho a ser explorado;

- Ao descobrir-se qual o melhor caminho a ser explorado o sistema de navegação irá movimentar o robô e ao mesmo tempo irá atualizar o mapa topológico incluindo nele a distância percorrida e qual direção descrita pela rede neural (Esquerdo, Centro, Direito).

6.5. Etapas de implementação

Após ser realizado todo o estudo relacionado à navegação e decidir a melhor forma de implementar um sistema de mapeamento topológico iniciou a fase de implementação deste sistema em linguagem C.

No desenvolvimento em linguagem C o software de mapeamento foi estruturado com algoritmos de alocação dinâmica e linguagem C padrão (ANSI).

Após toda a estruturação do código em linguagem C padrão, esse software foi convertido para a linguagem Microsoft Visual C++ versão 5, que o software de simulação Saphira utiliza como linguagem de programação do sistema. Com isso o software já está pronto para ser validado, ou seja, já está pronto para a realização dos testes de funcionalidade.

Os testes realizados com o software de mapeamento utilizaram os mapas existentes no próprio simulador Saphira, ilustrado na Figura 26. O sistema Saphira emulou a leitura dos sensores (sonares) para o software de mapeamento para a construção do mapa topológico.

Com o sistema testado e validado no simulador Saphira, iniciou-se a fase de conversão deste software desenvolvido em MS Visual C++ para a linguagem C GNUPro que é a linguagem de desenvolvimento do kit do Excalibur. Houve algumas mudanças nas estruturas de chamadas do simulador para o Excalibur, porém a essência do programa não foi modificada.

Uma vez convertido o software para o Excalibur este código teve que ser reajustado no seguinte aspecto: o Excalibur não tem a capacidade de adquirir a leitura dos sensores através do Saphiro ou de qualquer outro dispositivo que enviasse leituras do ambiente.

A primeira tentativa para a aquisição de informações do mundo externo, foi através de um dispositivo a laser que faz a varredura do ambiente (Laser Rang Finder da empresa Sick). O grande problema encontrado nesta interface foi que o ambiente Excalibur utiliza a linha de comunicação serial com um host o qual faz o download de sua programação, com isso o laser não consegue realizar a comunicação com Excalibur.

O mesmo problema ocorreu com a tentativa de desenvolver uma interface a qual teria como função captar as leituras dos sensores do Saphira e transporta-las para o ambiente Excalibur.

Com esses problemas encontrados, a solução foi utilizar as estruturas do Excalibur para simular a leitura dos sensores. O Excalibur utiliza modelos matemáticos para a realização desta simulação.

Concluída a fase de conversão do software de mapeamento topológico, foi estruturado o desenvolvimento de um hardware dedicado, que identifica qual o melhor caminho ser tomado.

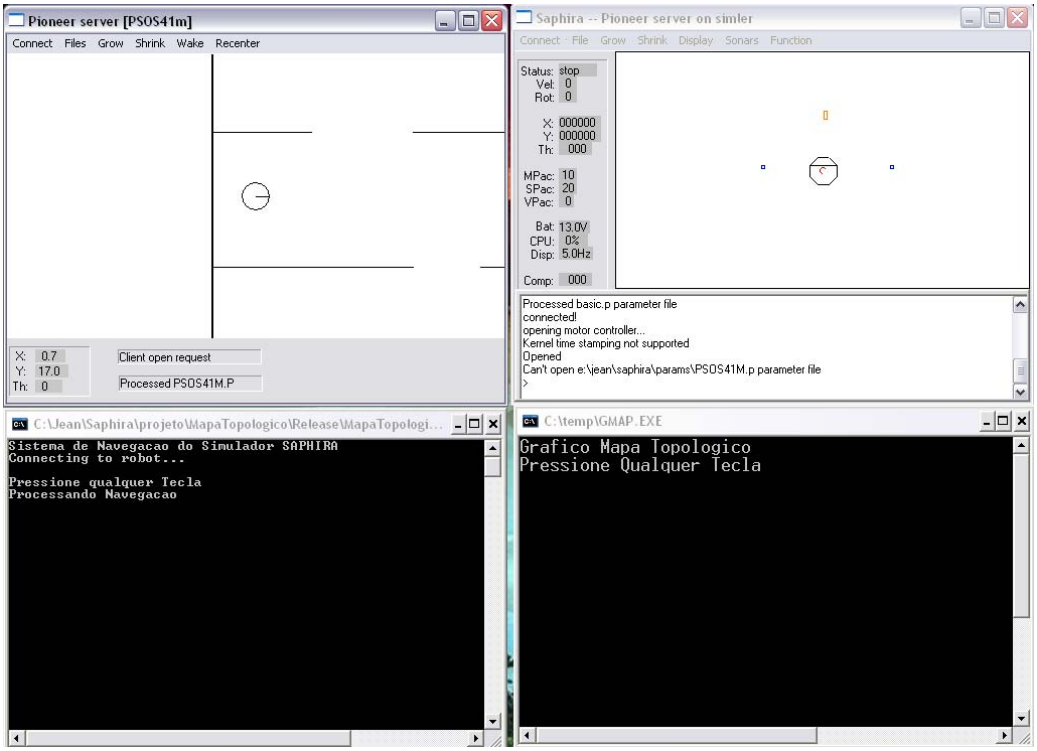
Após estas etapas concluiu-se que todo o software foi migrado dentro de uma FPGA ALTERA APEX200K200EFC, onde parte do software rodava com o software core do processador NIOS e a outra parte para o hardware dedicado.

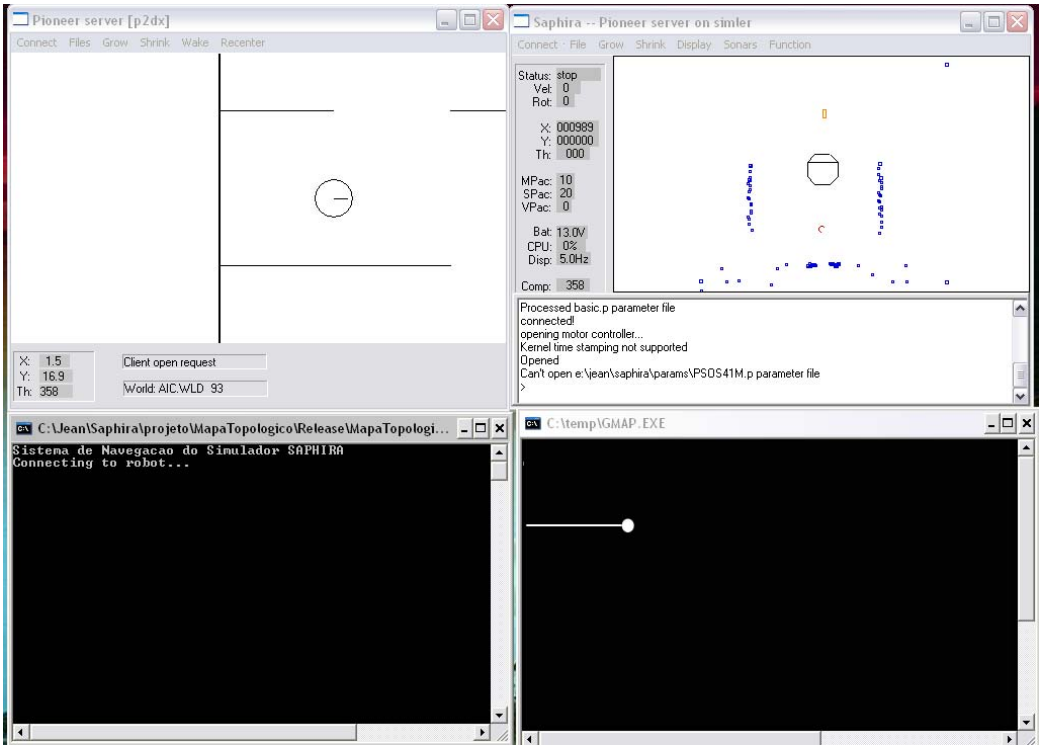
6.6. Resultados

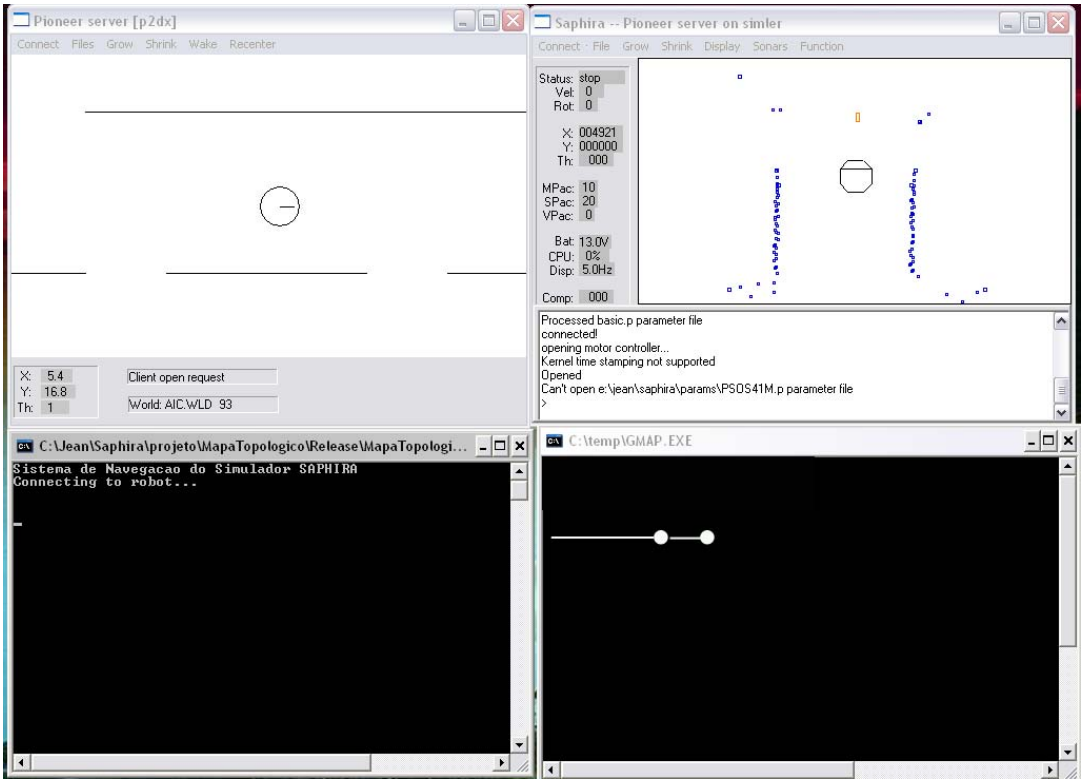
O resultados apresentados neste trabalho foram extraídos da execução dos sistemas de mapeamento topológico no ambiente Excalibur e do sistema de mapeamento topológico utilizando o ambiente de simulação de robôs Saphira. Antes de descrever os resultados em tempos de execução, será apresentada às ilustrações resultantes da execução desses sistemas.

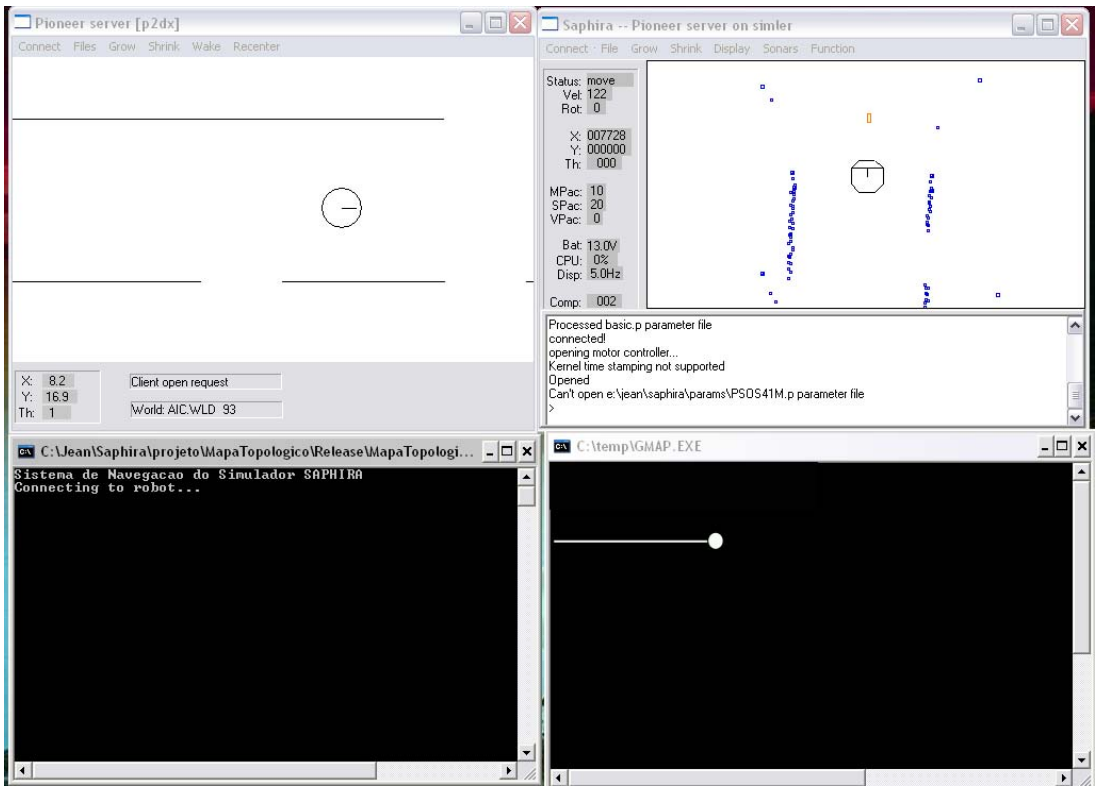
Primeiramente será apresentada a ilustração do sistema de mapeamento topológico no ambiente Saphira. Essas ilustrações podem ser visualizadas através da Figura 26, que esta dividia em duas partes:

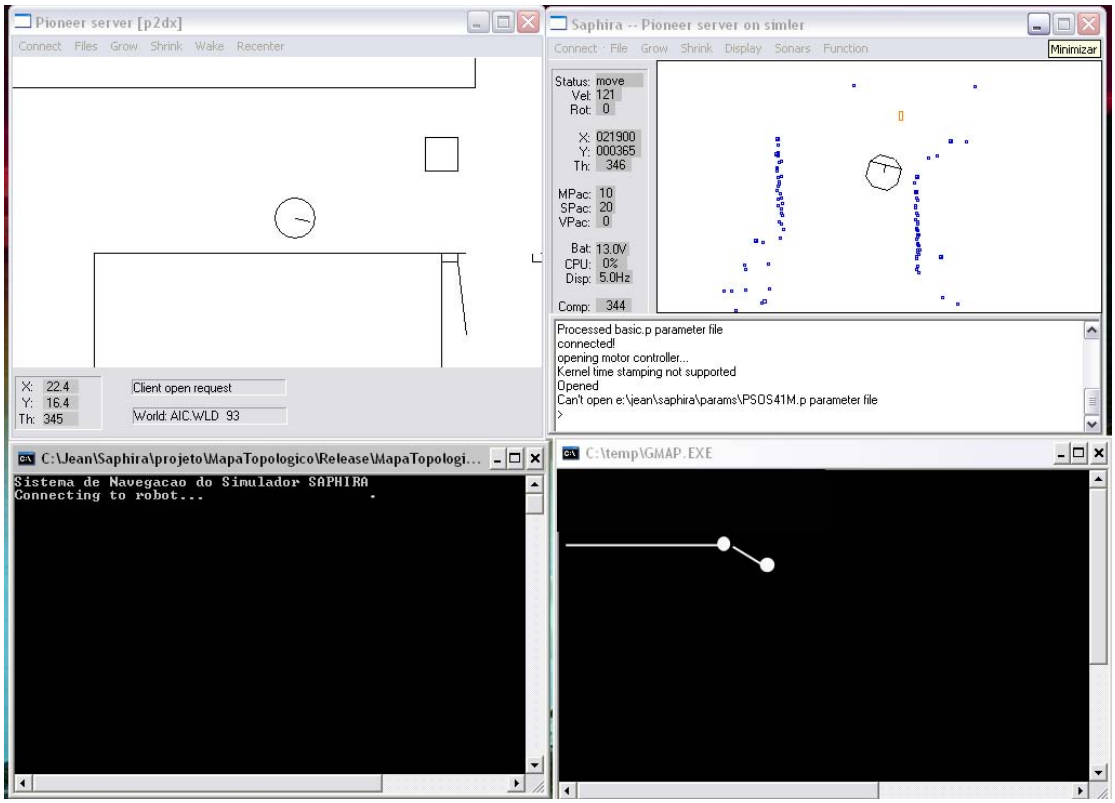
- A parte superior das imagens é representada pelo funcionamento do software no sistema Saphira;
- A parte inferior das imagens é representada pelo software gráfico de geração de mapas topológicos, desenvolvidos em linguagem C, que captura as informações do software Saphira através de um arquivo texto e cria o mapa topológico baseado neste arquivo.

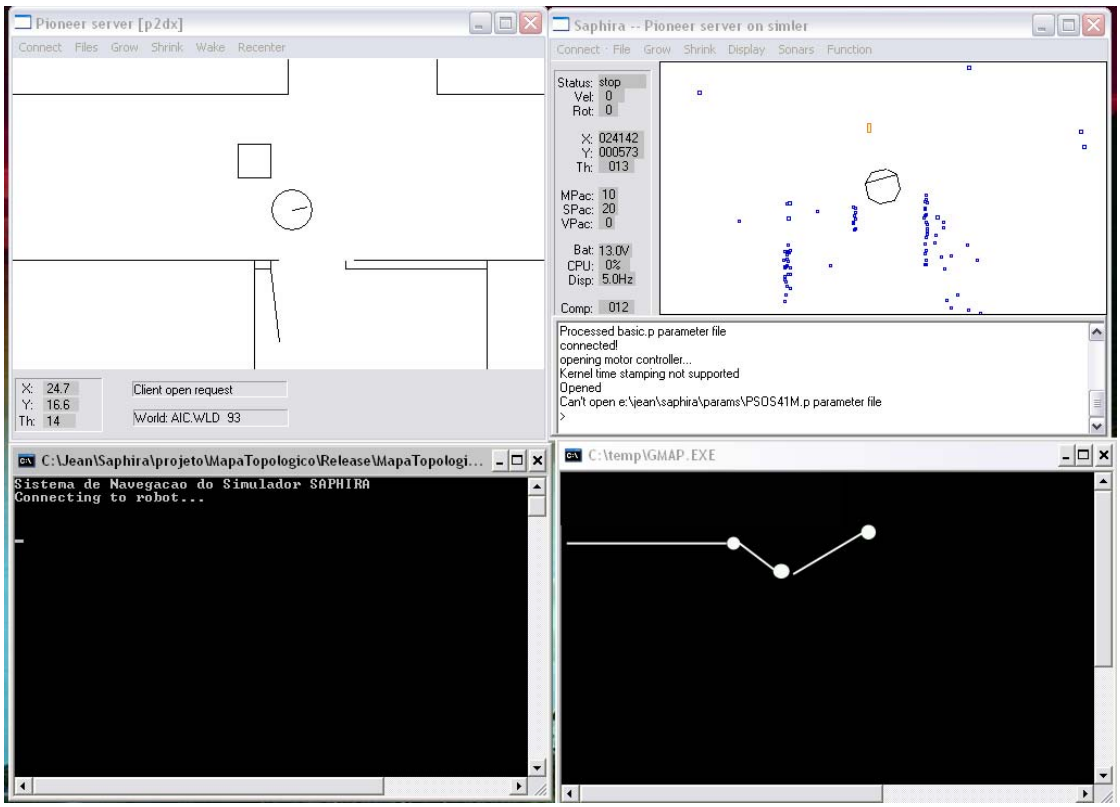


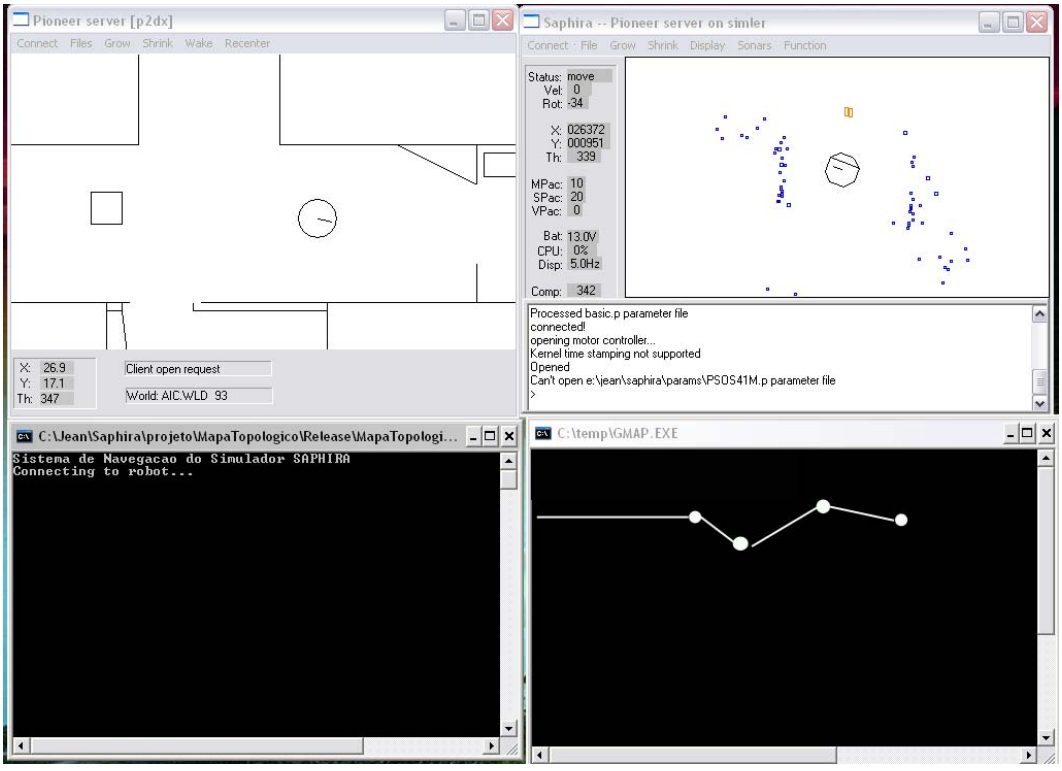

Pioneer server [p2dx]
Connect Files Grow Shrink Wake Recenter
X: 1.5
Y: 16.9
Th: 358
Client open request
World: AIC.WLD 93
Saphira -- Pioneer server on simler
Connect File Grow Shrink Display Sonars Function
Status: stop
Vel: 0
Rot: 0
X: 000989
Y: 000000
Th: 000
MPac: 10
SPac: 20
VPac: 0
Bat: 13.0V
CPU: 0%
Disp: 5.0Hz
Comp: 358
Processed basic.p parameter file
connected!
opening motor controller...
Kernel time stamping not supported
Opened
Can't open e:\jean\saphira\params\PSOS41M.p parameter file
C:\Jean\Saphira\projeto\MapaTopologico\Release\MapaTopologi...
Sistema de Navegacao do Simulador SAPHIRA
Connecting to robot...
C:\temp\GMAP.EXE


Pioneer server [p2dx]
Connect Files Grow Shrink Wake Recenter
X: 5.4
Y: 16.8
Th: 1
Client open request
World: AIC.WLD 93
Saphira -- Pioneer server on simler
Connect File Grow Shrink Display Sonars Function
Status: stop
Vel: 0
Rot: 0
X: 004921
Y: 000000
Th: 000
MPac: 10
SPac: 20
VPac: 0
Bat: 13.0V
CPU: 0%
Disp: 5.0Hz
Comp: 000
Processed basic.p parameter file
connected!
opening motor controller...
Kernel time stamping not supported
Opened
Can't open e:\jean\saphira\params\PSOS41M.p parameter file
C:\Jean\Saphira\projeto\MapaTopologico\Release\MapaTopologi...
Sistema de Navegacao do Simulador SAPHIRA
Connecting to robot...
C:\temp\GMAP.EXE


Pioneer server [p2dx]
Connect Files Grow Shrink Wake Recenter
X: 8.2
Y: 16.9
Th: 1
Client open request
World: AIC.WLD 93
Saphira -- Pioneer server on simler
Connect File Grow Shrink Display Sonars Function
Status: move
Vel: 122
Rot: 0
X: 007728
Y: 000000
Th: 000
MPac: 10
SPac: 20
VPac: 0
Bat: 13.0V
CPU: 0%
Disp: 5.0Hz
Comp: 002
Processed basic.p parameter file
connected!
opening motor controller...
Kernel time stamping not supported
Opened
Can't open e:\jean\saphira\params\PSOS41M.p parameter file
C:\Jean\Saphira\projeto\MapaTopologico\Release\MapaTopologi...
Sistema de Navegacao do Simulador SAPHIRA
Connecting to robot...
C:\temp\GMAP.EXE


Pioneer server [p2dx]
Connect Files Grow Shrink Wake Recenter
X: 22.4
Y: 16.4
Th: 345
Client open request
World: AIC.WLD 93
Saphira -- Pioneer server on simler
Connect File Grow Shrink Display Sonars Function
Minimizar
Status: move
Vel: 121
Rot: 0
X: 021900
Y: 000365
Th: 346
MPac: 10
SPac: 20
VPac: 0
Bat: 13.0V
CPU: 0%
Disp: 5.0Hz
Comp: 344
Processed basic.p parameter file
connected!
opening motor controller...
Kernel time stamping not supported
Opened
Can't open e:\jean\saphira\params\PSOS41M.p parameter file
C:\Jean\Saphira\projeto\MapaTopologico\Release\MapaTopologi...
Sistema de Navegacao do Simulador SAPHIRA
Connecting to robot...
C:\temp\GMAP.EXE


Pioneer server [p2dx]
Connect Files Grow Shrink Wake Recenter
X: 24.7
Y: 16.6
Th: 14
Client open request
World: AIC.WLD 93
Saphira -- Pioneer server on simler
Connect File Grow Shrink Display Sonars Function
Status: stop
Vel: 0
Rot: 0
X: 024142
Y: 000573
Th: 013
MPac: 10
SPac: 20
VPac: 0
Bat: 13.0V
CPU: 0%
Disp: 5.0Hz
Comp: 012
Processed basic.p parameter file
connected!
opening motor controller...
Kernel time stamping not supported
Opened
Can't open e:\jean\saphira\params\PSOS41M.p parameter file
>
C:\Jean\Saphira\projeto\MapaTopologico\Release\MapaTopologi...
Sistema de Navegacao do Simulador SAPHIRA
Connecting to robot...
C:\temp\GMAP.EXE


Pioneer server [p2dx]
Connect Files Grow Shrink Wake Recenter
X: 26.9
Y: 17.1
Th: 347
Client open request
World: AIC.WLD 93
Saphira -- Pioneer server on simler
Connect File Grow Shrink Display Sonars Function
Status: move
Vel: 0
Rot: -34
X: 026372
Y: 000951
Th: 339
MPac: 10
SPac: 20
VPac: 0
Bat: 13.0V
CPU: 0%
Disp: 5.0Hz
Comp: 342
Processed basic.p parameter file
connected!
opening motor controller...
Kernel time stamping not supported
Opened
Can't open e:\jean\saphira\params\PSOS41M.p parameter file
>
C:\Jean\Saphira\projeto\MapaTopologico\Release\MapaTopologi...
Sistema de Navegacao do Simulador SAPHIRA
Connecting to robot...
C:\temp\GMAP.EXE

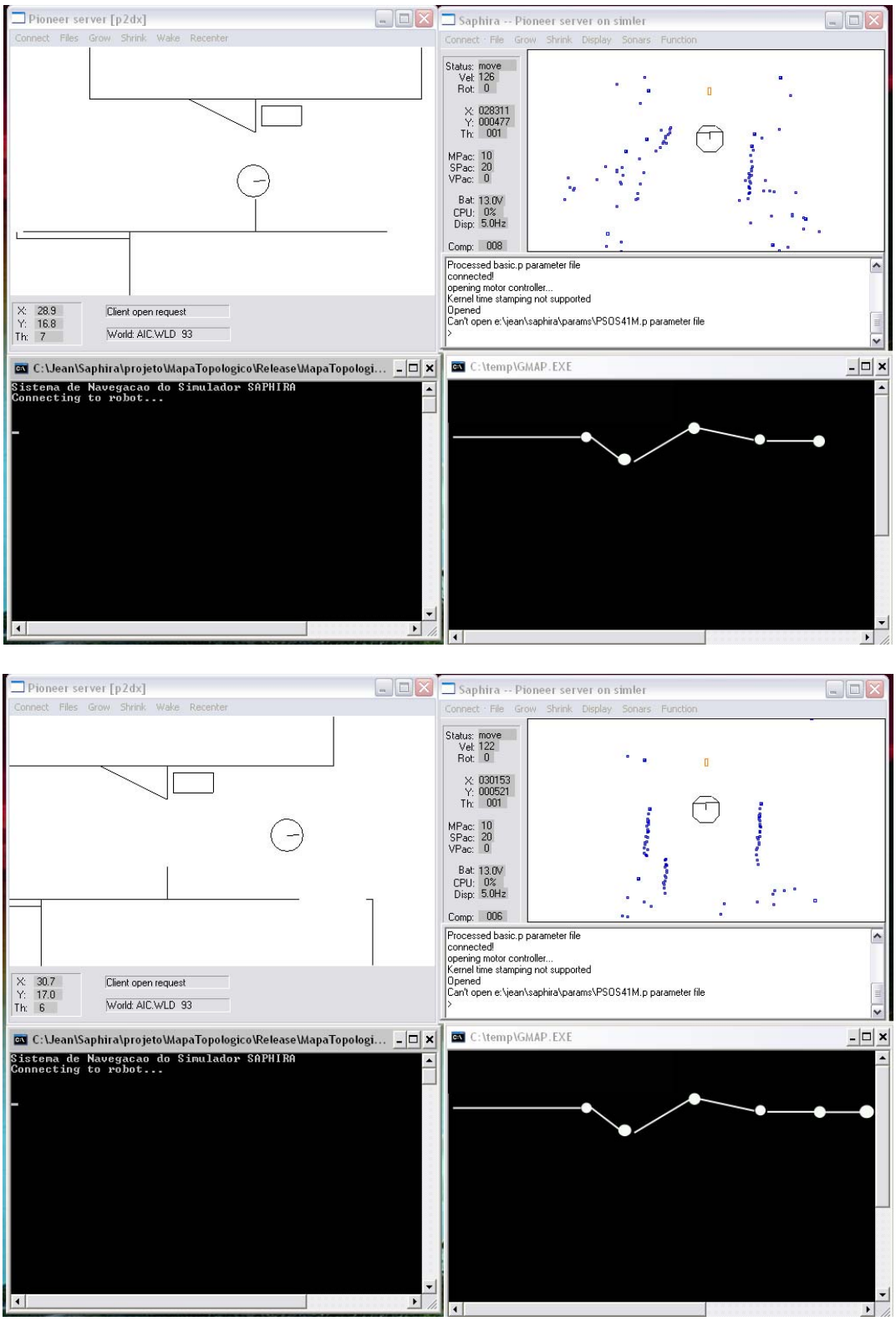
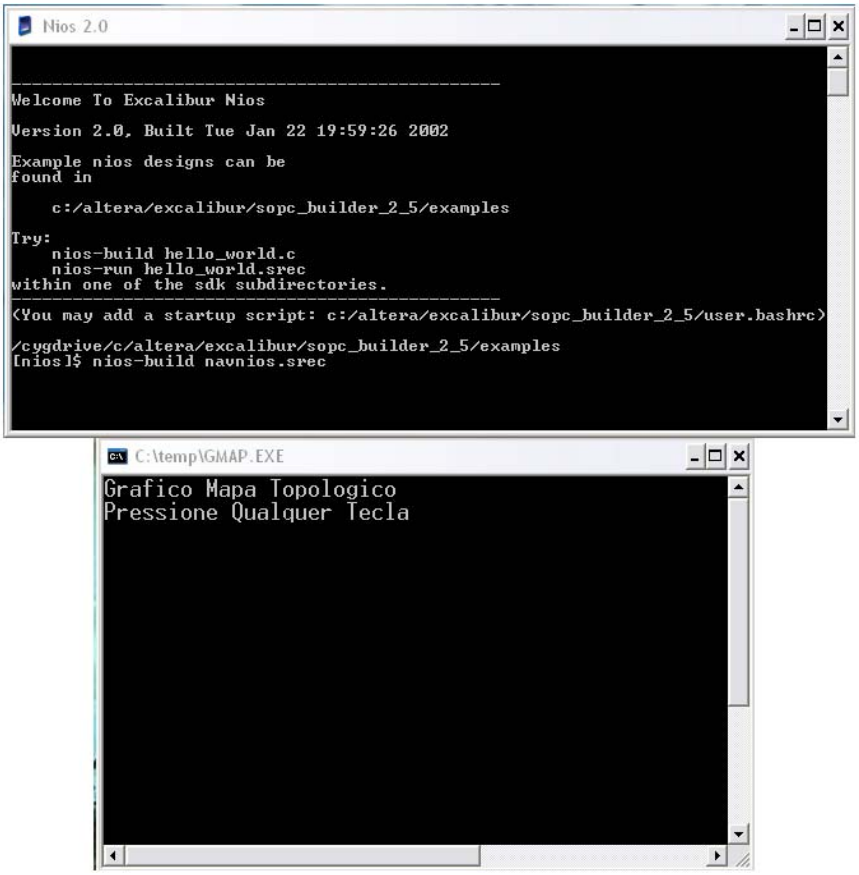


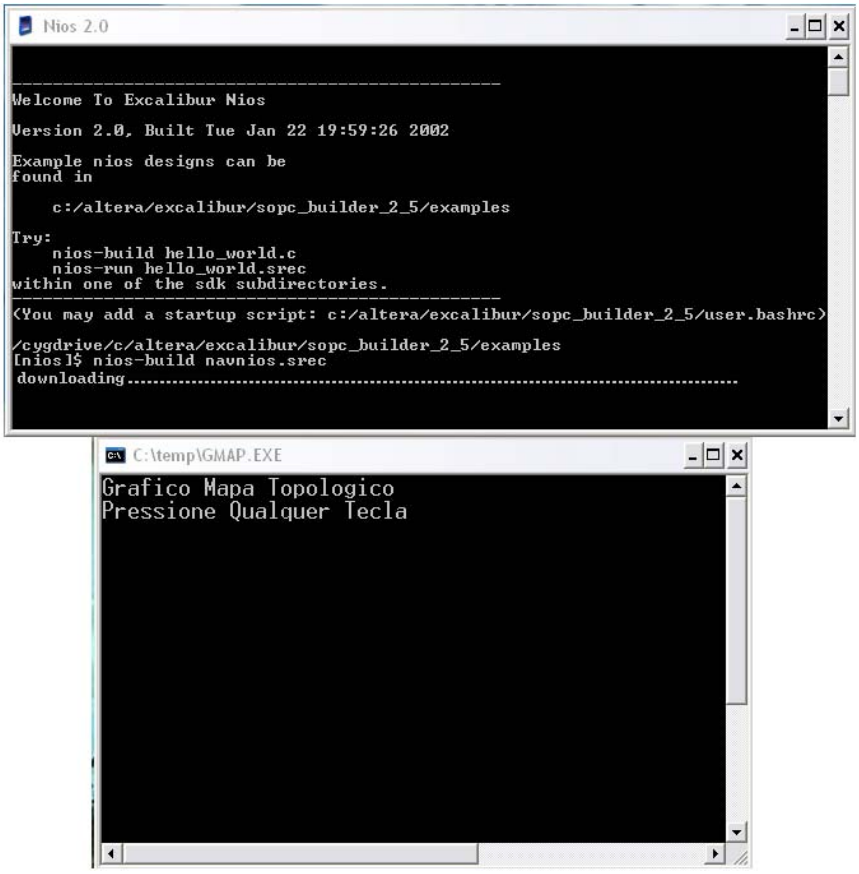
Figura 26 - Validação da Construção do Mapa Topológico com o Saphira

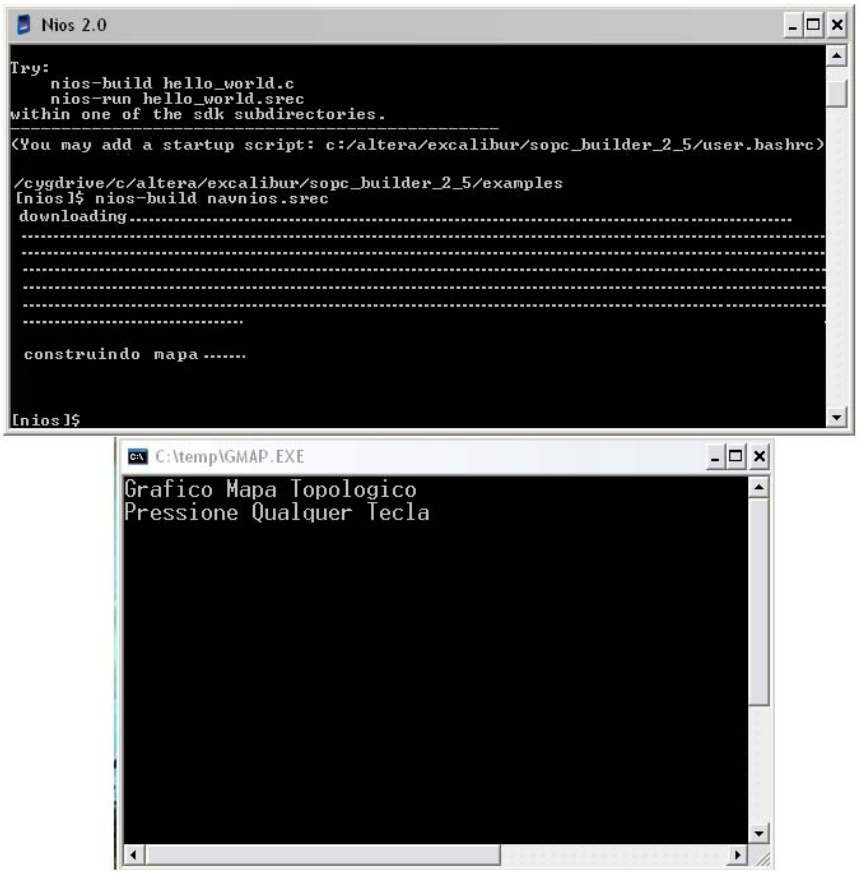
As ilustrações da Figura 26 demonstram todos os estados realizados pelo software de navegação de um ambiente simulado pelo software Saphira.

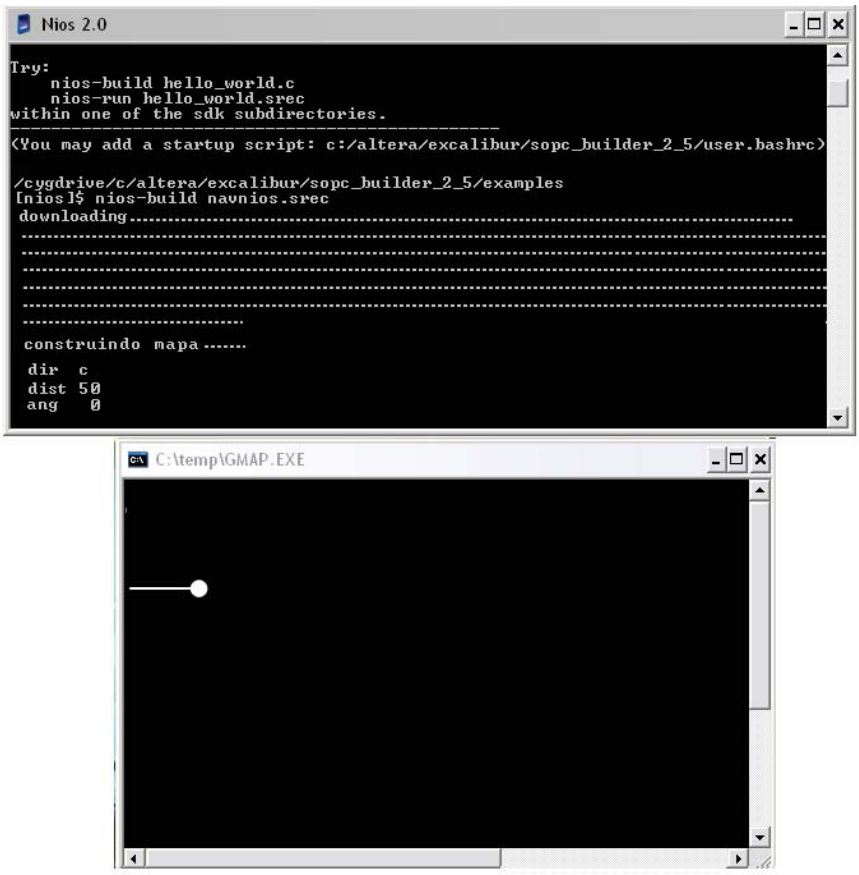
Após a validação do software de mapeamento no ambiente Saphira, foi utilizado os resultados dessa validação para a realização de testes e validação do sistema de mapeamento para o ambiente Excalibur. Esses resultados podem ser visualizados na Figura 27, que é composta por:

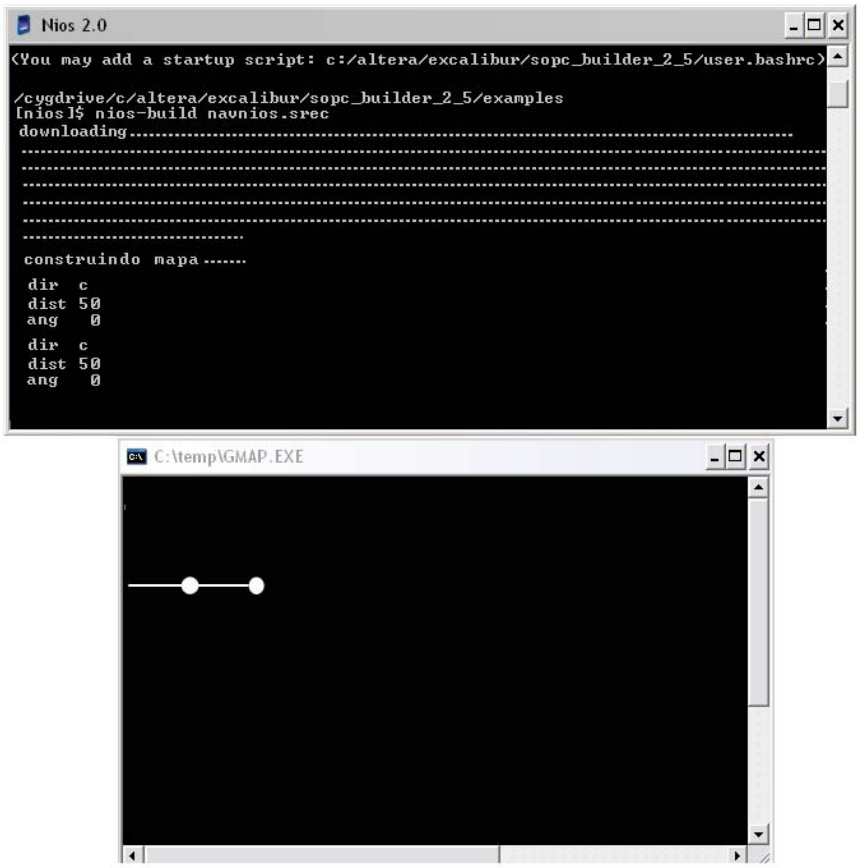
- *Imagem superior*, imagem retirada da execução do software de mapeamento utilizando o ambiente Excalibur. As informações contidas nestas imagens são referentes às rotas previstas pelo sistema para a construção do mapa topológico. Essas informações estão descritas da seguinte forma:
    - *Dir* informa à direção que o sistema percorreu (D – Direita, E – Esquerda, C – Centro) ;
    - *Dist* informa a distância percorrida em milímetros;
    - *Ang* informa o ângulo utilizado na rotação ou movimentação do robô;
- *Imagem inferior*, imagem retirada do software gráfico de construção de mapa topológicos que foi utilizado na construção de mapa topológico no sistema Saphira.

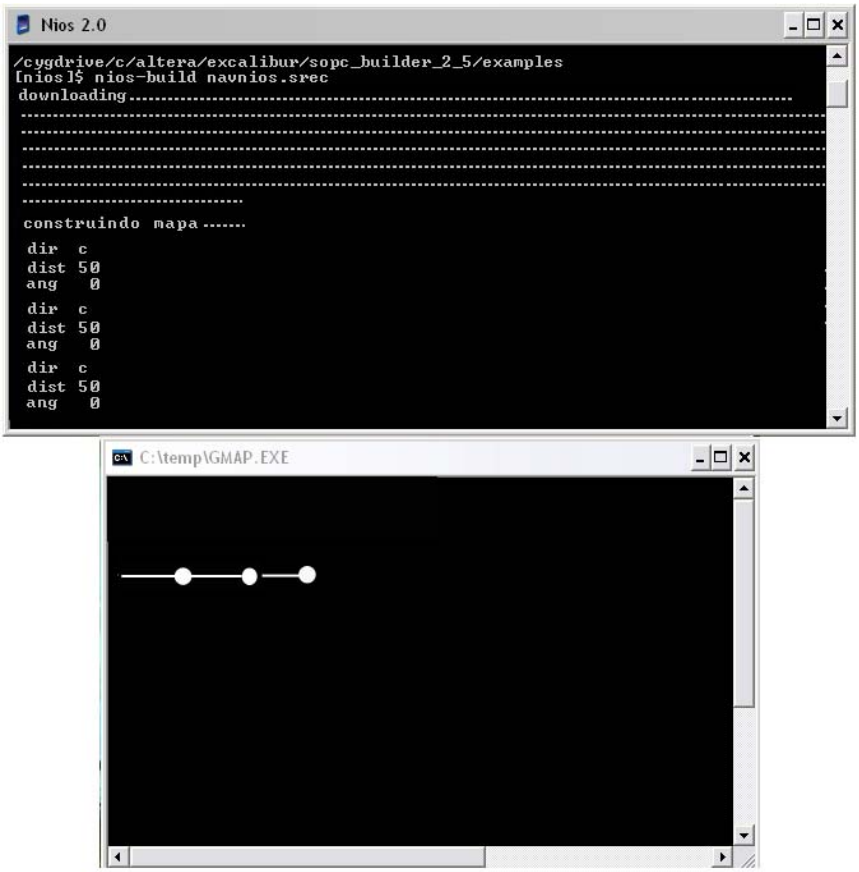

Nios 2.0
Welcome To Excalibur Nios
Version 2.0, Built Tue Jan 22 19:59:26 2002
Example nios designs can be
found in
c:/altera/excalibur/sopc_builder_2_5/examples
Try:
nios-build hello_world.c
nios-run hello_world.srec
within one of the sdk subdirectories.
(You may add a startup script: c:/altera/excalibur/sopc_builder_2_5/user.bashrc)
/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ nios-build navnios.srec
C:\temp\GMAP.EXE
Grafico Mapa Topologico
Pressione Qualquer Tecla

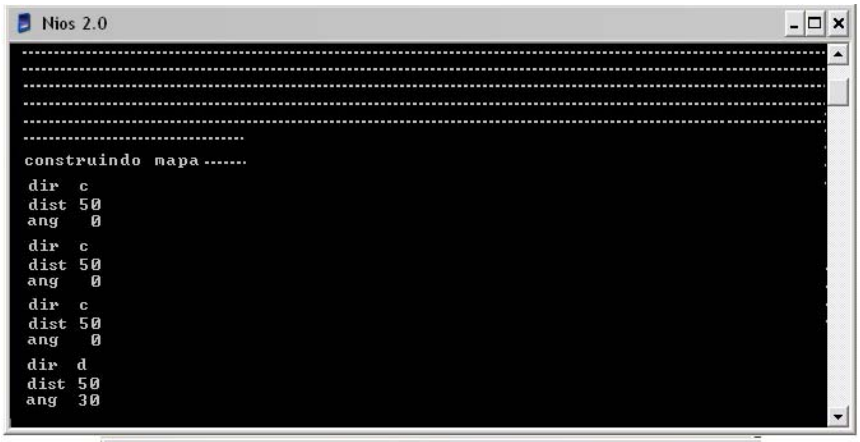

Nios 2.0
Welcome To Excalibur Nios
Version 2.0, Built Tue Jan 22 19:59:26 2002
Example nios designs can be
found in
c:/altera/excalibur/sopc_builder_2_5/examples
Try:
nios-build hello_world.c
nios-run hello_world.srec
within one of the sdk subdirectories.
(You may add a startup script: c:/altera/excalibur/sopc_builder_2_5/user.bashrc)
/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ nios-build navnios.srec
downloading
C:\temp\GMAP.EXE
Grafico Mapa Topologico
Pressione Qualquer Tecla

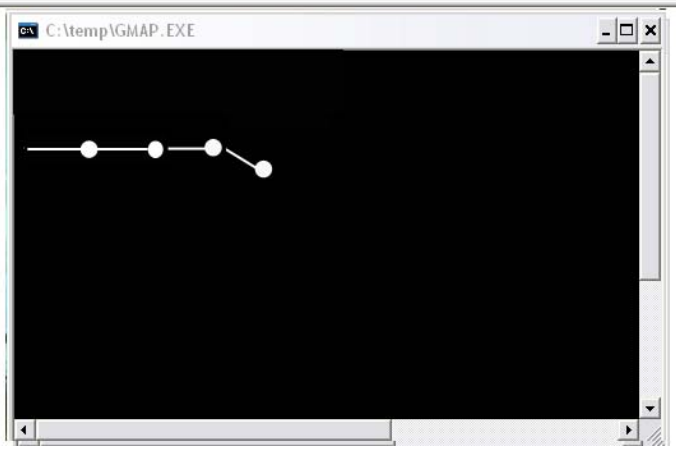

Nios 2.0
Try:
nios-build hello_world.c
nios-run hello_world.srec
within one of the sdk subdirectories.
(You may add a startup script: c:/altera/excalibur/sopc_builder_2_5/user.bashrc)
/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ nios-build navnios.srec
downloading
construindo mapa .....
[nios]$
C:\temp\GMAP.EXE
Grafico Mapa Topologico
Pressione Qualquer Tecla

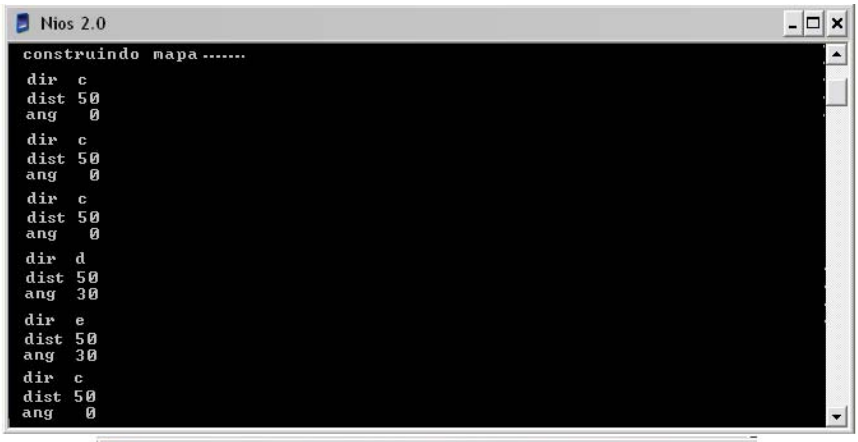

Nios 2.0
Try:
nios-build hello_world.c
nios-run hello_world.srec
within one of the sdk subdirectories.
(You may add a startup script: c:/altera/excalibur/sopc_builder_2_5/user.bashrc)
/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ nios-build navnios.srec
downloading
construindo mapa .....
dir c
dist 50
ang 0
C:\temp\GMAP.EXE

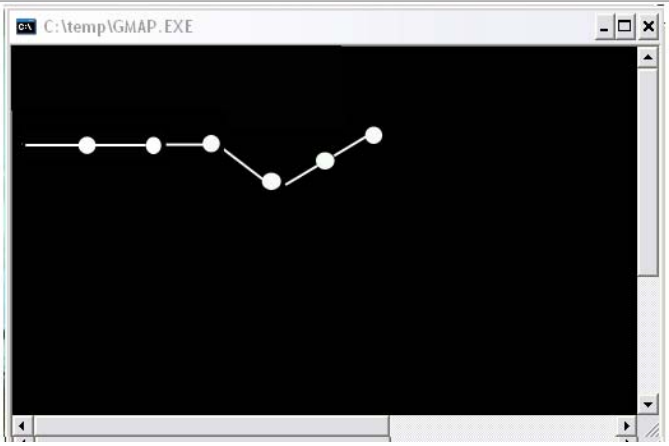

Nios 2.0
<You may add a startup script: c:/altera/excalibur/sopc_builder_2_5/user.bashrc>
/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ nios-build navnios.srec
downloading
construindo mapa .....
dir c
dist 50
ang 0
dir c
dist 50
ang 0
C:\temp\GMAP.EXE


Nios 2.0
/cygdrive/c/altera/excalibur/sopc_builder_2_5/examples
[nios]$ nios-build navnios.srec
downloading
construindo mapa .....
dir c
dist 50
ang 0
dir c
dist 50
ang 0
dir c
dist 50
ang 0
C:\temp\GMAP.EXE

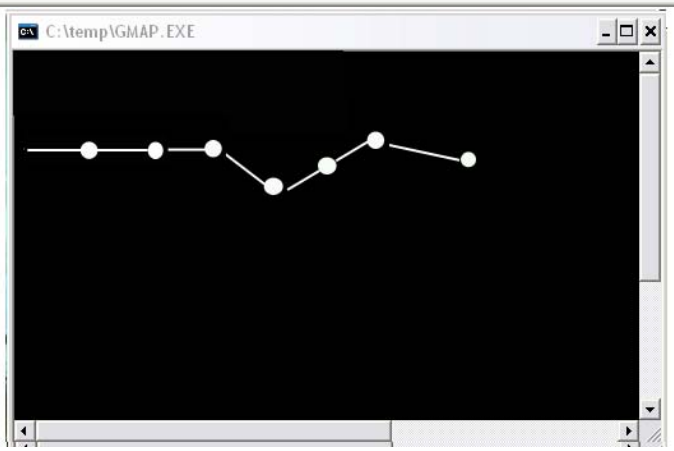

Nios 2.0
construindo mapa.....
dir c
dist 50
ang 0
dir c
dist 50
ang 0
dir c
dist 50
ang 0
dir d
dist 50
ang 30


C:\temp\GMAP.EXE

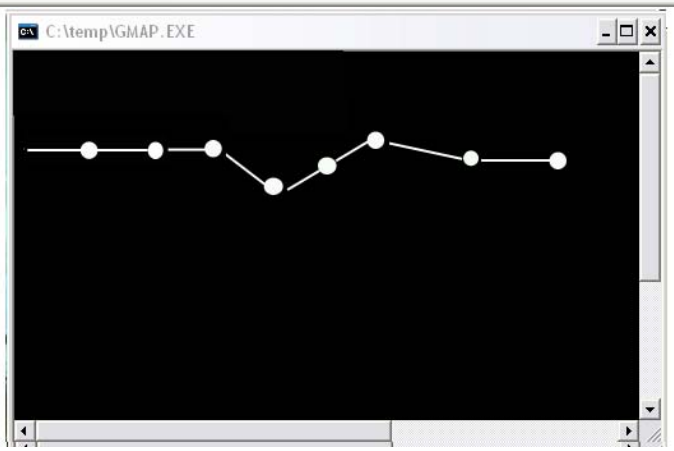

Nios 2.0
construindo mapa.....
dir c
dist 50
ang 0
dir c
dist 50
ang 0
dir c
dist 50
ang 0
dir d
dist 50
ang 30
dir e
dist 50
ang 30
dir c
dist 50
ang 0


C:\temp\GMAP.EXE


Nios 2.0
ang 0
dir c
dist 50
ang 0
dir c
dist 50
ang 0
dir d
dist 50
ang 30
dir e
dist 50
ang 30
dir c
dist 50
ang 0
dir d
dist 50
ang 30


C:\temp\GMAP.EXE


Nios 2.0
dir c
dist 50
ang 0
dir d
dist 50
ang 30
dir e
dist 50
ang 30
dir c
dist 50
ang 0
dir d
dist 50
ang 30
dir c
dist 50
ang 0


C:\temp\GMAP.EXE

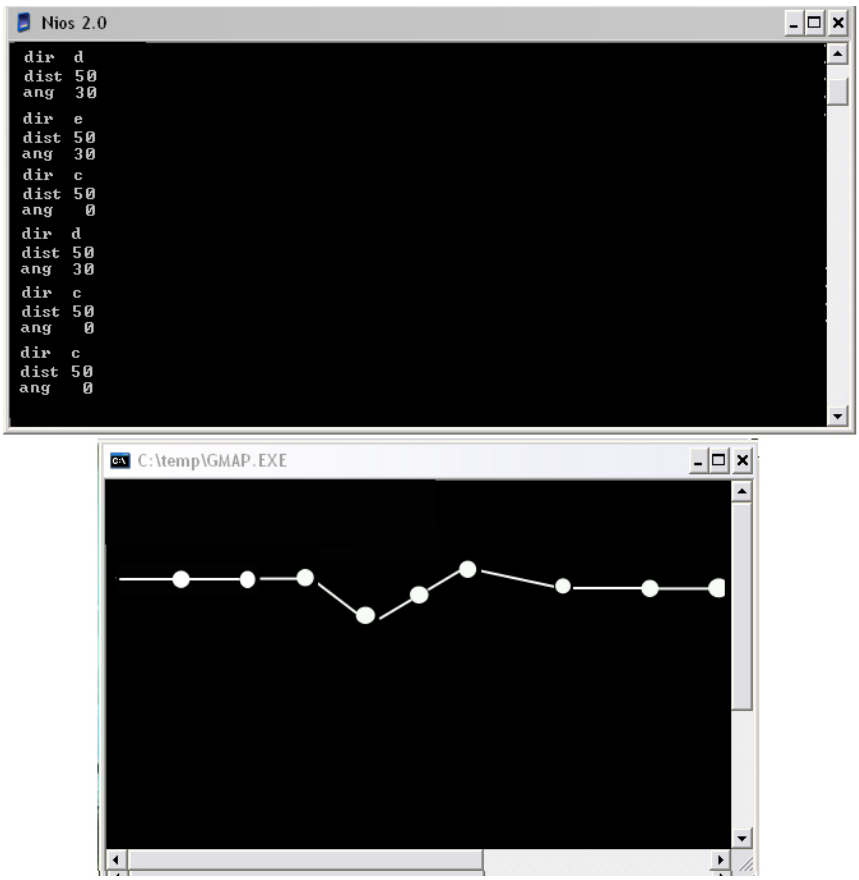


Figura 27 - Validação da Construção do Mapa Topológico com o NIOS

Concluído os testes referentes ao funcionamento e validação do sistema nos dois ambientes, Saphira e Excalibur, sendo que os resultados apresentados são satisfatórios, será apresentada a seguir uma análise temporal referente aos tempos retirados dos dois ambientes.

A primeira análise de temporal a ser descrita será a análise do ambiente Saphira. Antes de descrever sua análise temporal, não podemos deixar de reforçar as características temporais utilizadas para a execução e funcionamento deste ambiente, sendo que como descrito no Capítulo 5, este ambiente simula um sistema operacional próprio o qual possui uma latência peculiar ao seu funcionamento (100ms), para maiores detalhes consultar o Capítulo 5.

Já no sistema Excalibur, como descrito no Item 4.4, é um ambiente de construção de sistemas no chip, ou seja, esse ambiente é utilizado para a construção de sistemas diretamente em hardware sem nenhuma interface de ligação entre software e hardware. Com isso e pelo fato de estar sendo utilizado todo o potencial do hardware somente para a execução do sistema de mapeamento, tem-se um ganho de tempo de execução considerável em comparação com o sistema operacional do Saphira.

Para uma melhor visualização dos resultados retirados dessas das execuções dos dois ambientes estão descritos na tabela a seguir.

Sistema	Tempo de execução do mapeador (ms)	Tempo adicional gasto com sensores (ms)	Total (ms)
Saphira em Pentium III 800 Mhz com 1 Gb RAM e 20Gb HD	100	60	160
NIOS (Excalibur) + Hardware Dedicado (Rede Neural Artificial) (NIOS 33.3Mhz com 512Kb Ram e 1Mb de Memória Flash)	15	0	15

Tabela 1 - Tabela de Tempo de Execução.

Analisando essa tabela percebe-se que a diferença entre os ambientes esta a nível superior a dez vezes, ou seja, o software rodando no ambiente Excalibur é dez vezes mais veloz que o mesmo software rodando no ambiente Saphira. Porém essa grande diferença aplicada no software ocorre pelo fato do Saphira ter seu próprio S.O., como dito anteriormente, e com isso esse tempo de execução aumenta em grande proporção. Porém o tempo descrito como tempo adicional que é utilizado pelo ambiente Saphira é

referente à sincronização entre a leitura emulada dos sensores, o filtro de cada sensor, a tomada de decisão de qual o melhor caminho a percorrer. A navegação e as exeucacao de cada ação estão ligadas ao contexto geral do sistema no Saphira.

Este adicional não ocorre no processador NIOS pelo seguinte motivo: o kit Excalibur (Processador NIOS, Software Quartus e placa de desenvolvimento) não esta utilizando entradas para os sensores e por isso foram emulados os sensores dentro da própria placa de desenvolvimento do Excalibur. Não foi considerado o tempo de emulação dos sensores, pois esse tempo não é significativo na execução final do software.

Apesar deste detalhe, foi concluído que mesmo não estando o sistema totalmente conectado com os periféricos externos (sensores), esse kit de desenvolvimento se mostrou muito versátil pela sua maleabilidade tanto no desenvolvimento quanto na reconfiguração. Com isso torna-se possível à implementação de diversos algoritmos, que hoje são implementados na maioria das vezes em máquina de propósito geral IBM-PC com um ganho de desempenho melhorando a interoperabilidade desses sistemas.

Capítulo 7 - Conclusões e Dificuldades

No contexto de aprendizado com o desenvolvimento do sistema de mapeamento topológico houve diversos fatores que serviram como forma de crescimento, tanto no campo de estudo e análise de um objetivo a ser alcançado como na melhoria de nossas idéias iniciais.

Já no processo de desenvolvimento, pode-se destacar três fases principais que foram de grande importância para o desenvolvimento de todo o trabalho.

A primeira fase destaca-se pelo estudo e análise do princípio de orientação sobre os sistemas de navegação diversos, dentre eles o sistema de navegação e mapeamento baseado em mapas topológicos.

Nesta fase ainda, foi realizado o desenvolvimento do sistema de mapeamento topológico baseado nos estudos realizados, juntamente com sua validação em um sistema de emulação de um hardware de robôs móveis (Saphira), ilustrado na Figura 26. Esse sistema de simulação de robôs (Saphira) foi cedido pelo Laboratório de Inteligência Computacional (LABIC), pois nesta primeira fase em que o sistema de mapeamento foi construído precisava ser validado em algum sistema robótico existente e para comprovar seu funcionamento era necessário utilizar este sistema em uma arquitetura funcional.

Após essa fase foi observada a variação sobre as diversas formas de implementação desse sistema em diversos tipos de simuladores para robôs e suas respectivas arquiteturas.

As dificuldades encontradas na fase inicial foram analisar e decidir qual a melhor forma de implementação do sistema de mapeamento, acrescida da compreensão sobre o funcionamento do sistema Saphira e do robô Pioneer. Porém após várias leituras sobre o assunto conseguiu-se descobrir um padrão de implementação.

Após a decisão de implementação foi analisado o funcionamento do Saphira e do Pioneer onde houve uma pesquisa que implementou diversos programas para averiguar o correto funcionamento juntamente com os problemas apresentados pelo sistema e seu simulador.

Na segunda fase, depois de validado o software de mapeamento em um ambiente confiável, destaca-se o fato da reestruturação do sistema implementado no Saphira para o sistema Excalibur.

A princípio esta reestruturação do Saphira para o Excalibur apresentou-se simples, pois foi necessário apenas a conversão de linguagens de programação Microsoft Visual C++ do Saphira para GNUPro do processador NIOS.

Essa conversão obteve sucesso, porém havia o problema da simulação dos sonares, isto porque a idéia inicial era de que fosse implementado o sistema de mapeamento no processador NIOS, juntamente com o auxílio do simulador Saphira, onde o Saphira forneceria as informações necessárias sobre os sonares. Com isso, após a finalização da reestruturação, foi desenvolvida uma interface entre o ambiente Excalibur e o Saphira.

A interface tinha o seguinte propósito: realizar a comunicação entre o ambiente Saphira e o Excalibur, pois o sistema de mapeamento não tem a capacidade de simular leituras de sensores e por isso a interface tem o objetivo de capturar as simulações dos sensores do Saphira e transpor essas leituras para o Excalibur, com a finalidade de gerar um mapa coerente com as posições do mapa que está no Saphira.

Porém houve vários problemas existentes nessa interface. Antes de detalhar os problemas encontrados no funcionamento desta interface, será analisado como a interface interligava os dois ambientes.

Primeiramente, a interface mantinha comunicação com o Saphira através de arquivos binários cuja interface fazia as requisições ao Saphira e o Saphira respondia para esta interface através de um arquivo binário. Na comunicação com o NIOS era realizada da seguinte maneira: a interface mantinha comunicação com o processador NIOS através de uma linha serial a qual o Excalibur (placa de desenvolvimento) estava conectada ao seu host (IBM-PC). Utilizando esta linha serial a interface enviava a leitura

dos sonares as quais foram enviadas através do Saphira. Bem esse é o funcionamento da interface Saphira/NIOS.

O principal problema referente a essa interface é que enquanto o Excalibur estiver conectado ao seu host a comunicação serial fica totalmente ativa e direcionada ao Excalibur, isto porque o software desenvolvido para o NIOS necessita de um host para a realização da cópia do programa do host para o Excalibur. E o Excalibur utiliza o host para imprimir os resultados da execução do software do NIOS na tela.

Sendo assim, quando a interface tentava alocar o recurso da linha serial do host, esta linha estava completamente ocupada pelo processo do Excalibur que realizava esta operação, chamado de germs. Pelo fato da placa deste kit possuir apenas uma saída serial, como forma de comunicação, não foi possível a integração entre o simulador e esta interface. Contudo, com os novos kits de desenvolvimento que já dispõem de duas saídas seriais e uma saída paralela, torna-se viável e objetivo o funcionamento desta interface.

E na última fase deste projeto, tem-se a construção do hardware dedicado que especifica o funcionamento de uma rede neural artificial descrita no item 6.3.1.

Não houve maiores problemas nesta fase, pois o desenvolvimento desta rede neural foi estruturado primeiramente como um software de rede neural artificial. Após os testes e aprendizados realizados, os valores do aprendizado da rede e sua estrutura foram convertidos em circuitos lógicos os quais puderam ser implementados com o auxilio do software Quartus do kit do Excalibur.

Com essa implementação em hardware e a conexão com o processador NIOS, foi desenvolvido as sub-rotinas de controle desta rede, dentro do sistema de navegação. Esta sub-rotina foi desenvolvida por último porque quando o hardware é desenvolvido através do Quartus, o próprio software de desenvolvimento endereça todos os componentes de hardware existentes para o controle de I/O.

Após todo o trabalho de pesquisa, desenvolvimento, validação, testes e análises temporais realizados, conclui-se que com o desempenho de ordem 10, o objetivo principal do trabalho foi atingido, pois foi possível migrar todo o software desenvolvido para uma FPGA e deste modo contribuir de maneira significativa para o projeto ARMOSH.

Capítulo 8 - Trabalhos Futuros

Com os resultados obtidos nesta pesquisa, torna possível e viável a implementação de um sistema de navegação, baseado em mapas topológicos, juntamente com outros sistemas que podem auxiliar o sistema primário (topológico). Inclusive uma idéia atraente é utilizar a reconfigurabilidade de uma FPGA e desenvolver um sistema híbrido e dinâmico, ou seja, o sistema possuirá diversas formas de navegação e o próprio sistema irá identificar qual o melhor método a ser utilizado no momento.

Uma melhoria desta pesquisa seria desenvolver um mapa topológico com a junção de figuras que é proposto em [GAS2000], sendo implementado em um hardware reconfigurável, no qual o sistema de navegação pode utilizar essas imagens ligadas com o mapa para identificar o posicionamento global do sistema em relação à posição que se encontra. E, deste modo, quando esse recurso não for mais necessário ele será desativado pela FPGA.

Apêndice A

Neste apêndice está descrito o programa principal de navegação.

```
/*****************************************************************************************
/ Arquivo: NavNIOS.c
/ Titulo:       Modulo Principal do Mapa Topologico
/ Autor: Jean M.Scatena
/ Dissertação de Mestrado: ICMC - USP - Sao Carlos - BR
/  Criação: 07/01/2002
/  Revisao: 2.2
/  Data Ultima Revisao: 14/09/2002
/ Obs.: Reestruturacao do Codigo de Navegacao do Modulo Navegsaph.c para o NIOS
/ ** Leitura dos sonares simuladas via NIOS e impressas via shell do GNU (Germs) **
/*****************************************************************************************
#include "Process.h"
#include <nios.h>
////////////////////////////////////////////////////////////
//*                                Corpo  do  Programa                                  */
////////////////////////////////////////////////////////////
int main(void)
{
     int i=1;
     for(i=0; i < RS; i++)
     {
        Capt_Values(i);
        Send_RN(i);
        Print_RN_IN();
        RNavig();
        IncludeMapPoint(pos, colisao, DISTANCE);
        MakeMap();
     }
     Print_Path();
return 0;};
```

Apêndice B

Neste apêndice é descritas a biblioteca com as estruturas utilizadas no código.

```
/*************************************************************************************************
/ Arquivo: Nstruct.h
/ Titulo:Modulo de Estrutura Desenvolvida para Navegação Primaria e a Construção do Mapa
/ Autor: Jean M.Scatena
/ Dissertação de Mestrado: ICMC - USP – São Carlos - BR
/  Criação: 15/09/2001
/  Revisão: 2.3.6
/  Data Ultima Revisão:  08/09/2002
/  Inclusão das Estruturas dos Barramentos Incorporados no NIOS para Utilização da
/   Rede  Neural em hardware dedicado.
/*****************************************************/
#include <nios.h>
/*************************************************************************************/
        /* Define as distancias minimas e max. de segurança */
#define MAX 400
#define MINIM 350
/*************************************************************************************/
          /* Define o quanto o robo percorre por seg. */
#define DISTANCE 50
/*************************************************************************************/
        /* Define o angulo de rotacao para os sonares */
#define ANG   30
#define NORM   0
#define NANG -30
#define FATOR 858993
```

```
/*****************************************************************************/
/* Variáveis que recebem via serial os valores do simulador SAPHIRA */
#define RS 20
/* Variáveis que recebem via serial os valores do simulador SAPHIRA */
int SONAR0[RS], SONAR2[RS], SONAR3[RS], SONAR4[RS], SONAR6[RS];
/*****************************************************************************/
        /* Inclusão das estruturas da Rede Neural em hardware dedicado */
/*****************************************************************************
/ rn_in -> Saida do Nios e entrada na RN
/ rn_out -> Saida da RN e entrada no NIOS
/ select_n -> Seleciona qual o neuronio de entrada dos dados e seleciona o start p/calculo da RN
*****************************************************************************************************/
np_pio *rn_in = na_dados, *rn_out = na_rn, *select_n = na_seleciona;
/*****************************************************************************/
       /* Estrutura de Armazenamento de um Ponto no Mapa Topologico */
typedef struct SPM
{
     char side;
     int angle;
     int distance;
     float x,y;
     struct SPM *npoint;
}SPOINTMAP;
/*****************************************************************************/
```

Apêndice C

Neste apêndice descreve-se a biblioteca que realiza todas as operações de navegação.

```
/*************************************************************************************************
/Arquivo: Process.h
/ Titulo:       Modulo de Navegação Primaria com  Construção do Mapa Topológico
/ Autor: Jean M. Scatena
/ Dissertação de Mestrado: ICMC - USP - São Carlos – BR
/  Criação: 21/01/2002
/  Revisão: 2.0
/  Data Ultima Revisão: 10/09/2002
/  Obs.: Reestruturação do Código de Navegação e Colisão para o NIOS
/    ** Retirada dos modulos de navegacao via serial **
/*************************************************/
#include "Nstruct.h"
#include "stdlib.h"
#include <malloc.h>
#include <nios.h>

////////////////////////////////////////////////////////////
                /* Variaveis globais de Uso Geral */
SPOINTMAP *map= NULL, *aux= NULL, *aux_tm= NULL, *last= NULL;
int colisao=NORM, pointmap=0;
char pos='C';
////////////////////////////////////////////////////////////
                /*      Subrotinas    */
// Subrotinas de Comunicacao NIOS
void Start(void);
void Capt_Values(int i);
void Print_Com(void);
// Subrotinas de Navegacao
void Send_RN(int i);
void Rec_RN(int i);
void NoCollision(int s);
```

```
// Subrotinas de Mapeamento
void IncludeMapPoint( char sid, int ang, int dist);
void MakeMap(void);
void Print_Path(void);
////////////////////////////////////////////////////////////////
/*          Módulo de Comunicação NIOS                        */
////////////////////////////////////////////////////////////////
        /* Sub-rotina de Inicializacao */
void Start()
{
     int i=1;
     for(i=0; i < RS; i++)
     {
        Capt_Values(i);
        Send_RN(i);
        Rec_RN(i);
        IncludeMapPoint(pos, colisao, DISTANCE);
        MakeMap();
     }
     Print_Path();
}
////////////////////////////////////////////////////////////////
/*       Captura Valor dos SONARes via shell do GNU Pro       */
void Capt_Values(int i)
{
    SONAR0[i] = (int) ((rand()) / FATOR);
    SONAR2[i] = (int) ((rand()) / FATOR);
    SONAR3[i] = (int) ((rand()) / FATOR);
    SONAR4[i] = (int) ((rand()) / FATOR);
    SONAR6[i] = (int) ((rand()) / FATOR);
    printf(" leitura do sonar 0 %d \n", SONAR0[i]);
    printf(" leitura do sonar 2 %d \n", SONAR2[i]);
    printf(" leitura do sonar 3 %d \n", SONAR3[i]);
    printf(" leitura do sonar 4 %d \n", SONAR4[i]);
    printf(" leitura do sonar 6 %d \n", SONAR6[i]);
}
```

```
////////////////////////////////////////////////////////////////
/* Imprime Comandos do Navegador que esta sendo processado no NIOS */
void Print_Com()
{
     printf("A direcao decidida é %c", pos);
     printf("A distancia percorrida é %d", DISTANCE);
     printf("O angulo de direcionamento %d", colisao);
     switch(pos)
     {
       case 'L': {
              printf("A leitura escolhida do sonar foi do Sonar 2");
              break;
            }
       case 'C': {
              printf("A leitura escolhida do sonar foi do Sonar 3");
              break;
            }
       case 'R': {
              printf("A leitura escolhida do sonar foi do Sonar 4");
              break;
            }
     };
}

////////////////////////////////////////////////////////////////
/*              Módulos de Navegação                    */
////////////////////////////////////////////////////////////////
/* Envia informacoes para a RN */
/************************************
/** select_n = 100 -> Start
/**         001 -> Neuronio Left 1
/**         011 -> Neuronio Left 2
/**         101 -> Neuronio Center
/**         111 -> Neuronio Right 1
/**         000 -> Neuronio Right 2
************************************/
```

```
void Send_RN(int i)
{
  // Envia valores do Sonar 0 para select_n Final
  rn_in -> np_piodirection = 1;
  rn_in -> np_piodata = SONAR0[i];
  // Seleciona qual o Neuronio da Camada 1 ira receber a informacao
  select_n -> np_piodirection = 1;
  select_n -> np_piodata = 1;
  rn_in -> np_piodirection = 1;
  rn_in -> np_piodata = SONAR2[i];

  select_n -> np_piodirection = 1;
  select_n -> np_piodata = 3;
  rn_in -> np_piodirection = 1;
  rn_in -> np_piodata = SONAR3[i];

  select_n -> np_piodirection = 1;
  select_n -> np_piodata = 5;
  rn_in -> np_piodirection = 1;
  rn_in -> np_piodata = SONAR4[i];

  select_n -> np_piodirection = 1;
  select_n -> np_piodata = 7;
  rn_in -> np_piodirection = 1;
  rn_in -> np_piodata = SONAR6[i];

  select_n -> np_piodirection = 1;
  select_n -> np_piodata = 0;

  // Seleciona o Start para a execucao da rede
  select_n -> np_piodirection = 1;
  select_n -> np_piodata = 4;
}
```

```
////////////////////////////////////////////////////////////////
/* Recebe informacoes da RN */
/***********************************
/** Saida RN = 0 x x -> Centro   -> > 1 e < 7
/**          0 0 x -> Direita  -> 1
/**          1 1 1 -> Esquerda -> 7
************************************/
void Rec_RN(int i)
{
  int result =0;
  rn_in -> np_piodirection = 0;
  result = rn_in -> np_piodata;
  switch(result)
  {
   case 1:{
          if(colisao == NORM)
            colisao = NANG;
          else if(colisao == ANG)
             colisao = NORM;
          pos = 'R';
          break;
         }
   case 2:
   case 3:
   case 4:
   case 5:{
          pos = 'C';
           break;
         }
   case 7:{
          if(colisao == NORM)
           colisao = ANG;
          else if(colisao == NANG)
             colisao = NORM;
          pos = 'L';
           break;   }
```

```
   default: NoColision(i);
  }
}
////////////////////////////////////////////////////////////
        /* Evita Colisao com as paredes e/ou objetos */
void NoCollision(int s)
{
    if((SONAR0[s] < MINIM) || (SONAR2[s] < MAX))
    {
        pos = 'R';
        colisao = -15;
    }
    else
        if((SONAR6[s] < MINIM) || (SONAR4[s] < MAX))
        {
                pos = 'L';
                colisao = 15;
        }
        else
                if(SONAR3[s] < MAX)
                        if(SONAR0[s] >= SONAR6[s])
                        {
                                pos = 'L';
                                colisao = 90;
                        }
                        else
                        {
                                pos = 'R';
                                colisao = -90;
                        }
}
////////////////////////////////////////////////////////////
```

Referências

[ALT3W] Altera Corp.; *Nios Embedded Processor Development Board*; April 2002; Data Sheet, ver. 2.1; in http://www.altera.com/literature/ds/ds_nios_devboard.pdf;

[ALT2002] Altera Corporation, http://www.altera.com; visitada em 5 de setembro de 2002.

[ARA2000] Aragão, A. C. O. S., Uma Arquitetura Reconfigurável Dinamicamente Dedicada ao Controle de Robôs Móveis, Dissertação de Mestrado – ICMC USP.

[ARA2000B] Aragão, A. C. O. S.; Marques, E., A Tecnologia FPGA, Relatório Técnico no. 60 – ICMC USP.

[ARM2002] Romero, R. A. F., Marques, E., ARMOSH um robô auto-reconfigurável, Projeto Fapesp – ICMC USP, julho, 2002.

[BOY2002] Boyoon J.; Sukhatme, S. G.; A Region-based Approach for Cooperative Multi-target Tracking in a Structured Environment; Robotics Laboratory, University of Southern California; 2002; URL:http://robotics.usc.edu/submissions.html, visitada em agosto 2002.

[BOR96] Borenstein J., Everett R.H., Feng L., Where am I? Sensors and Methods for Mobile Robot Positioning, The University of Michigan, April 1.996

[CAR99] Cardoso, J. M. P.; Vestias M. P.; Architectures and Compilers to Support Reconfigurable Computing; ACM; 1999; in URL:http://ww.acm.org/crossroads/xrds5-3/rcconcept.html, visitada em março 2001

[D&R2001] D&R Industries Articles, URL:

Http://www.us.design-reuse.com/NEWS/papers.html, visitada em março de 2001

[DEH] DeHon A., Wawrzynek J., Reconfigurable Computing: What, Why, and Implications for Design Automation, Computer Science Division, Uiversity of California at Berkeley, Berkeley – CA 94720-1776

[DEM96] Demian V., Desprez F., Paugan-Moisy H., Pourzandi M., "Paralel Implementation of RBF Neural Networks", Ecole Normale Supérieure de Lyon, Research Report No. 96-11, 1996.

[DUC2000] Duckett T., Saffloti A., Building Globally Consistent Gridmaps from Topologies, Center for Applied Autonomous Sensor Systems Department of Technology, The University Örebro, SE-70182, Örebro, Sweden..

[DUC99] Duckett T., Nehmzow U., Exploration of Unknown Environments Using a Compass, Topological Maps and Neural Network, Department of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom.

[DUC99A] Duckett T., Nehmzow U., Self-localisation and Autonomous Navigation by a Mobile Robot, Department of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom.

[DUC98] Duckett T., Nehmzow U., Mobile Robot Self-localization and Measurement of Performance in Middle-scale Environment, Department of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom.

[GAS2000] Gaspar J.; Winters N.; Victor, S.J.; Vision-based Navigation and Environment Representation with an Omni-directional Camera; Instituto de Sistemas de Robótica, Instituto Superior Técnico and Department of Computer Science of University of Dublin; 2000; URL: http://viriato.isr.ist.utl.pt/~jasv/vislab/publications/ps/00-IEEETra.pdf ; visitada em agosto 2002

[HAU] Hauser J. R., Wawrzynek J., Garp: A MIPS Processor with a Reconfigurable Coprocessor, University of California, Berkeley.

[HAU97A] Hauser J. R., The Garp Architeture. University of California at Berkeley Department of Electrical Engineering and Computer Sciencies, October 1997.

[HAU97B] Haug, G.; Rosenstiel W.; Reconfigurable Hardware as Shared Resource for Parallel Threads; Proc. 5th Annual IEEE Symposium on Custom Computing Machines; IEEE Computer Society Press; 1997;

[JOC96] Jochem T., Pomerleau D., Life in the Fast Line, AI Magazine, 17, 2, 1996, pp. 11-50.

[KEI98] Kotay K., Rus D., Self-Reconfigurable Robots for Navigation and Manipulation, Dartmouth Robotics Laboratory, Department of Computer Science Dartmouth, Hanover, NH 03755, USA. 1.998

[KEI97] Kotay K., Rus D., Vona M., McGray C., The Self-Reconfigurable Robots Molecule, Dartmouth Robotics Laboratory, Department of Computer Science Dartmouth, Hanover, NH 03755, USA. October, 15, 1.997.

[KOR99] Kortenkamp D., Huber M., Mobile Robot Exploration and Navigation of Indoor Spaces Using Sonar and Vision, The University of Michigan, Artificial Intelligence Laboratory, Ann Arbor, MI 48109.

[KÜN2001] Künfas Peep, Learning State Machines in the Robot Moving, Institute of Cybernetics at Tallinn Technical University, 2001; http://greta.cs.ioc.ee/~peep/

[LYS93] Lysaght P.; Dunlop J.; Dynamic ReconFiguration of Field Programmable Gate Arrays; Department of Electronic and Electrical Engineering; University of Strathclyde; 1993;

[MIL95] Millan J., Torras C., Efficient Reinforcement Learning of Navigation Strategies in an Autonomous Robot, in Graefe V., ed. Intelligent Robots and Systems, Elsevier Science B.V., 1995, pp.185-199.

[MIT2001] Mitrovic D, Supporting Navigation Decision Making by Learning Driving Patterns. URL: http://www.cosc.canterbury.ac.nz/~dmi24/proposal.htm, visitada em março 2001

[MOE97] Moerland P., Fiesler E., "Neural Network Adaptations to Hardware Implementations", Handbook of Neural Computation E1.2:1-13 Institute of Physics Publishing and Oxford University Publishing, New York, 1997.

[MOL2000] Molz R. F., Engel P. M., Moraes F. G., Torres L., Robert M., "Estudo da Viabilidade de Implementação de um Sistema de Localização e Reconhecimento de Objetos com uso de RNAs Implementadas em FPGAs", Workshop de

Computação Reconfigurável CORE2000, Marília, Brazil, pp. 226-235, 2000 (in portuguese)

[MOR2001] Moreno J.M., Cabestany J., Cantó J. F., Insener J.M., The Role of Dynamic Reconfiguration for Implementing Artificial Neural Networks Models in Programmable Hardware, Dept. of Electronic Engineering, University of Catalunya.

[NAB98] Nabbe B., A Language for Reconfigurable Robot Control, Departmente of Computer Science, University of Amsterdam, September 1.998.

[NEH99] Nehmzow U., Duckett T., Knowing your Place in Real World Environments, Department of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom.

[PER96] Pérez-Uribe A., Sanchez E., FPGA Implementation of an Adaptable-Size Neural Network, Proc. of VI International Conference on Artificial Neural Networks ICANN96, pp.29-31 , 1996.

[PON98] Braga A. P., Carvalho A. C. P. L. F., Ludermir T. B., Fundamentos de Redes Neurais Artificiais, XI Escola brasileira de computação, 2 de maio de 1998

[RON2002] Ron W., The Constantly shifting promise of reconfigurability, EE Times, September 11, 2002.

[RVL2001] Robot Vision Lab, School of Electrical and Computer Engineering, Purdue University. URL: http://rvl1.ecn.purdue.edu/, visitada abril 2001

[SAP1997] Saphira Software Manual, Saphira version 6.1 - 1997.

[SBP2001] Star Bridge Systems, Overview of SBS´s Reconfigurable Computing Technology, URL:http://www.starbridgessystems.com/tech-over.html – visitado em janeiro de 2001.

[SMI94] Razdan R., Brace K., D.Smith M., PRISC Software Acceleration Techniques. In Proceedings of the IEEE International Conference on Computer Design, Oct. 1994, pp. 145-149.

[TAN2001] Tanurhan, Y., SOC Design Using Embedded Cores, Director of Actel Corporation, Sunnyvale, California, USA, March, 2001.

[TAY96] Taylor M. B., Design Decisions in the Implementation of a Raw Architecture Workstation, Dartmouth College 1996.

[THRUN96] Thrun S., Bücken A. Learning Maps for Indoor Mobile Robot Navigation, Technical Report CMU-CS-96-121, Carnagie Mellon University, School of Computer Science, Pittsburgh, PA 15213, April 1996.

[ULR2000] Ulrich I., Nourbakhsh I., Appearance-Base Place Recognition for Topological Localization, The Robotics Institute, Carnegie Mellon University Pittsburgh, PA 15213, April 2000.

[VAL2001] Vale,Alberto; Simões,J.; Machado J.; Lima, P., Multi-sensor Navigation without an a priori Map; Instituto de Sistemas de Robótica, Instituto Superior Técnico; 2001; URL: http://lrm.isr.ist.utl.pt/publications.html, visitada em agosto de 2002.

[VIL97] Villasenor J., Mangione-Smith W. H., Configurable Computing, Scientific American, Junho de 1997.

[YAM98] Yamauchi B., Schultz A., Adams W., Mobile Robot Exploration and Map-Building with Continuos Localization, Navy Center for Applied Research in Artificial Inteligence, Naval Research Laboratory, Washigton, DC 20375-5337, 1.998

[WIR95] Wirthlin M. J., Hutchings B. L., A Dynamic Instruction Set Computer. In Proceedings of the 4th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95), Napa Valley, California, USA, April 19-21, 1995, pp. 99-107.