LUCIANO ROTTAVA DA SILVA

ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT

FLORIANÓPOLIS
2003

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

LUCIANO ROTTAVA DA SILVA

Florianópolis, março de 2003.

ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT

Luciano Rottava da Silva

‘Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.’

Guilherme Bittencourt, Dr.
Orientador

Edson Roberto De Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Guilherme Bittencourt, Dr.
Presidente

Marcelo Ricardo Stemmer, Dr.

Mauro Roisenberg, Dr.

Raul Guenther, Dr.

Aos meus pais e à toda minha família…

AGRADECIMENTOS

Agradeço primeiramente ao meu orientador Prof. Guilherme Bittencourt pela oportunidade e confiança depositadas desde os tempos de iniciação científica. Ao Prof. Augusto Loureiro da Costa pelas importantes sugestões, principalmente durante as fases iniciais deste trabalho. Hoje, mais do que meros orientadores acadêmicos, amigos!

À todos colegas e amigos do Departamento de Automação e Sistemas que durante estes anos todos contribuiram de alguma forma para a realização deste trabalho. Em especial aos professores Edson Roberto De Pieri, Alexandre Trofino Neto, Jean-Marie Farines, Joni da Silva Fraga, Rômulo Silva de Oliveira, Eugênio Castelan, Marcelo Stemmer e Ricardo Rabelo.

À Carlos Brandão e Jerusa Marchi pelas estimulantes conversas regadas a muito chá e café e embaladas por sons esotéricos. A Eder Mateus pelas estimulantes conversas sobre IA e o Projeto UFSC-Team. A Rafael Obelheiro, Fabiano Bachmann e Hélio Loureiro pelos longos "papos" sobre o excêntrico mundo da computação. À Paulo Mafra pelas sugestões dadas aos meus algoritmos e programas de "engenheiro".

Também, Isabel Tonin, Frederico Freitas, Carlos Montez, Rodrigo Sumar, Priscila Martins, Fábio de la Rocha, Alysson Neves e tantas outras pessoas que hoje fazem parte da minha história. O convívio quase que diário com todos proporcionou muitos bons momentos para recordar.

Ao Bill Joy por ter inventado o excelente editor `vi`. Ainda não criaram nada tão *user-friendly* quanto. Ao Donald Knuth e Leslie Lamport por desenvolverem o TEX e LATEX, respectivamente. Não me imagino utilizando outro conjunto de ferramentas para edição de textos científicos.

Finalmente, sou grato pelo carinho dos meus pais que me suportaram (quando nem eu mesmo era capaz), apoiaram e incentivaram para que eu concluisse mais esta etapa. Obrigado!

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT

Luciano Rottava da Silva

Março/2003

Orientador: Guilherme Bittencourt
Área de Concentração: Automação e Sistemas
Palavras-chave: Inteligência Artificial, Robótica Móvel, Visão Computacional
Número de Páginas: 1 + 91

A utilização de robôs móveis autônomos em pesquisas de Inteligência Artificial despertou o interesse de muitos grupos ao redor do mundo. O futebol robótico transformou-se numa área de estudo capaz de congregar diversas outras. Trata-se de um campo riquíssimo para a pesquisa e o ensino.

Neste trabalho apresentamos, inicialmente, capítulos introdutórios nas áreas que dão suporte ao tópico futebol robótico, como Sistemas Especialistas, Sistemas Multiagentes e Lógica Nebulosa. A seguir traçamos um histórico do desenvolvimento do tema futebol de robôs. Organizações, regras, categorias, enfim, várias definições e conceitos são apresentados com o objetivo de familiarizar o leitor.

O núcleo da pesquisa é composto pela análise feita dos robôs móveis da plataforma EyeBot. Foram escritos programas com o objetivo de explorar as potencialidades destas máquinas, ou seja, todos os sub-sistemas do robô foram estudados e compreendidos. Também desenvolvemos um programa chamado *Xcameraviewer* capaz de enviar imagens do robô para uma estação de trabalho remota via enlace de rádio. A versão existente não contemplava sistemas Unices, apenas a plataforma MS-Windows.

Outros experimentos foram feitos envolvendo, principalmente, a câmera embarcada do robô. Por fim, resultados e conclusões são apresentados a fim de direcionar futuros trabalhos nesta área.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

ANALYSIS AND PROGRAMMING EYEBOT'S PLATFORM AUTONOMOUS MOBILE ROBOTS

Luciano Rottava da Silva

March/2003

Advisor: Guilherme Bittencourt
Area of Concentration: Automation and Systems
Keywords: Artificial Intelligence, Mobile Robots, Machine Vision
Number of Pages: 1 + 91

The use of autonomous mobile robots in Artificial Intelligence has been instigating the interest of several groups around the world. Robotic soccer game become an interesting area able to combine many others. It's a very rich field to develop researches and to taught concepts.

In this work we present introductory chapters in areas that form the underground of robotic soccer, like Expert Systems, Multi-agent Systems and Fuzzy Logic. Organizations, rules, categories, everything is showing and explaning to the reader.

Dissertation's core is formed by EyeBot autonomous mobile robots analysis. We have written programs to explore all robot's sub-systems. We also have developed a small aplication called *Xcameraviewer* which is able to send pictures from the robot's local camera to a remote base station via radio link. The existing version doesn't work with any UNIX version, just MS-Windows workstations.

Others experiments has been made, mainly evolving vision and communication. Finally, results and conclusions are presented as well as sugestions for future work.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas

FIRA ..... *Federation of International Robot-soccer Association*
HDT ..... *Hardware Description Table*
IA ..... Inteligência Artificial
IAD ..... Inteligência Artificial Distribuída
LHS ..... *Left Hand Side*
PSD ..... *Position Sensitive Device*
RHS ..... *Right Hand Side*
RIA ..... *Robotic Industries Association*
RNA ..... Redes Neurais Artificiais
RoboCup ..... *Robot World Cup Soccer Games and Conferences*
SDP ..... Solução Distribuída de Problemas
SE ..... Sistemas Especialistas
SMA ..... Sistemas Multiagentes

Capítulo 1

Introdução

1.1 Formulação do Problema

A robótica móvel ocupa-se de estudar e desenvolver máquinas capazes de se locomover, em geral, em ambientes não controlados, ruidosos e desconhecidos. Para atender a estes objetivos, a multidisciplinaridade faz-se necessária. Mecânica fina, elétrica/eletrônica e computação são integradas para dar suporte à esta nova e importante área (Pazos, 2002).

Os desafios são cada vez maiores. Não bastam apenas serem capazes de desviarem de obstáculos e evitarem colisões. Os robôs móveis de hoje devem ser dotados de alguma "inteligência" que lhes possibilite, por exemplo, reconhecer um dentre vários objetos. A fusão sensorial tem sido usada para tentar melhorar as respostas e com isto modelar o comportamento dos robôs. Mas somente isto não é suficiente, é preciso algo mais. Neste sentido, a Inteligência Artificial desempenha um papel fundamental. É ela a responsável por essa mudança, de simples máquinas estáticas repetidoras de tarefas a robôs totalmente autônomos capazes de tomarem suas próprias decisões.

1.2 Histórico

O interesse pela mobilidade de robôs teve início com hobistas das áreas de elétrica/eletrônica. Estes entusiastas construíram seus primeiros protótipos impulsionados pela curiosidade. Compostos a partir de peças de equipamentos como rádios, televisores e pequenos motores, tinham como inspiração os filmes de ficção científica das décadas de 60 e 70. Estes modelos também tinham em comum a imitação da forma humana (antropomórficos), a teleoperação e o total desprovimento de "inteligência" (Brooks e Flynn, 1989a; Brooks e Flynn, 1989b). Em (Everett, 1995) há uma descrição completa de vários modelos deste criativo período.

A partir do final da década de 70 e início dos anos 80, começaram a aparecer projetos sérios para aplicações como patrulhamento de ambientes, transporte de carga, segurança, exploração, entre outros, quase sempre patrocinados por órgãos militares, governamentais ou grandes empresas (Everett,

Figura 1.1: *Robart I*, *Robart II* e *Robart III* projetados pela marinha norte-americana para o patrulhamento *indoor*

2003). Percebeu-se aí um importante nicho ainda inexplorado e de grande futuro se nós pensarmos no enorme potencial econômico disto. Também alimentou o sonho de tornar realidade máquinas que o cinema imortalizou na série *Star Wars*, como o andróide *C*3*PO* e o robô *R*2*D*2.

Um dos primeiros modelos a obter sucesso foi o *Robart I*, desenvolvido pela marinha norte-americana entre os anos de 1980 e 1982. Este robô tinha como função patrulhar ambientes fechados à procura de situações indesejáveis, tais como indícios de incêndio e vestígios de intrusos (figura 1.1). Aplicações como estas foram escolhidas para demonstrar a utilidade destas funções livrando pessoas destas tarefas e para provar que não eram necessários dispositivos avançados, reduzindo a complexidade de todo sistema. Estes modelos já exibiam características como fusão sensorial e dispositivos sintetizadores de voz para anúncio destas situações indesejáveis detectadas. Também eram capazes de encontrar sozinhos para a recarga da bateria pontos previamente configurados com base na troca de sinais de rádio entre o robô e uma central posicionada previamente em algum ponto do ambiente. Os modelos seguintes, *Robart II* (1982-1992) e *III* (1992-), eram mais avançados, possuindo processamento distribuído de cada grupo de sensores, planejamento de trajetória, criação de mapas do ambiente, utilização de câmeras de vídeo para a transmissão de imagens e muitos mais.

O desenvolvimento prosseguiu com a adição de novos tipos de sensores, atuadores e programas de controle tornando-os cada vez mais autônomos e precisos na realização de suas tarefas. Também passaram a ser utilizados em outros ambiente além do terrestre. Um exemplo disto são os robôs para exploração submarina como aqueles utilizados na descoberta de naufrágios famosos e robôs para o patrulhamento aéreo, como os aviões espiões *Predator* norte-americanos usados na guerra do golfo. Ou seja, robôs permeiam praticamente todas as atividades humanas dos dias de hoje, mesmo sem nos darmos conta disso. Da limpeza de ambientes à aviação, das agências bancárias aos supermercados,

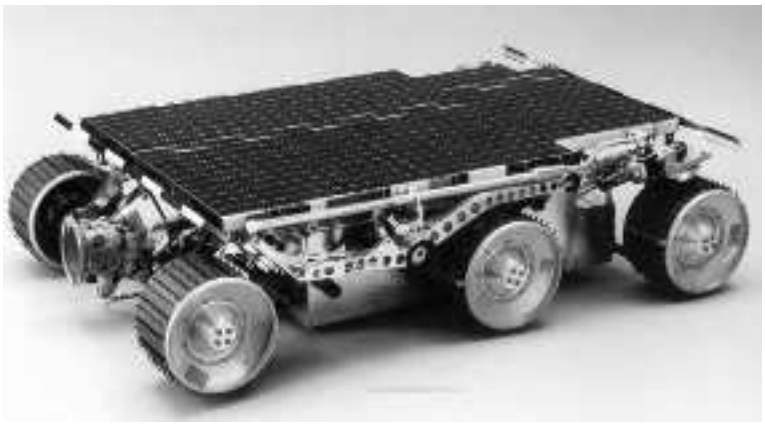

Figura 1.2: Robô Sojourner enviado para um missão em Marte

quase tudo da vida moderna possui máquinas programáveis para facilitar a vida das pessoas. A seção seguinte mostra o atual estágio de desenvolvimento destes robôs.

1.3 Estado da Arte

Existem hoje robôs extremamente avançados. A evolução nesta área é surpreendente. O caso mais famoso de projeto bem sucedido é o robô *Sojourner* (figura 1.2). Construído pelo *Jet Propulsion Laboratory (JPL)* do Instituto de Tecnologia da Califórina em parceria com a NASA e enviado a Marte[1] com o objetivo de explorar as características do terreno deste planeta e realizar experimentos científicos (Stone, 1996; Morrison e Nguyen, 1996). Esta missão foi a primeira de uma série de outras planejadas (NASA, 2002; Washington et al., 1999). Este robôs podem ser considerados o estado da arte na área de robótica móvel autônoma. Uma vez lançados e em órbita, pouco pode ser feito em caso de falhas. A viagem do *Sojourner* levou sete meses. Durante este período de tempo, tudo que os técnicos responsáveis pela missão tinham a sua disposição eram dados de telemetria. A intervenção (envio de comandos por exemplo) também era possível, apesar do longo intervalo de tempo necessário para isto, entre 6 e 41 minutos.

1.4 Aplicações

As seções anteriores nos dão uma boa noção sobre o futuro desta área. O desenvolvimento de robôs para a exploração interplanetária deixou de ser apenas uma experiência isolada. Cada vez mais projetos da agência espacial americana comtemplam a utilização de máquinas exploradoras. Além do

[1] Esta missão chamada de Mars Pathfinder teve início em 4 de dezembro de 1996 e o robô (*lander* e *rover*) pousou em solo marciano em 4 de julho de 1997.

aspecto tecnológico há, é claro, a questão financeira. Missões como a do *Sojourner* são consideradas de baixo custo se comparadas as missões tripuladas. O debate em torno da utilidade e segurança de vôos tripulados também tem contribuído para o desenvolvimento de robôs exploradores. Um exemplo disso é que a Nasa planeja enviar outro robô para Marte nos meses de maio ou junho deste ano (NASA, 2003).

O recente acidente com o ônibus espacial *Columbia* reacendeu o debate em torno da necessidade dos vôos tripulados e da própria segurança dos veículos espaciais. Este episódio terá grande impacto sobre o programa espacial e principalmente sobre a robótica móvel. Esta área tem tudo para avançar ainda mais no futuro, já que a própria agência americana admite que muitas das experiências à bordo das naves poderiam ser feitas por robôs, com menores custos e maior segurança.

Além da exploração espacial, podemos pensar em outras aplicações para esta tecnologia, tais como:

- transporte de carga em indústrias (Transrobotics, 2003)
- exploração de ambientes perigosos e/ou nocivos a seres humanos (Bares e Whittaker, 1994; NASA, 1999)
- limpeza e manutenção de ambientes (Electrolux, 2003)
- segurança e proteção (Everett, 2003)
- diversão, brinquedos (Sony, 2003)

Este último tópico, brinquedos, tem atraído a atenção principalmente de pesquisadores da empresa japonesa Sony. Sua linha de robôs *AIBO* (Sony, 2003) é um verdadeiro sucesso no Japão. Projetados para se parecerem com animais de estimação, estas máquinas estão sendo utilizadas mais para o ensino de conceitos básicos de programação e robótica às crianças do que propriamente para a diversão, idéia original. A linha completa do AIBO inclui amigáveis ferramentas de programação e acessórios para diversas funções, uma verdadeira plataforma de ensino sem, no entanto, se parecer como tal.

1.5 Objetivos

Este trabalho tem como objetivo principal implementar uma arquitetura multiagente em robôs de pequeno porte da plataforma EyeBot. Foram adquiridos três pequenos robôs da Universidade Werstern Australia para as pesquisas. A proposta inicial era portar um dos níveis decisórios da equipe de agentes de *software* do UFSC-Team (representado pelo processo Interface e responsável pela percepção, ação e respostas de tempo real do agente) para os robôs. Mas, devido as limitações impostas pelo próprio *hardware* dos robôs, decidiu-se mudar o foco do trabalho. Passamos então a desenvolver programas que explorassem ao máximo as potencialidades dos robôs.

Para todos os subsistemas do robô (câmera, sensores, atuadores, módulo de rádio, etc) foram criados exemplos que ilustram a programação destes. Também foi desenvolvido um programa para transmissão e captura de imagens pela câmera do robô via rádio, programa este batizado de *Xcameraviewer*.

Também contribuímos para o início de uma nova linha de pesquisas no Departamento de Automação e Sistemas, pois auxiliamos no desenvolvimento do nível decisório reativo dos agentes de *software*, estudamos e programamos os robôs reais adquiridos e integramos estas duas realidades. Até então, a maior parte dos trabalhos em robótica móvel desenvolvidos utilizavam apenas simuladores. Desta vez tivemos a oportunidade de experimentar com equipamentos reais.

1.6 Organização

O texto está dividido em três partes principais. A parte I, chamada de apresentação, contém apenas um capítulo (este) e apresenta as considerações iniciais do presente trabalho. A segunda parte contém o embasamento teórico necessário. Os capítulos 2, 3 e 4 tratam, respectivamente, dos assuntos Inteligência Artificial, Robótica Móvel e Futebol Robótico. A terceira e última parte contempla efetivamente a descrição do trabalho. O capítulo 5 descreve o projeto UFSC-Team. Em 6 a plataforma de robôs móveis EyeBot estudada é apresentada em detalhes. Ainda nesta parte III são apresentados os resultados obtidos dos testes (capítulo 7) e as conclusões e prováveis trabalhos futuros (capítulo 8). O apêndice A mostra o código fonte do programa de captura e transmissão de imagens *Xcameraviewer*. Em B há a tabela construída e utilizada pelo sistema operacional do robô, conhecida por *hardware description table*. Por fim, em C apresentamos o simulador utilizado nas pesquisas com agentes de *software*.

Capítulo 2

Inteligência Artificial

Este capítulo tem como objetivo apresentar ao leitor algumas das várias áreas de estudo da Inteligência Artificial. Serão abordados apenas aspectos relevantes ao trabalho desenvolvido. O enfoque dado será direto, sucinto e sem a pretensão de esgotar o assunto tratado.

2.1 Histórico

A Inteligência Artificial (IA) nasceu oficialmente em 1956 num *workshop* de verão em Dartmouth College, EUA (Bittencourt, 2001). O termo foi cunhado por John McCarthy[1] e Marvin Minski. Naquela ocasião, um grupo de dez cientistas de renome[2] se reuniu para pesquisar teoria de autômatos e estudar como se processa a inteligência. Naquela ocasião surgiu o primeiro artigo tratando do tópico redes neurais como um modelo de arquitetura computacional (Barreto, 1999). Nasceram também as duas principais linhas de pesquisa da IA; a conexionista (ou neuronal) e a simbólica.

A linha conexionista trata da modelagem da inteligência humana através da simulação dos próprios componentes mais básicos do cérebro, os neurônios. Esta proposta, como será visto na seção 2.2, surgiu antes mesmo da conferência de 56, e propunha um modelo matemático para o neurônio físico. Já a linha simbólica utiliza a lógica para a construção de sistemas inteligentes. Os principais expoentes desta vertente foram John McCarthy e Allen Newell.

Apesar dos objetivos iniciais estarem bem delimitados, a clara definição do termo recém criado não foi possível. Isto deve-se ao fato dos objetivos traçados serem muito abrangentes, tornando difícil o consenso em torno de uma única definição formal para o tópico IA.

Alan Turing[3] propôs um teste de inteligência (Saygin, 2003). Participam deste jogo dois humanos e um computador (figura 2.1). Um dos humanos e o computador (A e B) são separados e colocados em

[1] Criador da linguagem de programação LISP.

[2] John McCarthy, Marvin Minski, Claude Shannon, Nathaniel Rochester, Arthur Samuel, Ray Solomonoff, Oliver Selfridge, Trenchard More, Allen Newell e Herber Simon.

[3] Considerado o pai da computação, foi o criador do método que utilizava uma máquina abstrata (conhecida hoje por máquina de Turing) para provar que existem números não-computáveis e por conseguinte que a matemática é não-decidível.

ambientes isolados onde o único contato com o exterior e através de um terminal onde são digitadas as respostas aos questionamentos do outro humano, o mediador (C). Exclusivamente com base nas respostas dadas, o mediador (C) deve dizer se A ou B é o humano ou o computador. Segundo Turing, se após determinado tempo o mediador não tiver condições de decidir sobre esta questão, então, o computador (o programa do computador na realidade) poderá ser considerado inteligente.

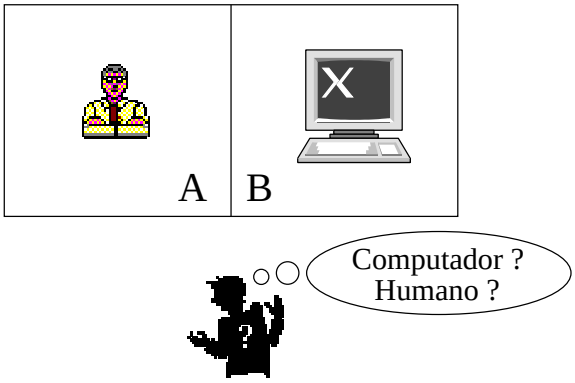

Figura 2.1: Teste proposto por Turing como forma de avaliar a inteligência

Até hoje existem várias definições possíveis de IA. Duas definições que exprimem bem o relacionamento com o presente trabalho (Russel e Norvig, 1995) são:

> "A arte de criar máquinas que executam funções que requerem inteligência quando executadas por pessoas"(Kurzweil, 1990).

> "Um campo de estudo que procura explicar e emular comportamento inteligente em termo de processos computacionais"(Schalkoff, 1990).

A Inteligência Artificial evoluiu e mudou muito desde aquela conferência de verão em Dartmouth onde foram traçados os primeiros objetivos. Na era moderna (considerada a partir da década de 80) compreendeu-se que o "sonho" inicial de se resolver todos os problemas utilizando apenas computadores de grande capacidade e sistemas baseados em conhecimento não eram suficientes. Os primeiros pesquisadores da área por vezes subestimaram a complexidade e a quantidade de informações necessárias para resolver mesmo os mais simples problemas. Este início de muitas promessas e poucos resultados práticos frustrou muitos pesquisadores e desacreditou em parte a IA a perante comunidade científica.

Este quadro começou a se alterar quando os primeiros Sistemas Especialistas (ver seção 2.3) começaram a mostrar resultados concretos. A partir de então, a principal meta da IA tem sido tornar os computadores mais úteis e compreender os princípios que tornam a inteligência possível. Com isto a Inteligência Artificial deixou de ser apenas uma ciência puramente teórica para se tornar uma

poderosa ferramenta para modelagem e representação de problemas, em geral, difíceis de se resolver com as técnicas clássicas de solução. Uma vez alçada ao *status* de ferramenta - pelo menos por parte da comunidade - passou a ser utilizada em campos de estudo tão distintos quanto, por exemplo, medicina e geologia.

2.2 Redes Neurais

As pesquisas em torno de redes neurais iniciaram-se por volta de 1943 pelo médico e filósofo Warren McCulloch e pelo matemático Walter Pitts, antes mesmo da conferência que marcou oficialmente o nascimento da IA. Por este motivo, este tópico é considerado o primeiro trabalho de Inteligência Artificial desenvolvido.

Estes dois pesquisadores propuseram o primeiro modelo matemático para um neurônio, conforme ilustrado na figura 2.2. Neste modelo um conjunto de entradas binárias são combinadas usando alguma função (geralmente apenas uma soma ponderada). A esta combinação dá-se o nome de confluência. O resultado desta confluência produz um estado de ativação que através de uma função de ativação (função de transferência da rede neuronal) ativa ou não a saída. Esta saída assume valor "1" caso o resultado da confluência seja maior que um determinado valor limite (*threshold*) ou "0" caso seja menor (Barreto, 1999).

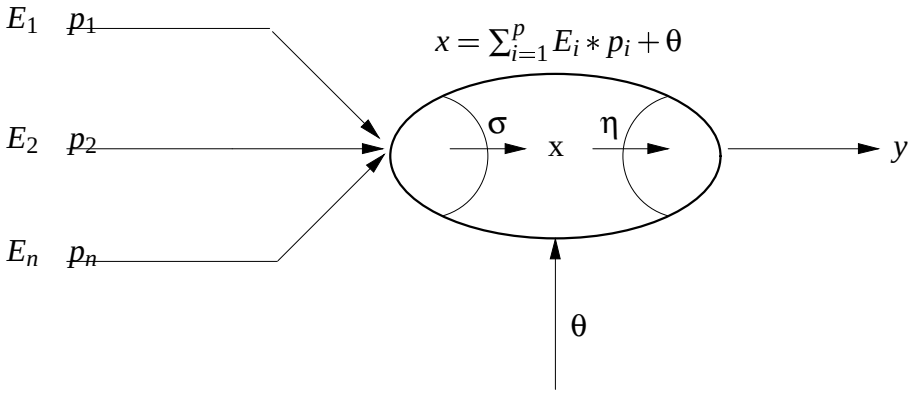

Figura 2.2: Modelo matemático geral do primeiro neurônio criado por McCulloch e Pitts

Este modelo simples de neurônio foi estendido para um mais geral, onde o nível de ativação e a função de ativação podem assumir qualquer função que se queira.

A interconexão de vários neurônios produz as chamadas redes neurais (ou neuronais). Um primeiro modelo de rede neural foi proposto por Frank Rosenblatt em 1957 e recebeu o nome de *Perceptron*. Este modelo de rede é composto por duas camadas; a primeira para a distribuição da informação de entrada e a segunda para o processamento propriamente dito.

As Redes Neurais Artificiais (RNAs) são inspiradas em modelos biológicos e no sistema nervoso.

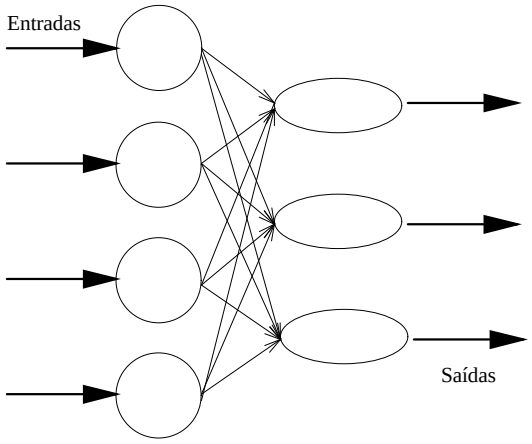

Figura 2.3: *Perceptron*: o primeiro modelo de rede neural criada

Mas, apesar disto, são poucas as semelhanças com redes neurais naturais. Mesmo redes biológicas simples apresentam uma complexidade muito grande. A quantidade de células nervosas de um sistema nervoso natural é de muitas ordens de grandeza maior que qualquer rede neural artificial. Isto, é claro, limita um pouco a aplicação desta técnica na solução de problemas. RNAs são apenas modelos!

Outro problema é que as máquinas disponíveis hoje para o processamento de informações possuem uma arquitetura que não favorece as pesquisas que utilizam algoritmos que exploram o paralelismo de tarefas. Este modelo de computador proposto por *John von Neumann* em meados do século 20 pressupõe uma seqüência quase única do fluxo de informações pois a unidade de controle responsável por organizar a forma como a computação deverá ser feita é única (figura 2.4). O programador é responsável por organizar as instruções numa ordem lógica e sequencial. Qualquer erro nesta etapa pode comprometer o resultado final. Os dados do programa devem estar disponíveis no exato momento em que forem requisitados.

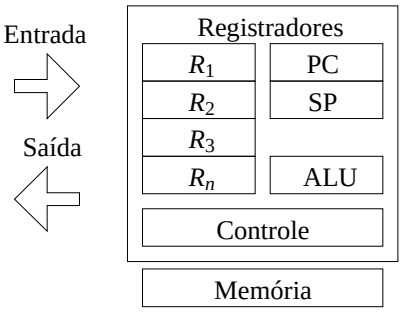

Figura 2.4: Clássica arquitetura de *von Neumann*

Já redes neurais são sistemas que utilizam as conexões entre seus neurônios como forma de resolver tarefas não mais de uma forma sequencial e única, mas distribuída e concorrentemente. Já existem hoje arquiteturas que imitam e exploram esta capacidade maciça de processamento das redes neurais artificiais. Trata-se de *chips* dedicados com vários "neurônios"em paralelo. A figura 2.5 mostra uma possível solução. Ao invés de se utilizar apenas um nó (neste caso cada nó equivale à uma CPU), diversos nós interconectados poderiam representar um grupo de neurônios. O resultado seria um aumento de desempenho da rede pois seria

possível executar diversas operações explorando o paralelismo real que existe. É claro que existem outros complicadores, como o controle deste processo que se tornaria mais complexo. Não podemos simplificar tanto a ponto de acharmos que apenas isto resolveria o problema.

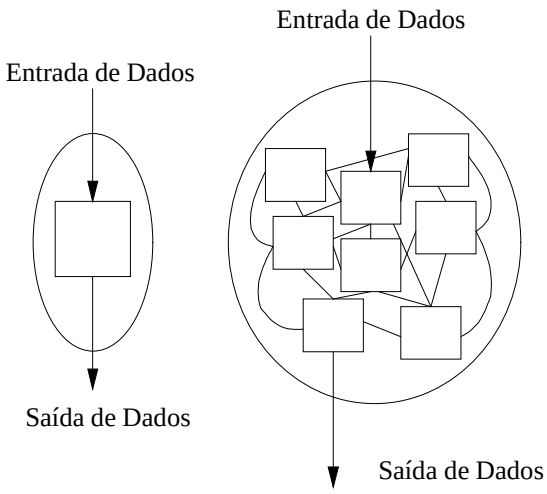

Figura 2.5: Arquitetura alternativa para Redes Neurais

Desde então, as teorias em torno de Redes Neurais Artificiais avançaram e modelos cada vez mais complexos foram desenvolvidos. Atualmente existem modelos de redes para diversas aplicações, tais como processamento da linguagem natural, processamento de imagens e teoria musical. Os modelos de redes mais difundidos são: *adaline*, *hopfield* e *kohonen*.

A tabela 2.1 apresenta uma tabela comparativa de alguns dos principais modelos de redes neurais (Bittencourt, 2001).

2.3 Sistemas Especialistas

Sistemas Especialistas (SE) são programas de computador que tentam imitar um especialista humano em determinada área. São projetados para resolverem problemas da mesma forma que um humano o faria. Em geral, são utilizados como sistemas de apoio a decisão (Waterman, 1985). As tabelas 2.2 e 2.3 mostram uma comparação entre o conhecimento humano e o artificial.

Muitos sistemas foram desenvolvidos para auxiliarem nas tarefas das mais diversas áreas, tais como: medicina, eletrônica, direito, geologia, computação, física, engenharia, entre outros.

Sistemas especialistas são, na verdade, uma generalização dos sistemas de produção de Post. Estes, por sua vez, compreendem todos os sistemas baseados em regras, isto é, compostos por pares de expressões condição/ação. A arquitetura de um SE pode ser tão simples a ponto de conter apenas três partes, como ilustrado na figura 2.6.

Perceptrons	
Aplicações	Reconhecimento de caracteres
Vantagem	Rede Neural mais antiga
Desvantagens	Não reconhece padrões complexos e é sensível a mudanças
Backpropagation	
Aplicações	Larga aplicação
Vantagem	Rede mais utilizada, simples e eficiente
Desvantagens	Necessita de treinamento supervisionado e requer muitos exemplos até o completo aprendizado
Counterpropagation	
Aplicações	Reconhecimento de padrões
Vantagem	Rapidez no treinamento
Desvantagens	Topologia complexa
Hopfield	
Aplicações	Recuperação de dados e fragmentos de imagens
Vantagem	Implementação em larga escala
Desvantagens	Sem aprendizado
Kohonen	
Aplicações	Recuperação de padrões não especificados
Vantagem	Auto-organização
Desvantagens	Pouco eficiente

Tabela 2.1: Comparação entre alguns modelos de Redes Neurais

- Regras: contém o conhecimento acerca do domínio em questão representado, na forma mais simples, de regras de produção (LHS $\rightarrow$ RHS).
- Memória de trabalho: percepção, ou seja, fatos sobre o problema que se pretende resolver.
- Motor de inferência: interpretador responsável por fazer a correspondência entre as regras (LHS) e a memória de trabalho. Aquela regra mais adequada será disparada e o RHS correspondente executado.

O conjunto formado pelas regras e pela memória de trabalho formam o que chamamos de *base de conhecimento* do Sistema Especialista (figura 2.6).

Um exemplo de regra é mostrado no algoritmo 1. Refere-se à um sistema de monitoramento de leituras de um reator nuclear chamado REACTOR com o objetivo de procurar vestígios de acidente (Nelson, 1982).

O conhecimento codificado nas regras deve utilizar uma representação de conhecimento[4], que é a forma como a informação é armazenada. Não existe uma fórmula para decidir qual formalismo

[4]Logica, Redes Semânticas e Quadros (do inglês *frames*) são os três formalismos mais usados.

Perícia Humana	Perícia Artificial
degradável (perecível)	permanente
difícil de transferir	fácil de transferir
difícil de documentar	fácil de documentar
impredizível	consistente
alto custo	nem tanto

Tabela 2.2: Comparação entre perícia humana e artificial (boas notícias)

Perícia Humana	Perícia Artificial
criativo	sem inspiração
adaptativo	precisa ser ensinado
experiência sensorial	entrada simbólica
visão abrangente	visão limitada
possui senso comum	conhecimento apenas técnico e limitado

Tabela 2.3: Comparação entre perícia humana e artificial (más notícias)

utilizar. A decisão ficará por conta do projetista do sistema que levará em consideração o domínio de aplicação do sistema e a forma como está estruturado o conhecimento.

Este conhecimento deve ser obtido através de um profissional experiente da área. A responsabilidade desta tarefa fica a cargo do chamado Engenheiro de Conhecimento (figura 2.7). Ele é o responsável por entrevistar, analisar, discutir e ordenar todas informações extraídas do profissional especialista. Esta é, sem dúvida, a parte mais importante e crítica no desenvolvimento de um SE, também por isto mais susceptível a erros.

Diversos outros aspectos devem ser levados em conta no projeto de um SE, tais como:

- Resolução de conflito: caso duas ou mais regras sejam escolhidas
- Representação de incerteza: como lidar com informações incompletas, incertas ou inexatas (ver seção 2.5)
- Estratégia de busca: raciocínio para frente (a partir dos estados iniciais) ou para trás (a partir dos estados finais), utilização de funções heurísticas

Algorithm 1 Exemplo de uma regra de um Sistema Especialista

IF: A transferência de calor do sistema refrigerador primário para o sistema refrigerador secundário é inadequada
AND o fluxo de entrada de água é baixo
THEN: O acidente é provocado pela perda do fluxo de entrada de água.

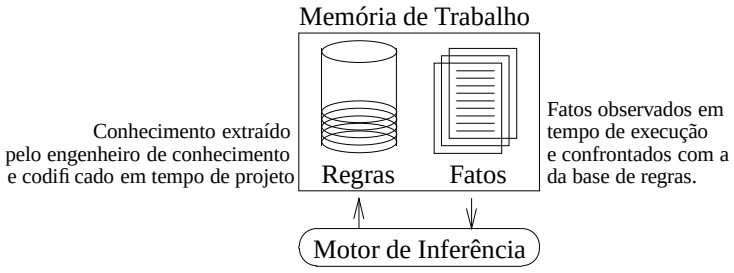

Figura 2.6: Arquitetura de um Sistema Especialista simples

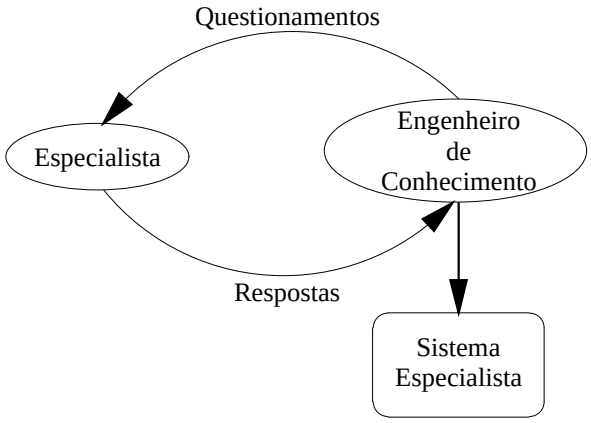

Figura 2.7: Processo de aquisição de conhecimento

- Interface com o usuário
- Aprendizado, ou seja, incorporação de novos fatos à base de regras

Aplicações como estas, talvez, ainda representem a mais bem sucedida tentativa da IA de se produzir algo que se possa considerar inteligente. Este sucesso alcançado por alguns sistemas utilizados comercialmente só foi possível quando percebeu-se que criar um programa para resolver muitos problemas - como no caso do sistema *General Problem Solver (GPS)* idealizado nos primórdios da IA (Ernst e Simon, 1969) - era impraticável. Deste modo restringiu-se ao máximo o escopo de atuação dos sistemas como forma de tornar o problema tratável. Hoje, Sistemas Especialistas são programas altamente eficientes sob domínios extremamente restritos.

2.4 Inteligência Artifi cial Distribuída

A Inteligência Artificial Distribuída (IAD) é a área da IA que trata fundamentalmente de agentes de *software*. Seu funcionamento depende da interação de diversas partes de forma colaborativa para

a correta solução do problema.

A estratégia de decompor um problema em partes menores e resolver cada uma delas isoladamente parece sensata quando a tarefa a ser executada é muito grande e complexa ou quando, naturalmente, o problema apresenta características de paralelismo intrínseco.

Além disso, "dividir para conquistar" também parece ser uma tática bastante utilizada por animais na natureza. Este comportamento pode ser encontrado em colônias de muitas espécies de insetos.

São duas as principais linhas de pesquisa da IAD:

1. Solução Distribuída de Problemas (SDP): neste enfoque, problemas demasiadamente complexos ou grandes são subdivididos com o objetivo de facilitar a resolução. Também são exploradas as características distribuídas e paralelas de tarefas. Isto é feito graças ao desenvolvimento das redes de computadores.

2. Sistemas Multiagentes (SMA): esta linha de pesquisa faz uso dos chamados agentes de *software*. Na seção 2.4.1 serão apresentados maiores detalhes sobre este domínio.

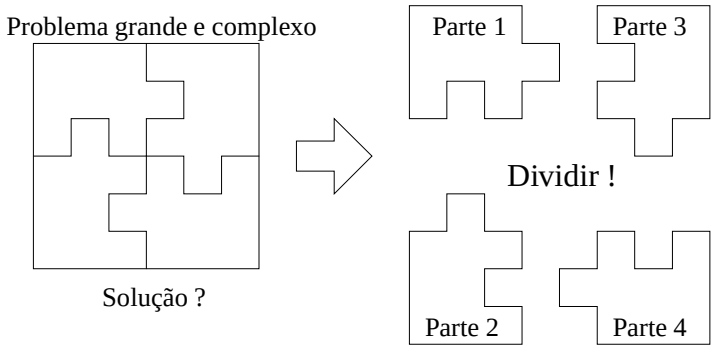

Figura 2.8: SDP: dividir para conquistar

Freqüentemente esta divisão entre o que é um problema SDP e SMA não pode ser feita. Isto porque ambos enfoques são complementares um do outro. Além disso, há superposição de áreas. Tudo depende da caracterização e modelagem do problema. Por este motivo, classificações rígidas não podem ser feitas.

2.4.1 Sistemas Multiagentes

Sistemas multiagentes (SMA) são um dos ramos de pesquisa da chamada Inteligência Artificial Distribuída (IAD). Ambos enfoques (SMA e SDP) têm em comum a utilização de técnicas de IA clássica (redes neurais, sistemas especialistas, etc.) aliadas à teoria de sistemas distribuídos e redes de computadores. A IAD trata primordialmente de agentes. Assim como no caso da própria IA, a

definição do que é um agente não tem a aceitação de toda comunidade. Uma possível definição que contempla os principais aspectos do presente trabalho é mostrada abaixo (Bittencourt, 2001):

> Chama-se agente uma entidade real ou abstrata que é capaz de agir sobre ela mesma e sobre seu ambiente, que dispõe de uma representação parcial deste ambiente, que, em um universo multiagente, pode comunicar-se com outros agentes, e cujo comportamento é conseqüência de suas observações, de seu conhecimento e das interações com outros agentes.

Parece que a inspiração para este domínio de estudo tenha sido a colaboração entre animais em uma colônia. Numa colméia, por exemplo, existem grupos de indivíduos especializados em tarefas bem específicas. A primeira vista parece não haver qualquer organização mas, observando-se atentamente percebe-se que toda colônia vive numa perfeita harmonia e que esta depende de todos, independente da posição ocupada na escala hierárquica.

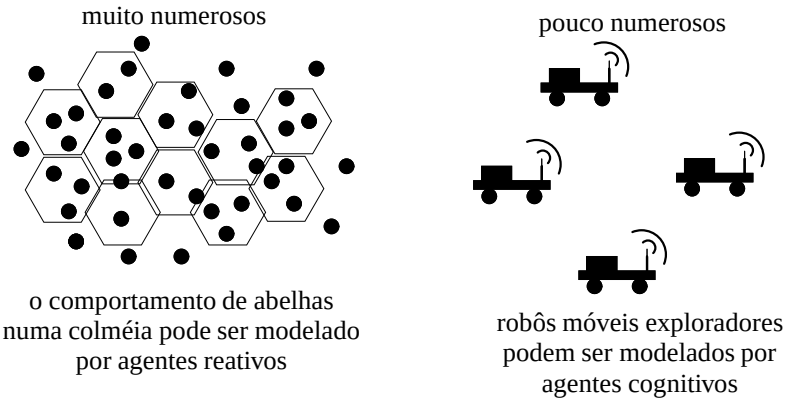

Figura 2.9: Agentes reativos e cognitivos

Existem dois tipos de agentes: os reativos e os cognitivos. Agentes reativos são aqueles cujo comportamento é inspirado em comunidades de insetos, como no caso de abelhas e formigas. Sistemas multiagentes baseados nestes tipos de agentes caracterizam-se por serem formados por um grande número de agentes. Cada agente apresenta uma resposta em virtude de um estímulo (condição-ação) e não planeja suas futuras ações. Estas características os tornam ideais para aplicações como robótica móvel e sistemas de tempo real. Estas requerem, mais do que respostas instantâneas, garantias temporais determinadas *a priori*. A figura 2.10 mostra a arquitetura proposta por Rodney Brooks para agentes reativos chamada de *arquitetura de subsunção*. Este modelo propõe uma divisão em camadas chamado de níveis de competência. Cada uma destas camadas responde por diferentes tarefas, ou seja, comportamentos distintos (Brooks, 1986) perante os obstáculos do caminho.

Podemos vislumbrar uma gama bastante grande de aplicações tais como:

- supervisão e manutenção preventiva de *hosts* em uma rede de computadores sem a intervenção de um operador humano (Prouskas et al., 2000)
- exploração de ambientes insalubres ou perigosos por uma equipe de robôs móveis (Bares e Whittaker, 1994)
- automatização de sistemas de controle de trafego (Nunes, 2002)
- robótica

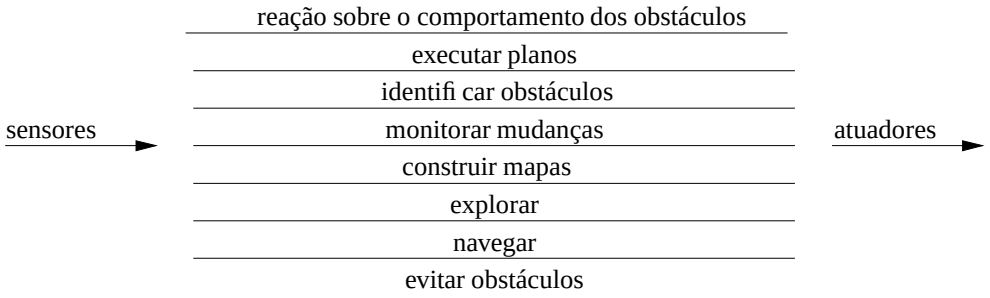

Figura 2.10: Arquitetura de subsunção proposta por Brooks

A segunda categoria de agentes chama-se cognitivos. Sistemas que utilizam estes tipos de agentes podem ser considerados inteligentes. Domínios como *processamento da linguagem natural* e *sistemas de ensino assistido por computador (tutores inteligentes)* são bons exemplos de campos onde está presente esta tecnologia (Frigo, 2002). Suas principais características são:

- são pouco numerosos ao contrário dos agentes reativos.
- possuem uma representação do ambiente onde estão inseridos e, por este motivo, planejam suas futuras ações.
- são capazes de comunicarem-se com outros agentes da comunidade utilizando alguma linguagem/protocolo.

Este ramo da Inteligência Artificial Distribuída é uma das que mais tem crescido ultimamente. Há muitas pesquisas sendo desenvolvidas na área. Podemos pensar em centenas de tarefas, inclusive do dia-a-dia, em que sistemas automatizados geridos por agentes nos ajudariam.

Imagine a problemática de se marcar uma reunião de trabalho em que os participantes são muito ocupados e possuem pouca margem de manobra em suas agendas. Neste caso, cada um dos membros deste grupo de trabalho iria postar seus melhores horários numa aplicação inspirada em *softwares* de *groupware* (Ferreira e Wainer, 2000). Após todos terem feito o mesmo, agentes representando os participantes do grupo iniciam uma "conversação" para tentar chegar a um acordo com relação a data

da reunião. Caso não seja possível atender a todos, sugestões podem ser feitas de modo a acomodar o maior número de pessoas. No final do processo, os membros simplesmente seriam comunicados (via *e-mail* por exemplo) do resultado das negociações. Esta tarefa, enfadonha e trabalhosa, realizada na maioria dos casos por secretárias, poderia ser executada pelas próprias pessoas envolvidas de forma automática.

2.5 Lógica Nebulosa

A teoria clássica de conjuntos estabelece limites rigorosos e precisos para a pertinência de elementos de conjuntos. Isto é, suponha que estejamos querendo classificar os números naturais pares e ímpares utilizando para isto a teoria de conjuntos. Se fossemos utilizar a representação de Venn (figura 2.11), teríamos dois conjuntos distintos, infinitos e totalmente independentes. Não há como um número natural não ser nem par nem ímpar. Por que isto acontece ? A resposta é simples; não se pode pensar em números "meio-pares" ou "meio-ímpares". Eles pertencem aos conjuntos dos pares *ou* dos ímpares, mas nunca a ambos. Para este tipo de informação, a teoria clássica de conjuntos se presta perfeitamente.

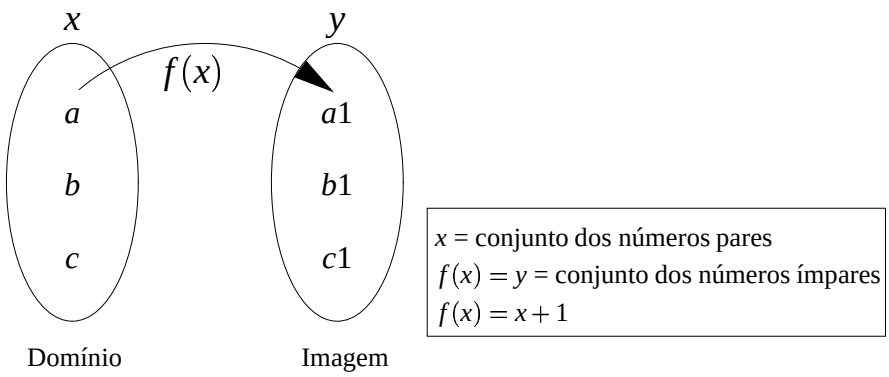

Figura 2.11: Diagrama de Venn para a representação de conjuntos

Acontece que em IA freqüentemente temos que lidar com situações vagas e imprecisas. Alias, nós humanos estamos acostumados a pensar desta forma. Podemos concluir que raciocinar sobre estas bases requer inteligência e, se conseguirmos de alguma forma passar esta idéia de incerteza para nossos programas, teremos dado um grande passo na manipulação deste tipo de dado. Pense no problema de se classificar um conjunto de pessoas segundo sua estatura (figura 2.12). Se utilizarmos para isto apenas três conjuntos e a noção clássica de conjuntos, provavelmente o resultado será algo um tanto artificial, fora da realidade. Isto porque ninguém é considerado totalmente alto ou baixo. Esta classificação é difusa, sem fronteiras bem definidas. Há, portanto, valores intermediários. Assim,

neste caso, a teoria de conjuntos clássica que representa os dados de forma abrupta[5] deve ser evitada.

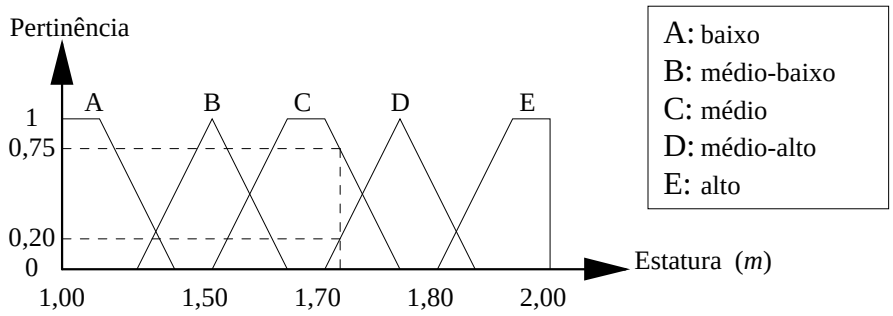

Figura 2.12: Classificação da estatura humana utilizando conjuntos nebulosos

Neste exemplo, "baixo", "mediano" e "alto" são chamados de *variáveis lingüísticas*. Isto porque são palavras, ao invés de números, que são utilizadas para quantificar a *pertinência* com relação à um conjunto, chamado neste modelo de conjunto nebuloso (ou *fuzzy*). Este conjunto é composto por um intervalo de valores que definem comportamentos com base em certas regras. A transição entre um conjunto nebuloso e outro se dá aplicando a função de pertinência. Na figura 2.12 são mostrado dois tipos de conjuntos; um triangular e outro trapezoidal. Estas duas funções são as mais usadas para representação, mas não as únicas. Sua utilização deve-se mais a simplicidade frente às outras funções, como a linear e a sigmóide.

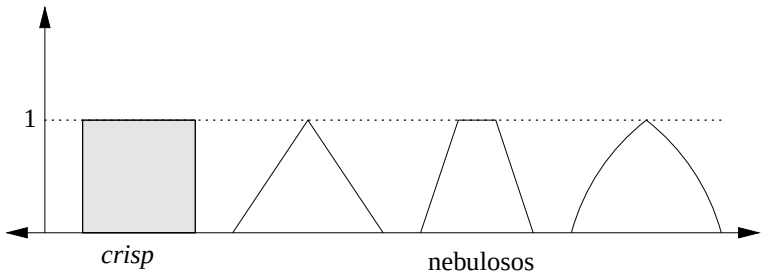

Figura 2.13: Comparação entre conjuntos *crisp* e nebulosos

Em lógica nebulosa, a pertinência não é mutualmente exclusiva; pode-se pertencer ao mesmo tempo a dois conjuntos. Uma pessoa segundo esta classificação pode pertencer $0,7$ ao conjuntos dos altos e $0,3$ ao conjuntos dos medianos. Freqüentemente esta pertinência fica restrita ao intervalo $[0,1]$. Ou seja, à medida que se pertence mais a um determinado conjunto, em geral pertence-se menos a outro. Dependerá da função de transição a pertinência a varios conjuntos nebulosos simultâneos (Levine, 1996).

Um dos pontos mais importantes desta teoria consiste na escolha desta *função de pertinência* que

[5]Também chamados de conjuntos *crisp*.

faz o mapeamento entre variáveis numéricas comuns em *variáveis lingüísticas* para serem usadas pelo sistema. Este processo chama-se de *fuzzificação*. O caminho inverso, ou seja, de variáveis lingüísticas (alto, baixo, magro, quente, longe, etc) para valores numéricos chama-se *defuzzificação*. A figura 2.15 ilustra uma aplicação muito utilizada desta teoria; o controle de processos utilizando lógica nebulosa. Neste caso, ao invés dos parâmetros do controlador serem valores numéricos fixos, são utilizadas variáveis lingüísticas. Em (Martinez et al., 1993) é apresentado um sistema que utiliza esta teoria para a detecção e o desvio de obstáculos de um robô móvel.

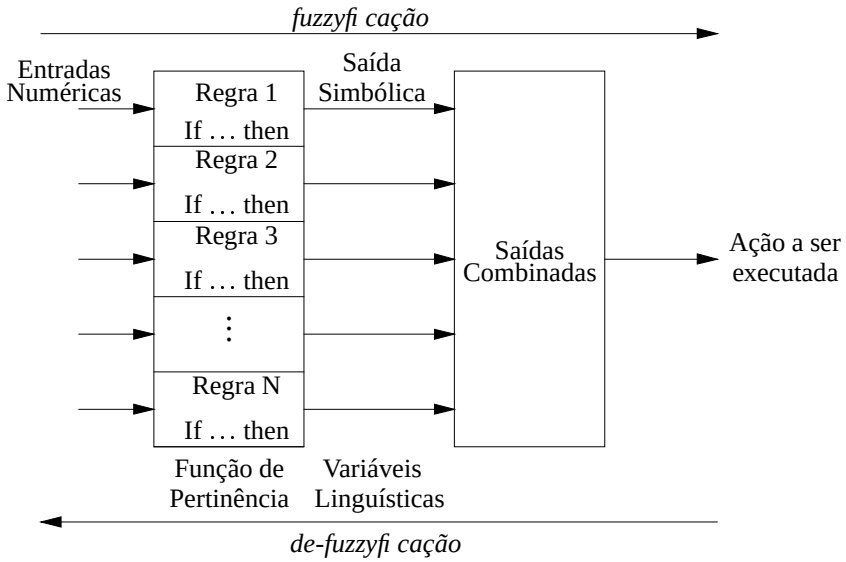

Figura 2.14: Processo de *fuzzificação* e *defuzzificação*

Vantagens em se utilizar este tipo de abordagem:

- os conjuntos nebulosos podem ser construídos de uma forma muito mais intuitiva e natural, sem o total conhecimento matemático do sistema com que se está tratando
- trabalham com informações imprecisas
- uma solução pode ser alcançada por diferentes caminhos
- é mais robusto pois a falta de regras não necessariamente inviabiliza o resultado

Além destas características, no caso de controladores, também destacamos as seguintes vantagens:

- ajustes podem ser feitos tanto nos conjuntos quanto nas funções de pertinência (ajustes finos)
- a transição entre um nível e outro de saída do controlador se dá de forma gradual e sem sobressaltos

- apresentam bons resultados em processos não-lineares, entradas irregulares e restrições conflitantes
- futuras alterações podem ser feitas mais facilmente alterando-se as regras do sistema nebuloso.

Entre as desvantagens estão (Cruz, 1998):

- é mais difícil desenvolver um modelo a partir de um sistema nebuloso; alias, este é o principal motivo que nos leva a adotar esta técnica, problemas em que a modelagem é complexa ou desconhecida
- não há uma definição matemática precisa como outros sistemas

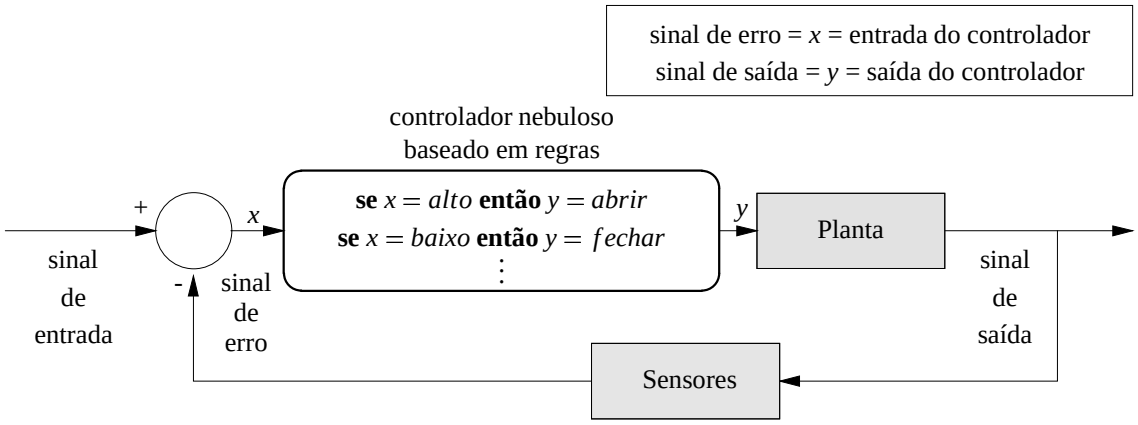

Figura 2.15: Controle de processos utilizando lógica nebulosa

As bases para esta teoria foram lançadas por um pesquisador americano chamado Lotfi A. Zadeh em meados da década de 60 com o artigo Fuzzy Sets (Zadeh, 1965). Este artigo serviu de inspiração para outros pesquisadores que logo a seguir começaram a aplicar e aperfeiçoar esta teoria. Hoje, esta teoria tem aplicações em muitas áreas distintas onde raciocínio impreciso se faz necessário, tais como controle e sistemas baseados em conhecimento.

2.5.1 Exemplos

Citamos abaixo alguns exemplos onde esta teoria se faz presente com ótimos resultados:

- Metrô de Sendai: sistema nebuloso desenvolvido nos anos 80 pela empresa japonesa Hitachi para o controle do tráfego de trens. Cobre 16 estações em 13,5km. Em 1987 este sistema substituiu os operadores humanos, que passaram a operar o sistema apenas nos horários em que os trens estão mais vazios. Antes de entrar em operação foram feitas 300.000 simulações e 3000 testes com os vagões vazios

- Lavadoura de roupas: a Continental do Brasil fabrica uma lavadoura com tecnologia nebulosa
- Aspirador de pó: controle nebuloso detecta mudanças no fluxo de pó para ajustar a potência do motor
- Indústria automobilística: a Mitsubishi desenvolveu um sistema nebuloso que regula a força com que os freios são acionados, evitando derrapagens nos sistemas

Capítulo 3

Robótica Móvel

3.1 Introdução

A *Robotic Industries Association (RIA)* define um robô como sendo um manipulador programável multi-funcional capaz de mover materiais, partes, ferramentas ou dispositivos específicos através de movimentos variáveis programados para realizar uma variedade de tarefas (Russel e Norvig, 1995).

Esta definição descreve toda uma categoria de máquinas. Desta forma, qualquer equipamento capaz de ser programado de alguma forma é considerado um robô. A indústria já utiliza esta tecnologia como substituta da mão-de-obra humana, em especial onde há riscos envolvidos. Os robôs utilizados na indústria são, de um modo geral, grandes, fixos e desprovidos de inteligência de alto nível e não-autônomos. Poderíamos classificá-los como máquinas automáticas programáveis (ver figura 3.1).

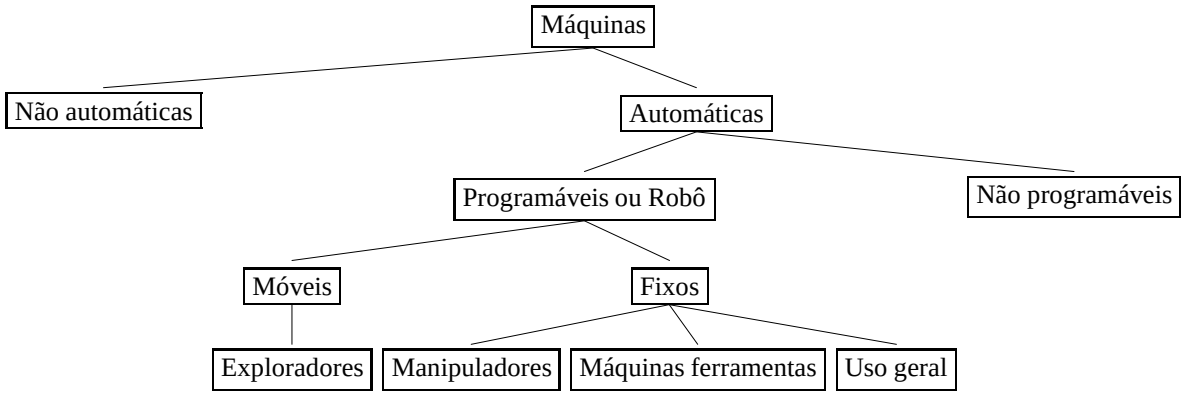

Figura 3.1: Possível classificação para máquinas

Além desta classificação funcional, também podemos distribuí-los em gerações segundo sua complexidade (Nof, 1999):

- Primeira Geração: São robôs tipicamente utilizados em ambientes industriais, desprovidos de "inteligência" senão aquela programada. O mundo onde atuam deve ser preparado pois eles

não são capazes de perceber os objetos ao seu redor. Poucos utilizam um computador dedicado embarcado. No entanto, ainda são usados em fábricas devido ao aumento de produtividade e qualidade do produto gerado e da durabilidade. Tendem a desaparecer à medida que as exigências em termos de produtividade e velocidade aumentarem ainda mais

- Segunda Geração: A rápida expansão do mercado de semicondutores fez com que computadores se tornassem produtos baratos o suficiente para equipar robôs. Isto permitiu um avanço significativo pois máquinas anteriormente muito limitadas puderam ser equipadas com controladores que permitiram cálculos em tempo real e controle mais preciso dos atuadores ao longo de trajetórias. Sensores de força, torque e proximidade puderam ser integrados proporcionando maior adaptabilidade e precisão ao trabalho desempenhado pelo robô em seu ambiente. Entre as aplicações estão, pintura, montagem e soldagem.

- Terceira Geração: Representam os modelos mais avançados hoje disponíveis. Caracterimzam-se por incorporarem múltiplos processadores, cada um deles funcionando de forma assíncrona e independente dos demais. Também possuem sistemas supervisórios e de controle destes processadores utilizando funções de alto nível. Além disso, a fusão sensorial, ou seja, a incorporação e utilização em conjunto de vários tipos de sensores está permitindo comportamentos reativos e, eventualmente, cognitivos. São capazes de se comunicarem com outros sistemas. Todo seu potencial ainda não foi explorado. Entre suas aplicações estão tarefas extremamente delicadas e especializadas como medicina.

Nossa atenção estará voltada, no entanto, para aquela categoria de máquinas capazes de serem programáveis em alto nível via *software*. Também, trataremos apenas de robôs móveis, ou seja, máquinas não estáticas passíveis de locomoção. A figura 3.1 ilustra uma possível classificação para máquinas.

Todo robô, móvel ou não, possui características estruturais em comum, tais como:

- Manipulador: parte mecânica móvel do robô; geralmente representado por uma garra.

- Atuador: dispositivo responsável pela movimentação do manipulador. Conforme a figura 3.2, os atuadores são responsáveis por atuarem no mundo físico. Eles representam esta interface. Motores elétricos são um caso típico de aplicação em robôs de menor porte, embora atuadores pneumáticos e hidráulicos também sejam usados onde maiores forças são exigidas.

- Sensor: dispositivo sensorial do robô. É ele que "sente" o mundo ao seu redor. Representa a outra "ponta" do par **sensor-atuador**. Assim como o dispositivo atuador, é uma parte vital para o bom funcionamento de qualquer robô, pois o mundo do robô restringe-se às interfaces *mundo físico-sensor* e *mundo físico-atuador*. Todos os cálculos e algorítmos executador pela CPU da máquina dependem da boa resposta deste dispositivo.

- Controlador: unidade central, geralmente representada por um computador e responsável pelo controle de todos os dispositivos e planejamento das ações da máquina.
- Fonte de energia: necessária para o funcionamento do controlador e por conseqüência de todas as outras partes. Tipicamente representada por uma bateria.
- Transmissões: representados por engrenagens, correias e caixas de redução. Estão presentes em robôs industriais de médio e grande porte para a transmissão da energia mecânica.

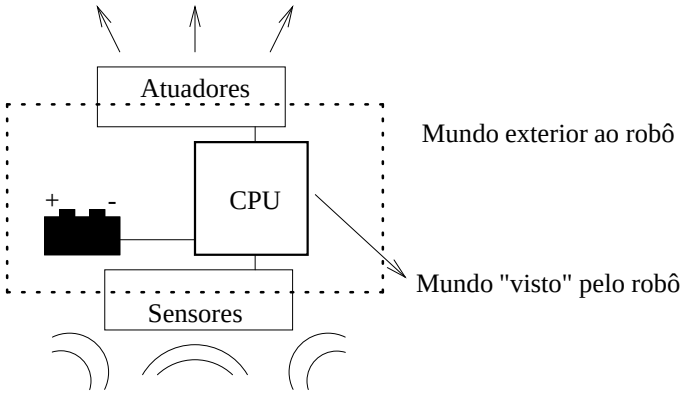

Figura 3.2: O mundo do robô

A lista a seguir dá uma pequena mostra do quão antigo é o desejo humano de se construir mecanismos para automatizar as mais diversas tarefas.

≈270aC um engenheiro grego chamado Ctesibus projeta órgãos e relógios d'água com figuras móveis

1818 Mary Shelley escreve *Frankenstein* o qual fala sobre uma criatura artificial montada a partir de diferentes partes de cadáveres humanos. Façanha esta realizada pelo Dr. Frankenstein

1921 O termo robô foi usado pela primeira vez num jogo chamado *Rossum's Universal Robots (R.U.R)* pelo escritor tcheco Karel Capek. Capek achava que os robôs iriam dominar e subjugar a raça humana, uma visão bastante diferente do pesquisador Isaac Asimov

1941 O escritor de ficção científica Isaac Asimov usa pela primeira vez a palavra *robótica* para descrever a tecnologia de robôs e para prever o nascimento de uma poderosa indústria de robôs

1942 Asimov escreve *Runaround*, uma estória sobre robôs onde são enunciadas as Três Leis da Robótica:

Um robô não pode ferir um humano ou, através da inação, permitir que um ser humano venha a se machucar

Um robô deve obedecer às ordens de ser humano exceto quando tais ordens entrem em conflito com a primeira

Um robô deve proteger sua existência a menos que tal proteção nao seja conflitante com a primeira ou segunda leis

1948 Publicado *Cybernetics* de Norbert Wiener, uma influência na pesquisa de inteligência artificial

1956 George Devol e Joseph Engelberger formam a primeira companhia de robôs do mundo

1959 Um sistema de manufatura assitido por computador foi demonstrado no laboratório de servo-mecanismos no MIT

1961 O primeiro robô industrial foi ativado numa fábrica de automóveis da General Motors em New Jersey. Chamava-se UNIMATE

1963 O primeiro braço robótico artificial controlado por computador foi projetado. O *Rancho Arm* foi idealizado como uma ferramenta em que suas seis juntas proporcionavam a flexibilidade de um braço humano

1965 criado DENDRAL, o primeiro sistema especialista a executar instruções com base no conhecimento armazenado em sua base de regras e obtido através de um especialista humano

1968 Marvin Minsky desenvolve o *Tentacle-Arm*, um braço mecânico controlado por um computador PDP-6 e capaz de erguer uma pessoa. O nome vem do fato deste dispositivo movimentar-se com um polvo

1969 Criado o *Stanford Arm*, o primeiro braço mecânico controlado por computador e movido à energia elétrica

1970 *Shakey* foi considerado o primeiro robô móvel controlado por técnicas de Inteligência Artificial. Era produzido pela SRI International

1974 Um braço mecânico (*Silver Arm*) que executava montagem de pequenas peças utilizando a informação recebida de sensores de toque e pressão foi desenvolvido

1979 O *The Stanford Cart* atravessou uma sala repleta de cadeiras sem auxílio humano. O robô possuia uma câmera de TV que registrava imagens de vários ângulos e enviava ao computador que analizava as distâncias entre o robô e as cadeiras e decidia qual ação executar.

3.2 Aspectos Construtivos

Na construção de robôs móveis, uma gama razoavelmente grande de dispositivos mecânicos e elétricos são necessários. As tabelas 3.1 e 3.2 abaixo dão uma noção dos componentes sensores e atuadores mais comuns encontrados em aplicações típicas (Nehmzow, 2000; Jones et al., 1999).

Proximidade: magnéticos, indutivos, capacitivos, ultrasônicos, óticos.
Odométricos: potenciômetros, sincros e resolvers, óticos.
Navegação: *Doppler*, inercial.
Localização: ultrasônicos, óticos.

Tabela 3.1: Sensores classificados segundo a aplicação

Além da programabilidade da máquina, outra restrição a ser imposta por nós é a autonomia. Este trabalho tem como objeto de estudo robôs móveis autônomos, ou seja, robôs programáveis capazes de tomarem suas próprias decisões com ou sem auxílio externo. Para realizar tal tarefa é necessário, além dos sensores e atuadores vistos nas tabelas 3.1 e 3.2, uma unidade de processamento embarcada no robô. Esta unidade de processamento é tipicamente representrada por uma CPU ou um micro-controlador. Os robôs móveis em uso hoje possuem, praticamente, um computador pessoal de médio porte embarcado, tal o nível de sofisticação e processamento exigidos pelos algoritmos. Estes são, atualmente, o estado da arte da robótica móvel, como visto anteriormente no capitulo 1.

3.3 Desafios

O projeto de robôs móveis autônomos representa um desafio aos pesquisadores da área devido às seguintes constatações (Russel e Norvig, 1995):

1. o mundo é inacessível: sensores não são perfeitos em perceber o mundo real.
2. o mundo é não-determinístico: devido às incertezas envolvidas, o resultado final não pode ser completamente conhecido.
3. o mundo é não-episódico: os efeitos de uma ação mudam com o passar do tempo; problemas relacionados a decisão e aprendizado devem ser tratados.
4. o mundo é dinâmico: é preciso saber quando deliberar e quando agir.
5. o mundo é contínuo: a evolução dos estados internos da programação do robô evoluem de forma discreta enquanto que o mundo ao redor é contínuo; dificultando a programação e a estimação de ações e comportamentos.

Além dos desafios técnicos, existem também as implicações sociais das mudanças causadas por esta nova área. Nunca antes tantas modificações foram feitas no modelo de produção e nas relações

1. Pneumáticos
   (a) Cilindros Pneumáticos
   (b) Motores Pneumáticos
      i. aletas rotativas
      ii. pistão axial
2. Hidráulicos
3. Elétricos
   (a) Corrente Alternada (AC)
      i. síncronos
      ii. assíncronos (de indução)
   (b) Corrente Contínua (DC)
   (c) Motores de Passo
   (d) Servomotores (DC)

Tabela 3.2: Classificação de atuadores com base na energia de acionamento utilizada

do trabalho. O receio de que as máquinas acabarão por nos substituir torna-se real à medida que a tecnologia avança. As famosas Três Leis da Robótica enunciadas por Isaac Asimov podem ser reescritas com uma conotação mais atual:

1. Robôs devem continuar substituindo pessoas em tarefas perigosas e insalubres (isto beneficia à todos)
2. Robôs devem continuar substituindo pessoas em tarefas que elas não querem desempenhar (isto também beneficia a todos)
3. Robôs devem substituir pessoas naquelas tarefas em que robôs são mais viáveis economicamente (isto irá prejudicar muitos inicialmente mas, inevitavelmente, irá beneficiar todos á médio e longo prazo)

3.4 Aplicações

Robôs são cada vez mais empregados em ambientes nocivos ou perigosos aos seres humanos, em tarefas repetitivas e enfadonhas. Mas além destas "clássicas" aplicações, medicina, segurança, exploração, patrulhamento e mesmo diversão estão entre os nichos onde pesquisas de ponta estão

sendo conduzidas (ver capítulo 1). Outros motivos para o crescente uso de robôs na indústria são os menores custos a médio e longo prazo, o aumento da produtividade e a garantia de um nível constante de qualidade do produto gerado.

3.5 Futuro

Sem especularmos muito, podemos esperar que tecnologias que estão aparecendo hoje se transformem rapidamente em emergentes soluções amanhã. Abaixo estão algumas delas.

3.5.1 Computadores e Sistemas Computacionais

Computadores com grande poder de processamento podem ser comprados por menos de US$1000. A preocupação com os custos do equipamento que vai equipar um determinado robô deve desaparecer com o tempo. A velocidade dos modernos equipamentos torna possíveis a execução de algorítmos em tempo real a um custo muito baixo. Mais graus de liberdade (braços, mãos, etc) podem ser controlados simultâneamente. As novas tecnologias de semicondutores vão tornar estes dispositivos tão comuns a ponto de estarem presentes em eletrodomésticos. Estes avanços também devem favorecer a utilização de sistemas operacionais embarcados. Não existe mais a obrigatoriedade de se desenvolver algo novo, "enxuto" e personalizado. Pode-se perfeitamente modificar sistemas baseados em software livre disponíveis gratuitamente.

3.5.2 Sensores

Cada vez mais este tópico passa a ter maior importância. Aplicações sofisticadas exigem dispositivos de excelente resposta. A visão deve logo se tornar o sentido padrão. A tecnologia de construção de câmeras digitais de alta resolução e de largo espectro estão começando a aparecer. Algorítmos para o processamento destas imagens também estão sendo pesquisados. Aplicações como inspeção e monitoramento serão muito beneficiados com estes avanços. O aumento da oferta e a miniaturização dos sensores permitirão que novas áreas sejam exploradas com grandes implicações disto na nossa sociedade.

3.5.3 Energia

Um dos maiores problemas da robótica móvel é justamente a sua fonte de energia. Quase sempre esta é de origem química. Mesmo com os recentes avanços na contrução das pilhas e baterias, estas continuam a limitar determinadas aplicações. O caso do robô *Sojourner* enviado a Marte é um exemplo. Apesar de ter um painel solar capaz de suprir as necessidades da maior parte dos circuitos, suas baterias eram fundamentais nos períodos em que não havia luz. Devido a menor densidade de armazenamento de carga e ao volume e peso, a NASA optou por não utilizar baterias recarregáveis. Isto limitou tremendamente a vida útil do robô.

Este exemplo mostra o quanto ainda é frágil esta tecnologia. Novos métodos de geração de energia elétrica suplantaram as baterias de chumbo-ácido e níquel-cádmio. O estado da arte hoje são baterias de íons-de-lítio (Li-ion) e níquel-metal-hidreto (NiMH). Mas, mesmo estas não são suficientes para grandes autonomias. Parece que o futuro aponta para as células combustíveis, conversores eletroquímicos de hidrocarbonetos em energia elétrica. Esta tecnologia poderá, futuramente, mover diversos tipos de equipamento com menos poluição e ruído.

3.5.4 Redes

A *World Wide Web* plantou a semente das comunicações em nível global. Parece que a tendência do mundo de hoje é que todas as coisas se comuniquem e troquem informações entre si. Na robótica também não é diferente. Tanto robôs móveis quanto fixos estarão equipados com transmissores/receptores de rádio. O controle remoto em tempo real já é utilizado, por exemplo, na tele-medicina. Um cirurgião à quilômetros de distância comanda um robô que repete os movimentos feitos por ele numa sala de cirurgia no paciente. O sistema de posicionamento global (rede de satélites GPS) permitirá a qualquer um ter a exata posição. As possibilidades serão muitas quando este sistema começar a equipar dispositivos do dia-a-dia. Nota-se também uma evolução das redes industriais. Os equipamentos de "chão-de-fábrica" como CLPs estão melhorando e se sofisticando a cada geração. Dentro de pouco tempo todos equipamentos de uma indústria (dependendo da natureza desta) estarão integrados e "conversando" protocolos comuns. Esta parece ser a mais forte tendência. É quase inadmissível hoje dispositivos *stand-alone*.

3.5.5 Aprendizado de máquina e Redes Neurais

No início da década de 40 o primeiro modelo de rede neural foi criado. Batizado de *Perceptron*, esta estrutura era capaz de reconhecer caracteres simples. Após um período de inatividade e pouco trabalho, nas últimas duas décadas as pesquisas se intensificaram, tanto na teoria quanto na prática. Hoje o tópico Redes Neurais é muito ativo e, para o futuro, espera-se que seja a solução para problemas como reconhecimento da fala, modelagem de sistemas lineares e não-lineares, sistemas especialistas. O objetivo final é conseguir fazer com que robôs aprendam e se adaptem ao ambiente onde estão inseridos da forma mais natural possível.

3.5.6 Supercondutividade

A engenharia de materias tem feito bons progressos na descoberta de materiais e métodos para se obter o efeito da supercondutividade. Cerâmicas especiais podem ser o futuro na fabricação de dispositivos que dissipem baixíssima energia, reduzindo custos e, principalmente, tamanho no caso de aplicações mais exigentes. A supercondutividade também pode abrir as portas para outro desejo antigo, armazenar energia elétrica para ser utilizada quando necessário, sem perdas.

Espera-se que robôs autônomos possam integrar todas estas tecnologias num futuro próximo. Este tópicos vistos rapidamente aqui dão uma noção do que pode vir a ser encontrado.

3.6 IA e Robótica

Inteligência Artificial e Robótica são duas áreas distintas. Apesar disto, é cada vez maior a integração entre elas. Os avanços conseguidos por estes tópicos estão entre os maiores feitos da ciência na última década. A Robótica personificou a IA, que saiu do interior dos computadores para ganhar "vida". Mesmo com os recentes avanços, ainda há muito por fazer. Os aspectos relacionados a construção destas máquinas estão muito bem estabelecidos, mesmo nas aplicações mais exigentes. A situação só deve mudar significativamente quando os resultados das pesquisas em engenharia de materiais comecarem a aparecer no mercado.

Os estudos concentram-se atualmente no desenvolvimento de estratégias de controle, planejamento de trajetória, localização, e orientação (Marchi, 2001). Uma área bastante promissora é o controle utilizando técnicas de Inteligência Artificial, como por exemplo lógica nebulosa. A seção 5 apresenta um exemplo onde isto foi utilizado com sucesso.

3.7 Visão Computacional

A visão computacional desempenha papel de grande importância na robótica móvel. Quando se pretende utilizar o sentido da visão para a localização e navegação, problemas maiores surgem. A integração de câmeras de vídeo ao já complexo *hardware* traz complicadores, por exemplo, o tratamento e processamento destas imagens capturadas vai requerer rotinas elaboradas para a extração de informações úteis (Orth, 2001). Entretanto, isto nem sempre é possível devido a limitações computacionais, como é o caso dos robôs da plataforma EyeBot com que estamos trabalhando. Detalhes mais específicos do tópico visão computacional aplicado aos robôs da plataforma *EyeBot* serão vistos no capítulo 6.

Esta é uma área muito vasta e de intensas pesquisas. Este trabalho não tem como objetivo estudar a fundo as teorias relacionadas à visão computacional, mas apenas aquilo que for relevante ao presente trabalho.

Capítulo 4

Futebol Robótico

4.1 Introdução

Desde muito cedo, jogos e problemas matemáticos fazem parte dos estudos da Inteligência Artificial. No início estes eram simples, mas nem por isto menos importantes. *Puzzles* como "Jogo do oito" e "Torre de Hanói" sempre fascinaram os pesquisadores da área. Além destes, clássicos jogos de tabuleiro como xadrez, damas e gamão também despertaram o interesse e alimentaram o desejo de homens em desenvolver programas ou máquinas "inteligentes" capazes de vencer competidores humanos em torneios mundiais. Há, inclusive, um ramo de estudo da IA que trata justamente do tema jogos[1]. Busca em espaço de estados e representação do problema são alguns dos tópicos estudados em teoria de jogos e que podem ser aplicados a muitos outros campos.

Aparentemente, a Inteligência Artificial sempre teve uma espécie de *problema padrão*, algo que orientasse as pesquisas de muitos grupos e que, principalmente, representasse um desafio a ser vencido. O jogo de xadrez sempre ocupou esta posição. Jogos de computador capazes de realizar tal proeza figuram entre as primeiras e principais implementações em IA. Mais antigos ainda são os autômatos enxadristas; máquinas ditas capazes (na realidade não passavam de fraudes) de jogarem xadrez[2]. Mas, recentemente[3], uma máquina especialmente desenvolvida para esta finalidade mudou a história. Pela primeira vez, um grande mestre perdeu uma partida num torneio mundial. *Deep Blue*, a máquina desenvolvida pela IBM foi capaz de realizar tal feito. De lá para cá várias outras partidas foram realizadas com vitórias para ambos os lados. Máquinas ainda mais possantes foram desenvolvidas e, também, outros grandes mestres venceram e foram derrotados.

Mas, o fato é que estes acontecimentos parecem ter encerrado um capítulo importante da história da IA. Ainda mais com os recentes desenvolvimentos em tecnologias de construção de processadores, este problema que a tanto tempo desafiava, não mais desperta tanta atenção. Deste modo, parte da

[1] Não confundir com a área de teoria de jogos da matemática.
[2] Como o famoso autômato do Barão Wolfgang von Kempelem por volta do ano de 1769.
[3] 1996 para sermos mais exatos.

comunidade de IA ficou orfã de um grande problema, um novo desafio que pudesse, ao menos em parte, reunir novamente esforços em torno de uma nova meta.

O futebol, como campo de validação de teorias, parece ter ocupado este espaço deixado pelo jogo de xadrez. O motivo para isto é que o futebol é um jogo coletivo e, como tal, a participação de todos para que o objetivo principal (marcar o maior número de gols e vencer a partida) seja alcançado depende da boa atuação de cada um. É necessário que haja certa interação e comunicação entre os membros da mesma equipe para que se obtenha sucesso. Talentos individuais não são garantias de bons resultados; a palavra chave é *cooperação*. A escolha do futebol foi feliz em muitos aspectos pois ele representa um desafio para diversas áreas, não só para a computação como no caso do xadrez (Noda et al., 1999).

Para a engenharia elétrica, a contrução de dispositivos elétricos/eletrônicos resistentes a choques mecânicos, de pequenas dimensões e baixo consumo bem como o desenvolvimento e aprimoramento de sensores e transdutores. Para a automação e robótica, o desenvolvimento de robôs ágeis, leves e de fácil controle. Para a computação, o desenvolvimento de sistemas mutiagentes e sistemas de tempo-real. Enfim, é um vasto campo multidisciplinar onde pode-se testar e comparar abordagens a um custo relativamente baixo. Trata-se realmente de um campo de provas onde pesquisas são realizadas e conclusões extrapoladas para aplicações reais em benefício da sociedade.

4.2 Organização

O interesse ao redor do tema atingiu proporções mundiais, a ponto de serem criadas federações para organizarem *workshops* e outros eventos de cunho científico onde são apresentadas as inovações e propostas de cada equipe bem como campeonatos reais - e por vez divertidos - onde pode-se comprovar na prática as melhores estratégias e as táticas usadas pelas equipes.

Existem atualmente duas federações internacionais envolvidas em pesquisas com futebol robótico; a *Robot World Cup Soccer Games and Conferences (RoboCup)* e a *Federation of International Robot-soccer Association (FIRA)*. Cada uma delas possui seus próprios regulamentos e categorias como veremos nas seções 4.4 e 4.3. A mais notável diferença entre estas duas federações está no enfoque. Enquanto a *RoboCup* está mais empenhada em fomentar pesquisas na área de IA, dando grande ênfase às inovações e ao aspecto teórico, a *FIRA* está mais voltada à robótica em si e nas tecnologias de construção dos robôs. São abordagens diferentes com importantes influências nos resultados.

Há também a participação de grandes empresas do setor de tecnologia. Elas foram atraídas pelo enorme leque de oportunidades que pesquisas como estas representam para o futuro e hoje são patrocinadoras destes eventos.

Figura 4.1: Logo da *RoboCup*

Figura 4.2: Logo da *FIRA*

4.3 RoboCup

4.3.1 Breve Histórico

A idéia do futebol robótico, ao que tudo indica, foi mencionada pela primeira vez pelo professor Alan Mackworth da Universidade British Columbia, Canadá. Mas, independentemente dele, um grupo de pesquisadores japoneses organizaram um *workshop* sobre os grandes desafios da Inteligência Artificial em outubro de 1992, em Tóquio. Este evento desencadeou uma série de discussões sobre como utilizar o jogo de futebol para fomentar a pesquisa em ciência e a tecnologia. Foi feita então uma investigação sobre a viabilidade tecnológica, financeira e o impacto social. Juntamente, foram desenvolvidas regras e protótipos de robôs-jogadores e simuladores. Como resultado destes estudos, concluiram que o projeto era possível e desejável. Assim, em junho de 1993, Minoru Asada, Yasuo Kuniyoshi e Hiroaki Kitano decidiram iniciar uma competição, primeiramente com o nome de *Robot J-League* (O "J" deve-se a *japanese*). Esta acabou se transformando na primeira liga profissional de futebol robótico do mundo. Logo a seguir, devido a pedidos de pesquisadores de todo mundo também interessados, mudou-se o nome da liga e do projeto como um todo para *Robot World Cup Initiative*, ou simplesmente *RoboCup*.

Paralelamente a estas discussões, outros pesquisadores já haviam tentado usar o jogo de futebol como um campo de pesquisas. Por exemplo, Itsuki Noda (Kitano et al., 1997) do laboratório *Electro-Technical Laboratory (ETL)* estava conduzindo pesquisas com sistemas multiagentes experimentando com o futebol, iniciou o desenvolvimento de um simulador (seção C). Este simulador veio a se tornar oficial nos jogos de robôs simulados promovidos pela *RoboCup*. Também na mesma época, a professora Manuela Veloso e seu aluno de doutorado Peter Stone da Universidade Carnegie Mellon, EUA, conduziam pesquisas envolvendo o jogo de futebol robótico e sistemas multiagentes. Tanto a equipe de agente de *software* quanto a de robôs reais foram campeãs mundias da *RoboCup* e, ainda hoje, CMU é referência nesta área.

Em setembro de 1993, o primeiro anúncio público foi feito e regras específicas foram desenvolvidas. Ao mesmo tempo, a equipe de Itsuki Noda anunciava a versão 0 do sistema SoccerServer (escrito

em LISP), o primeiro simulador aberto que permitia pesquisas em futebol robótico e sistemas multi-agentes. A seguir veio a versão 1.0 escrita em C++ e distribuída juntamente com o código-fonte via Internet. A primeira demonstração pública do simulador aconteceu na *International Joint Conference on Artificial Intelligence* de 1995. Durante a IJCAI'95 em Montreal, Canadá, foi feito o anúncio de se fazer o primeiro campeonato mundial juntamente com a IJCAI'97 em Nagoya no Japão. Decidiu-se organizar uma Pré-*RoboCup'96* para identificar o principais problemas. Isto proporcionou tempo suficiente para o desenvolvimente e preparação dos primeiros times.

A Pré-*RoboCup'96* aconteceu durante a *International Conference on Intelligence Robotics and Systems (IROS'96)* em Osaka, Japão. Oito equipes competiram na liga de simuladores. Houve também demonstrações na liga de robôs de médio porte. Esta competição ficou marcada como sendo a primeira a utilizar o futebol robótico para promover o ensino e a pesquisa na área.

A primeira *RoboCup* oficial aconteceu como planejado em 1997 com grande sucesso. Mais de quarenta equipes participaram - entre times simulados e reais - e um público de mais de cinco mil expectadores compareceu.

Desde então, tanto o número de inscrições de times quanto de expectadores cresceu enormemente. O evento tornou-se cada vez mais internacional com a participação de vários países, inclusive do Brasil.

O mais ambicioso objetivo da *RoboCup* pode ser resumido em uma única frase:

> Por volta do ano de 2050, desenvolver uma equipe de robôs humanóides totalmente autônoma capaz de vencer o atual campeão mundial de futebol humano.

4.3.2 Categorias

As competições na *RoboCup* são divididas em ligas ou categorias. Cada uma destas ligas possui seus próprios regulamentos. Busca-se com esta medida diversificar as atividades de pesquisa, englobando diversas áreas e mantendo a multidisciplinaridade. Há também os comitês que organizam cada um dos eventos e decidem sobre os rumos das pesquisas. Por exemplo, na categoria de simuladores, exitem três comitês. O papel desempenhado por eles é o seguinte:

- Comitê Técnico: Decide sobre os rumos da liga de simuladores. O papel deste comitê é servir de ponte entre os pesquisadores desta categoria e os membros do corpo técnico. Toda discussão se dá através de uma lista de discussão dedicada. Através dela, questões como regras e procedimentos são debatidos com o intuito de melhorar as pesquisas. Até mesmo a política de desenvolvimento do programa simulador é discutido aqui.

- Comitê Organizacional: Gerencia as competições anuais da RoboCup. Decide quais modificações devem ser feitas. Estas mudanças devem partir exclusivamente dos membros do comitê técnico. Detalhes como parâmetros de jogo e esquemas de avaliação também são de competência do comitê organizacional.

- Comitê de Manutenção: Responsável por manter e estender o programa simulador utilizado nas pesquisas. Está ligado diretamente ao comitê técnico. Apenas pequenas mudanças no código sem impacto no todo podem ser feitas sem o aval do comitê técnico.

A próxima edição (de 2 a 11 de Julho de 2003 em Padua, Itália) contará com a participação de equipes nas seguinte categorias:

- Liga de simuladores (agentes de *software*).
- Liga de robôs de pequeno porte, tecnicamente chamada de $F180$ ($180cm^2$ de área máxima da base).
- Liga de robôs de médio porte.
- Liga de robôs quadrúpedes.
- Liga de robôs humanóides.

4.3.3 Liga de Simuladores

Esta é a liga dedicada aos agentes de *software*. Nesta categoria, programas de computador simulam robôs reais numa partida virtual. Cada jogador de cada equipe conecta-se a um programa servidor (Soccerserver) para receber e enviar comandos segundo um protocolo específico.

Devido à inexistência de complicadores físicos, esta categoria é a preferida por pesquisadores da IA interessados em aspectos mais teóricos. Sistemas multiagentes, estratégias de cooperação e aprendizado, por exemplo, estão entre os tópicos estudados.

A disputa entre as equipes se dá num ambiente muito semelhante ao real. Cada equipe é composta por onze "programas-robôs". As partidas duram em torno de cinco minutos e as regras são adaptadas para atender as peculiaridades desta liga. O apêndice C apresenta o simulador utilizado nestas pesquisas bem como as regras e regulamentos.

4.3.4 Liga de Robôs de Pequeno Porte

Categoria dedicada às disputas de pequenos robôs reais. Esta é a porta de entrada para pesquisas mais avançadas que envolvem, além da programação, outros aspectos mais ligados às engenharias. Cada time é livre para desenvolver seus próprios robôs, segundo regras específicas. A disputa se dá com cinco robôs por equipe, podendo um deles ser adaptado para a função de goleiro. O campo da partida é construído para este fim, com medidas e adaptações especiais (Games e Conferences, 2002).

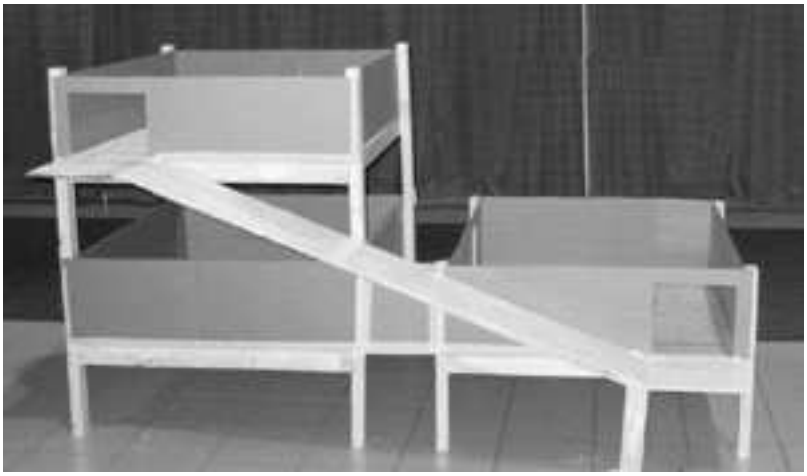

Figura 4.3: Estrutura utilizada pela *RoboCupJunior Rescue* para simulação de busca e resgate

4.3.5 Outras Pesquisas

A *RoboCup Federation* expandiu seus horizontes além destas já tradicionais pesquisas com futebol robótico. Hoje ela conta também duas outras áreas de pesquisa, a *RoboCupJunior* e a *RoboCupRescue*.

A *RoboCupJunior* é um projeto que objetiva despertar em crianças e jovens estudantes o gosto pelas ciências e pela pesquisa acadêmica. Conceitos de eletrônica e robótica podem ser aprendidos por elas de forma divertida e sem os rigores teóricos. O trabalho em equipe, cada vez mais importante nos dias de hoje, é incentivado. São três os desafios propostos:

1. Futebol: times de dois robôs móveis autônomos se enfrentam num campo retangular fechado. A diferenciação das equipes se dá com o uso de cores.

2. Dança: interessante desafio onde um ou mais robôs dançam e exibem coreografias ao ritmo de uma música. Vence quem for mais criativo e original em sua apresentação.

3. Resgate: robôs são utilizados para encontrar o mais rápido possível vítimas de desastres artificialmente provocados. O objetivo é encontrar passagem entre os obstáculos e com isso chegar em locais de difícil acesso (ver figura 4.3).

Em (Federation, 2003a) há dicas de construção de robôs, fotos de competições passadas, regras, vídeos e muitas outras informações.

O outro domínio de estudo, *RoboCupRescue*, é um projeto novo motivado por catástrofes naturais, como o terremoto que abalou a cidade de Kobe no Japão. Assim como no futebol, problemas desta ordem exigem cooperação para encontrar vítimas, navegação em ambientes desconhecidos e ruidosos, planejamento, etc. Trata-se de um tremendo problema a ser resolvido cujas implicações sociais são

muito grandes. Assim como na *RoboCup*, aqui também existe pesquisas com simuladores e robôs reais (Federation, 2003b; Tadokoro et al., 2000a; Tadokoro et al., 2000b; Kitano, 2000).

4.4 FIRA

4.4.1 Histórico e Categorias

A *FIRA* surgiu a partir de um comitê internacional formado sob a liderança do professor Jong-Hwan Kim com o objetivo de promover o *Micro-Robot World Cup Soccer Tournament (MiroSot)*. Assim, um encontro foi marcado e em 29 de julho de 1996 regras e regulamentos foram criados. Trinta times de treze países aceitaram o desafio.

A primeira *MiroSot'96* aconteceu em KAIST (*Korea Advanced Institute of Science and Technology*), entre 9 e 12 de novembro do mesmo ano. Esta primeira edição contou com a participação de vinte e três equipes de dez países diferentes. A equipe Newton, do Laboratório de Pesquisas Newton foi a vencedora, seguido por SOTY da Coréia. A equipe do CMU foi a campeã na categoria *S-MiroSot*, onde um partida é disputada por apenas dois robôs, um por equipe.

Assim como a *RoboCup*, o objetivo principal da *FIRA* é o desenvolvimento tecnológico, especialmente na área da robótica e visão computacional. Os eventos promovidos pela *FIRA* continuam ano após ano, sediando competições em vários países do mundo, inclusive no Brasil. Novamente aqui são agregadas novas categorias a cada edição do evento, comprovando os avanços obtidos desde o início das atividades desta federação.

A *FIRA* está dividida nas seguintes ligas:

- NaroSot (*Nano Robot World Cup Soccer Tournament*): Partida jogada por duas equipes, cada uma composta por cinco robôs sendo que um deles pode ser o goleiro. Apenas três humanos (um gerente, um técnico e um treinador) podem permanecer na área reservada ao jogo. Não mais que um computador por equipe pode ser usado para o processamento da imagem, identificação e localização. Cada robô possui as seguintes dimensões: $4cm\ x\ 4cm\ x\ 5,5cm$.

- MiroSot (*Micro Robot World Cup Soccer Tournament*): Nesta categoria uma partida é jogada por duas equipes compostas de três robôs, sendo um deles o goleiro. Os robôs não deverão exceder as seguintes medidas: $7,5cm\ x\ 7,5cm\ x\ 7,5cm$. As regras restantes são as mesmas da NaroSot.

- RoboSot (*Autonomous Robot-soccer Tournament*): Equipes compostas de um à três robôs. Estes podem ser autônomos ou semi-autônomos. Cada robô possui as dimensões: $20cm\ x\ 20cm\ x\ 40cm$.

- HuroSot (*Humanoid Robot-soccer Tournament*): O robô humanóde deverá ter apenas duas pernas (bípede). O robô não deverá ter mais que $40cm$ de altura e $15cm$ de diâmetro total, ou seja,

as pernas não deverão exceder o espaço determinado por este círculo. Por ser uma categoria nova, ainda não existem regras definidas. Exibições são feitas mais a título de demonstração.

- KheperaSot (*Khepera Robot-soccer Tournament*): Nesta categoria, equipes são compostas compostas por um robô Khepera. A participação humana esta restrita a duas pessoas por time na área de jogo. Os robôs devem ser totalmente autônomos. Desta forma, o processamento das imagens deve ser feito inteiramente *on-board*.
- QuadroSot (*Quadruped Robot-soccer Tournament*): Categoria recém criada que utiliza robôs quadrúpedes. Utilizam, em sua maioria, robôs desenvolvidos pela Sony. São os famosos cães-robô.
- SimuroSot (*Simulated Robot-soccer Tournament*): Categoria dedicada ao desenvolvimento de programas clientes. Cada equipe codifica suas estratégias para serem confrontadas com a ajuda de um simulador, onde atuam de cinco a onze programas-jogadores. A grande vantagem neste caso é a ausência total de *hardware*. Pesquisas podem ser feitas sem grandes custos envolvidos.

4.5 Tecnologias

Talvez a maior vantagem do futebol robótico seja a capacidade de integrar conceitos e tecnologias, dentre elas destacamos:

- Inteligência Artificial
- Sistemas de Tempo-Real
- Visão computacional
- Eletrônica
- Mecânica fina

4.6 Aplicações

A seguir são citadas algumas aplicações práticas que estas pesquisas proporcionam:

- Semáforos inteligentes que se comunicariam entre si para desobstruir regiões inteiras de uma cidade e não apenas cruzamentos (Nunes, 2002).
- Veículos autônomos para a exploração de ambientes perigosos e desconhecidos, como robôs de exploração interplanetária (Zlot et al., 2002).
- Automatização das subestações de transmissão de energia elétrica com o objetivo de reduzir o tempo de retorno à normalidade.

Capítulo 5

Projeto UFSC-Team

5.1 Histórico e Objetivos

O Projeto UFSC-Team teve início no final do ano de 97, apenas um ano após a realização do primeiro evento mundial de futebol robótico realizado em 1996. Inicialmente contando com a participação de um aluno de doutorado e um professor, evoluiu para conter também alunos de iniciação científica e mestrado. Tornou-se então um importante campo de estudos para a validação de pesquisas principalmente para interessados em Inteligência Artificial. A primeira e principal meta deste projeto era desenvolver um equipe para participar das competições de futebol robótico na categoria simuladores e, talvez mais tarde também participar com robôs reais de pequeno porte.

Após alguns meses de trabalho, em 1998 o primeiro time de futebol capaz de participar da *RoboCup* na categoria simuladores ficou pronto. Um artigo descrevendo (da Costa e Bittencourt, 1998) as inovações e a proposta foi apresentado no *workshop* da *RoboCup*. Este evento acontece em paralelo às competições e serve para promover e incentivar as pesquisas nesta área. Também é o lugar onde são expostas e apresentadas as inovações de cada time. Antes de mais nada, este é um evento científico. Apesar do "ar" de descontração e brincadeira proporcionado pelo futebol, pesquisas sérias são feitas. Mais do que vencer uma partida, o importante é inovar, integrar conceitos e tecnologias.

Esta primeira participação evidenciou tanto as virtudes quanto os defeitos do time e também serviu de base para as versões seguintes que se sucederam. As seções seguintes explicitarão as duas arquiteturas criadas para o time simulado de futebol de robôs.

5.2 Proposta

A proposta do projeto UFSC-Team é utilizar técnicas de Inteligência Artificial no desenvolvimento do time de futebol robótico. Sistemas multi-agentes e lógica nebulosa são as principais técnicas utilizadas. Mecanismos de cooperação, comunicação e planejamento são implementados; requisitos fundamentais num jogo essencialmente coletivo como o futebol.

Hoje, este projeto está servindo como plataforma de estudo para diversas outras pesquisas em IA, como otimização de parâmetros de controladores nebulosos utilizando algorítmos genéticos (Gonçalves, 2001) e aprendizado de máquina. Percebe-se o enorme potencial que este tópico futebol robótico representa. É completo e permite uma gama bastante grande de opções, podendo ser utilizado como verdadeiro laboratório por diversos cursos.

5.3 Equipe de Agentes de *Software*

As pesquisas com agentes de *software* tiveram como resultado a criação de duas arquiteturas de agentes, cada uma delas deu origem a um time simulado diferente. As duas subseções a seguir apresentam com maior riqueza de detalhes as duas arquiteturas desenvolvidas.

5.3.1 UFSC-Team'98

Esta primeira versão desenvolvida propunha uma arquitetura de agente cognitivo. A idéia principal dessa primeira arquitetura era implementar percepção, ação, comunicação, cooperação, planejamento e tomada de decisão utilizando a programação concorrente (da Costa e Bittencourt, 1998).

Essa primeira arquitetura concorrente baseava-se em três processos: Interface, Coordinator e Expert. O processo Interface foi projetado para manipular a percepção e a ação. A interação entre agente e ambiente suportada pelo SoccerServer consiste na troca de mensagens através de *sockets* domínio Internet (*socket* Inet).

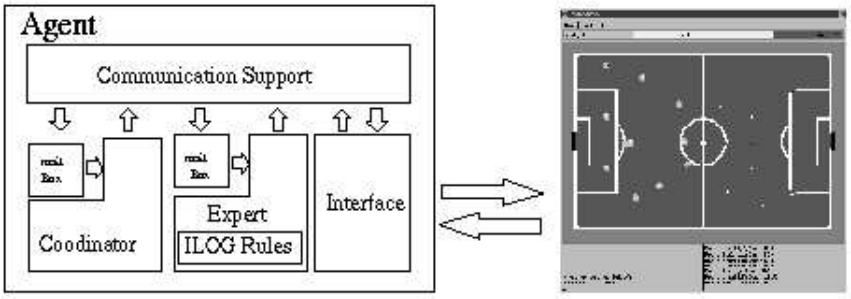

Figura 5.1: Arquitetura concorrente do agente do UFSC-Team'98

A função do processo Interface era inicialmente converter as informações visuais recebidas do SoccerServer (percepção) e as mensagens recebidas do juiz e dos demais agentes (comunicação) na linguagem Parla[1] (da Costa, 1997).

O processo Coordinator responsabiliza-se pela comunicação, pela abertura e pelo gerenciamento dos processos de cooperação entre os agentes do UFSC-Team. Baseado na arquitetura original proposta no ambiente ExpertCoop (da Costa, 1996), esse processo era responsável pelo gerenciamento

[1]Linguagem para comunicação de agente utilizada pelo UFSC-Team.

da comunicação entre os agentes (este processo recebia todas as mensagens provenientes dos demais agentes e as manipulava). Entretanto, de acordo com as regras da categoria simuladores da *RoboCup*, toda a comunicação entre os agentes deve ser efetuada através do SoccerServer. Sendo assim, tanto a percepção do agente quanto as mensagens de comunicação entre os agentes do time são recebidas pelo mesmo *socket* no domínio Internet. Conseqüêntemente, nesta implementação, o processo Interface recebe as mensagens trocadas entre os agentes e as mensagens enviadas pelo juiz, redirecionandoas para o processo Coordinator onde são manipuladas.

Finalmente, o processo Expert era responsável pelo planejamento e pelo processo decisório do agente. Possui um sistema baseado em conhecimento encapsulado no qual a percepção, as mensagens enviadas pelo juiz e as mensagens enviadas pelos demais agentes do UFSC-Team eram armazenadas e utilizadas para inferir as decisões apropriadas, de acordo com as regras do sistema baseado em conhecimento. Estes três processos comunicavam-se através dos *sockets* no domínio do Unix.

Essa primeira implementação concorrente, com o processo decisório do agente centralizado, apresentou alguns problemas de sincronização entre o agente e o ambiente, e a resposta em tempo real não foi tão rápida quanto se esperava. Realmente, a melhor resposta em tempo real apresentada pelos agentes do UFSC-Team'98 ficou entre 70 e 80 ms. Uma hipótese para este fato talvez seja a utilização de uma ferramenta externa para a construção das regras. O sistema baseado em conhecimento, responsável pela tomada de decisão tornou-se complexo. Esse sistema possuía regras destinadas a tratar todo tipo de informação, inclusive de alto nível, como por exemplo que tipo de jogada ensaiada a ser utilizada.

5.3.2 UFSC-Team'99

Esta segunda versão do time simulado baseava-se numa arquitetura alternativa proposta em (da Costa e Bittencourt, 1999). Devido aos problemas de *timing* e sincronização, migrou-se de uma arquitetura de agente concorrente para outra arquitetura de agentes autônomos concorrente.

Esta arquitetura é composta por três processos: Interface, Coordinator e Expert. Cada um deles estava associado a um nível decisório: Reativo, Instintivo e Cognitivo.

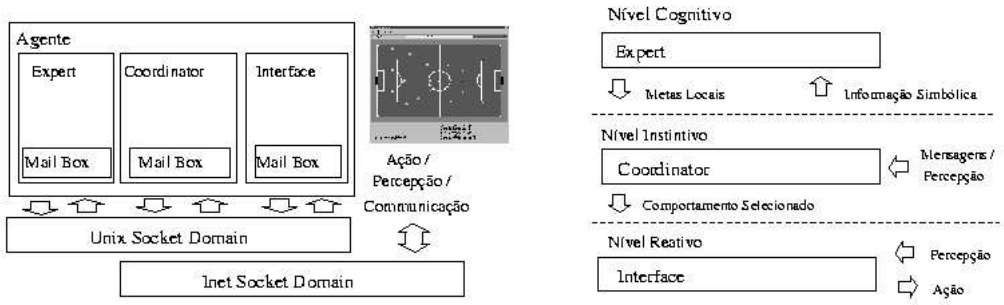

Figura 5.2: Arquitetura de agente autônomo concorrente do UFSC-Team'99

O nível reativo é implementado no processo Interface e é responsável pela resposta em tempo real do agente, isto é, recebendo a informação sobre a percepção e enviando os comandos de ação adequados ao simulador. Ele consiste de um conjunto de controladores nebulosos. A cada momento somente um controlador nebuloso está ativo e este decide quais ações imediatas deverão ser tomadas. Esta escolhoa é baseada na informação recebida do simulador e é determinada pelas regras do controlador nebuloso ativo. Cada um dos controladores nebulosos disponíveis no agente representam um comportamente específico e tem algumas condições associadas que determinam as situações nas quais ele é efetivo. Por exemplo, um agente-jogador de defesa não precisa, necessariamente, ter a habilidade de chutar com precisão. Já para um agente-jogador de ataque isto é fundamental, além de ser rápido e ágil. Um agente-jogador goleiro precisa apenas ter habilidade com as "mãos". Estas diferenças são conseguidas com ajustes da quantidade e tipo dos controladores nebulosos.

O nível instintivo é implementado no processo Coordinator e é responsável por: identificar o estado em que a partida, atualizar a informação simbólica usada pelo nível cognitivo e escolher o comportamento reativo mais adequado dada a meta local e o estado em que se encontra a partida.

Uma meta local, gerada pelo nível cognitivo, é mantida neste nível decisório. Esta meta pode ser alcançada pela execução de uma determinada seqüência de comportamentos reativos. A cada nova informação visual ou mensagem recebida, um sistema especialista de um ciclo identifica o estado em que se encontra o jogo, gera a informação simbólica utilizada pelo nível cognitivo e verifica se o comportamento reativo atual é o mais adequado, dado o estado do jogo e a meta local. Caso não seja, um novo comportamento reativo é escolhido.

Finalmente, o nível cognitivo é implementado no processo Expert e é responsável por determinar as metas locais e globais. O nível cognitivo não tem um efeito direto sobre o nível reativo, ele somente estabelece a meta atual e a passa para o nível instintivo. Enquanto a meta não for alcançada ou não falhar, o nível cognitivo não interfere no nível instintivo. Este tempo ocioso é usado para planejar metas futuras e para especificar requisições de cooperação, visando alcançar metas globais. O nível cognitivo também é implementado através de um sistema especialista simbólico, cuja complexidade é maior do que o do nível instintivo, uma vez que o tempo de resposta para este nível não apresenta as mesmas restrições. As tabelas 5.1 e 5.2 resumem as atribuições de cada um dos níveis decisórios.

Metas globais são representadas por jogadas e, estas por sua vez são compostas por metas locais. Metas locais se desdobram em comportamentos, como por exemplo **encontrar-bola**, **conduzir-bola** e **driblar**. Finalmente, cada um destes "papéis" ou comportamentos encapsulam controladores nebulosos. A meta local **encontrar-bola**, por exemplo, é composta pelos controladores **watch_ball** e **move_to_direction**.

5.4 Equipe de Robôs Reais de Pequeno Porte

Após algum tempo de pesquisas com simuladores e agentes de software, foram adquiridos três robôs da plataforma *EyeBot* para iniciar as pesquisas com robôs reais. Este é o primeiro trabalho

- Identifica e seleciona a meta global
- Meta global $\equiv$ jogada
- As metas globais de um lado do campo serão espelhadas no outro longitudinalmente
  lado esquerdo $\rightarrow$ par (2,4,6,7)
  lado direito $\rightarrow$ ímpar (1,3,5,7)
- Jogada * é subdividida em *n* metas locais

Tabela 5.1: Atribuições do nível decisório cognitivo

- Recebe do nível cognitivo as metas locais
- Responsável pela geração das variáveis de estado
- Responsável por escolher o próximo controlador ativo
- Colhe informações do simulador, determina as **variáveis de estado** e as envia ao nível cognitivo
- Meta local $\equiv$ "papel", comportamento
- Meta local * é composta por *n* controladores nebulosos

Tabela 5.2: Atribuições do nível decisório instintivo

realizado tendo como foco principal estes robôs reais.

Os principais objetivos são estudar e programar estes robôs da plataforma EyeBot e portar parte do código do agente autônomo concorrente para a plataforma móvel. O capítulo 6 apresenta esta análise enquanto os resultados dos testes feitos são mostrado em 7.

Capítulo 6

A Plataforma EyeBot

6.1 Apresentação

EyeBot é o nome dado à família de robôs desenvolvidos e comercializados pelo Departamento de Engenharia Elétrica e Computação da Universidade Western Australia. Sob a supervisão do professor Thomas Bräunl, são projetados controladores[1] para equipar igualmente todos os modelos de robôs que compõem a plataforma *EyeBot* (figura 6.1).

O fato de todos membros da família *EyeBot* utilizarem o mesmo controlador permite uma flexibilidade bastante grande pois é possível desenvolver programas para todos os modelos da plataforma sem grandes mudanças. Também reduz os custos de projeto do *hardware* e *software*, fator crítico para equipes pequenas de desenvolvimento.

Neste projeto há também a participação de outras universidades, como as alemãs (*Kaiserslautern* e *Stuttgart*), neo-zelandesas (*Auckland*) e americanas (*Rochester Institute of Technology*).

A comercialização destes robôs é feita por empresas representantes espalhados pelos cinco continentes. São elas:

- Europa: AndroTec, Alemanha, `http://www.eyebot.de`
- América do Norte: Zagros Robotics, EUA, `http://www.zagrosrobotics.com`
- Austrália e Ásia: Robot OZ, Austrália, `http://www.robotoz.com.au`
- Outros países: Joker Robotics, `http://www.joker-robotics.com`

Diversos grupos de pesquisa ao redor do mundo utilizam estes protótipos para pesquisas com robótica móvel. Eles representam uma alternativa de baixo custo para quem não pode construir um robô "do zero" ou então não tem interesse nisso e deseja apenas desenvolver seu trabalho e testar suas teorias sem preocupar-se com aspectos construtivos.

[1] Também chamados de EyeBot.

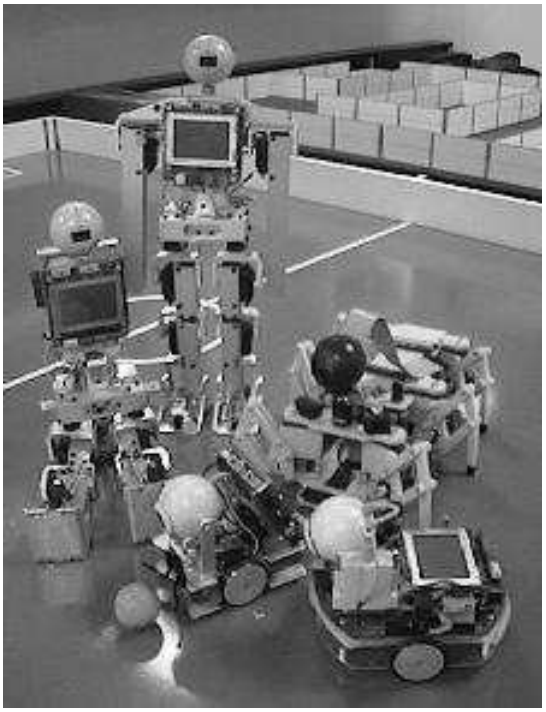

Figura 6.1: Família de robôs *EyeBot*

Figura 6.2: Imagem do LCD do controlador *EyeBot*

6.2 Descrição

EyeBot também é o nome dado ao controlador utilizado nestes robôs móveis. Trata-se de uma unidade de pequenas dimensões e baixo consumo de energia, equipada com um microcontrolador de 32 *bits MC68332* da Motorola, um display de LCD capaz de exibir gráficos de baixa resolução (e com apenas 1 *bit por pixel*), uma câmera colorida (24 *bits por pixel* e resolução de 80*x*60 *pixels*), portas seriais, paralelas, saídas digitais, saídas analógicas, etc. As características completas podem ser vistas na tabela 6.1. A figura 6.2 mostra uma imagem do *display* LCD deste controlador (Motorola, 1995; Motorola, 1996). A figura 6.4 apresenta um diagrama esquemático (diagrama de blocos) do controlador *EyeBot*.

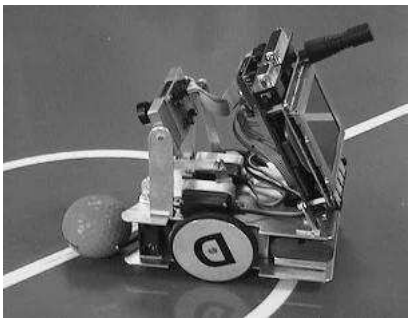

Figura 6.3: Robô *SoccerBot* utilizado nas pesquisas

Todos os modelos vendidos são compostos, em sua grande maioria, por peças facilmente encontradas em boas lojas de eletrônica ou modelismo. São componentes modulares que quando combina-

CPU	Motorola MC68332
Freqüência	35*MHz*
Memória	1*MB* de RAM e 512*kB* de Flash-ROM (sistema + programas usuário)
Portas	1 paralela e 2 serias
Outras	8 entradas e 8 saídas digitais + 8 entradas analógicas *speaker* para saída de áudio e microfone para entrada
Motores	2 motores DC acoplados às rodas 2 motores de passo (controle de mecanismo de chute e câmera)
Sensores	3 sensores de proximidade infra-vermelho (frontal, esquerdo e direito)
Visão	câmera CMOS colorida
Navegação	2 *encoders* acoplados a cada uma das rodas para *deduced reckoning*
Alimentação	1 bateria recarregável de Li-Ion $(7,2V)$

Tabela 6.1: Características do controlador *EyeBot*

dos integram-se perfeitamente. A excessão, é claro, fica por conta do *software* de controle do robô, ou seja, o sistema operacional embarcado chamado de RoBIOS. Todo este programa foi desenvolvido especialmente para esta plataforma e os usuários dela não tem acesso à estas rotinas de mais baixo nível, nem aos mecanismos básicos de funcionamento que por vezes são necessários. Isto faz com que freqüentemente tenhamos que tirar conclusões com base apenas em hipóteses. Não há como garantir nada pois esta máquina, do ponto de vista do usuário-pesquisador comporta-se como uma caixa preta.

6.2.1 Modelos

A família de robôs *EyeBot* compreende diversos modelos. Estão disponíveis exemplares com rodas como o *SoccerBot* e o *Omni*, versões bípedes com o *Android* e com pernas como o modelo *Walker*. Fotos de todos eles podem ser vistas no *site* oficial do projeto em `http://www.ee.uwa.edu.au/~braunl/eyebot/robots/index.html` Nossas pesquisas foram feitas com o modelo *SoccerBot* mostrado na figura 6.3. Em meados do ano de 2000 três exemplares da terceira geração (MK3) foram adquiridos. A meta era iniciar pesquisas também com robôs reais e, futuramente, participar de competições na categoria de pequeno porte da *RoboCup*. Mas, devido à alguns contra-tempos, decidiu-se mudar o foco do trabalho não visando mais as competições mas sim o desenvolvimento tecnológico de uma nova linha de pesquisa no departamento.

O *SoccerBot* é um robô projetado para atender as regras e regulamentos da *RoboCup Federation*. A versão utilizada tinha as seguintes características:

- Controlador *EyeBot*
- Dois motores-de-passo para o controle da câmera e do mecanismo de chute
- Dois motores DC acoplados a dois *encoders*
- Três sensores infra-vermelhos

- Uma câmera colorida chamada de *EyeCam*
- Uma bateria recarregável de Li-Ion
- Um módulo de rádio UHF com portadora sintonizada em 433MHz para comunicação remota com outros pontos (*rede EyeNet*)

Além destes periféricos já incorporados a máquina, outros mais, como por exemplo, mais sensores infra-vermelhos além dos três existentes podem ser adicionados sem que seja necessário grandes mudanças.

6.3 RoBIOS

RoBIOS é o nome do sistema operacional primitivo do robô. Este pequeno em tamanho mas fundamental programa é responsável por controlar todos equipamentos conectados ao robô, conforme a tabela de descrição de *hardware* (ver apêndice B) gravada em memória Flash-ROM. Um mapa completo da memória do robô é apresentado a seguir. Nota-se que portas e registradores são acessados utilizando o típico método de E/S mapeada em memória.

A programação se dá através da API disponibilizada pela RoBIOS. No endereço apresentado anteriormente existe um pequeno manual com todas as funções disponibilizadas. Isto, alias, é tudo que o programador terá para desenvolver seus programas. Existem funções para desde se imprimir caracteres no display até ajustar os parâmetros do controlador PI integrado.

Apesar de simples, consegue-se realizar praticamente tudo com a API apresentada. Ela é bastante variada e rica em funções. Durante este trabalho não tivemos problemas com relação ao desenvolvimento de programas em si.

O único ponto crítico com relação a RoBIOS fica por conta da falta de suporte a multiprogramação. Por vezes quizemos executar mais de um programa ao mesmo tempo na memória. A RoBIOS não prevê isto levando a programas maiores e mais complexos. A solução encontrada foi simular uma multiprogramação utilizando programação multi-*threads*, esta sim possível. Os resultados foram satisfatórios e, dependendo da aplicação, perfeitamente factíveis. Realizamos testes em que três programas foram transformados em *threads* e incorporados à um quarto, responsável apenas pelo chaveamento entre eles. Estas *threads* tinham a função de monitorar sensores infra-vermelhos, atuadores e câmera. Neste caso, ficou evidente a fragilidade do processador. Em vários momentos informações foram perdidas principalmente provenientes dos PSDs, prejudicando também a atuação dos motores DC que utilizam esta informação de distância em suas regras. Mas de um modo geral pode-se utilizar esta abordagem como forma de contornar esta limitação.

6.4 Utilização

A utilização destes robôs é bastante simples. A seqüência passo-a-passo que ilustra todo o processo é mostrada a seguir:

1. instala-se um compilador cruzado (mais conhecido por *cross-compiler*) na plataforma onde deseja-se trabalhar. Estão disponíveis versões pré-compiladas para PC rodando sistemas MS-Windows e GNU/Linux. Estes pacotes foram gerados tendo como plataforma alvo a arquitetura do robô, ou seja, o microcontrolador Motorola MC68332
2. instala-se nos locais apropriados os arquivos de cabeçalho e bibliotecas de ligação do sistema monitor do robô RoBIOS
3. com o auxílio de um editor de textos simples cria-se um programa em linguagem C ou então em linguagem de montagem
4. compila-se este programa utilizando o compilador cruzado e os arquivos relativos a RoBIOS
5. o resultado desta compilação, conhecido por código objeto, é um arquivo texto composto por caracteres hexadecimais chamado de **S-Record**
6. este arquivo com extenção *.hex deve ser carregado via porta serial para o robô e a seguir executado

Além desta operação de carga de programas, existe também outras possíveis. A atualização da RoBIOS gravada em Flash-ROM e a compilação e carga da HDT. Estes processos são semelhantes à este apresentado e são descritos em detalhes em `http://www.ee.uwa.edu/~braunl/eyebot/`.

6.5 Vantagens

Vantagens na utilização desta plataforma para pesquisas de robótica móvel e futebol robótico:

- Programação em alto nível na linguagem C
- Rotinas de mais baixo nível podem ser escritas em linguagem de montagem (*assembly*) do microcontrolador *MC68332*
- A câmera integrada permite pesquisas mais avançadas envolvendo visão computacional e tratamento de imagens
- Expansibilidade da plataforma com a adição de novos periféricos, como sensores, por exemplo
- *Display* de LCD monocromático

- Transferência de programas via porta serial
- Módulo de rádio para comunicação sem fio
- Compatível com PC e estações de trabalho UNIX
- Ferramentas de programação livres e disponíveis via Internet[2]

O usuário também tem a disposição diagramas e informações completas a respeito do funcionamento de cada umas das partes do robô. Se isto não for suficiente, existe ainda a lista de discussão (`http://maillists.uwa.edu.au/mailman/listinfo/eyebot`) para esclarecer qualquer dúvida que possa aparecer.

6.6 Desvantagens

A grande desvantagem desta plataforma é fato de ser proprietária e semi-fechada. Não temos acesso, por exemplo, ao código fonte da **RoBIOS**, o sistema monitor do robô. Realmente estas informações não são imprescindíveis para o desenvolvimento de programas usuários mas, são importantes para quem se propõe a estudar a fundo a plataforma alvo com o intuito de desenvolver algo o mais otimizado possível.

Outro ponto negativo fica por conta da câmera incapaz de mostrar e reproduzir com qualidade as imagens capturadas, como será mostrado no capítulo seguinte; e o processador que, para tarefas mais exigentes provou não ser capaz de suportar o trabalho (Berry, 1999).

6.7 Exemplos

Durante a fase inicial deste trabalho, buscamos desenvolver programas simples que ilustrassem melhor o funcionamento de cada uma das partes. Desta forma cada sub-sistema do robô pode ser explorado e entendido em sua totalidade. Mostramos a seguir alguns trechos de código que ilustra a programação em linguagem C destes robôs (códigos 1 e 2).

[2] O *download* destas e outras ferramentas úteis pode ser feito em `http://robotics.ee.uwa.edu.au/eyebot/ftp`.

Código 1 Exibe no *display* imagens capturadas pela câmera

```
#include "eyebot.h"

int main (void)
{
        int camera;
        image mono_image;
        colimage color_image;

        camera = CAMInit(NORMAL);
        CAMSet(30,0,0);
        CAMMode(AUTOBRIGHTNESS);

        LCDPrintf("EyeCam model: %d\n",camera);
        AUBeep();
        OSSleep(200);

        LCDMode(NONSCROLLING|NOCURSOR);
        LCDClear();
        LCDMenu(" ", " ", " ", "Fim");

        do {
                CAMGetColFrame(&color_image,0);
                IPColor2Grey(&color_image,&mono_image);
                LCDPutGraphic(&mono_image);
        } while (KEYRead() != KEY4);

        CAMRelease();

        return 0;
}
```

Código 2 Envia dados do robô para o PC via rádio

```
        if (RADIOInit() != 0) {
                printf("Erro!\n");
                return 1;
        }
        else printf("Radio Link Ok!\n");

        while (TRUE) {
                printf("comando> ");
                scanf("%s",&mensagem);
                tamanho = strlen(&mensagem);
                if (tamanho >= 20) {
                        printf("\nErro! Limite de 20 caracteres excedido\n");
                        return 1;
                }
                erro = RADIOSend(destino1,tamanho,mensagem);
                if (erro != 0) {
                        printf("Erro!\n");
                        return 1;
```

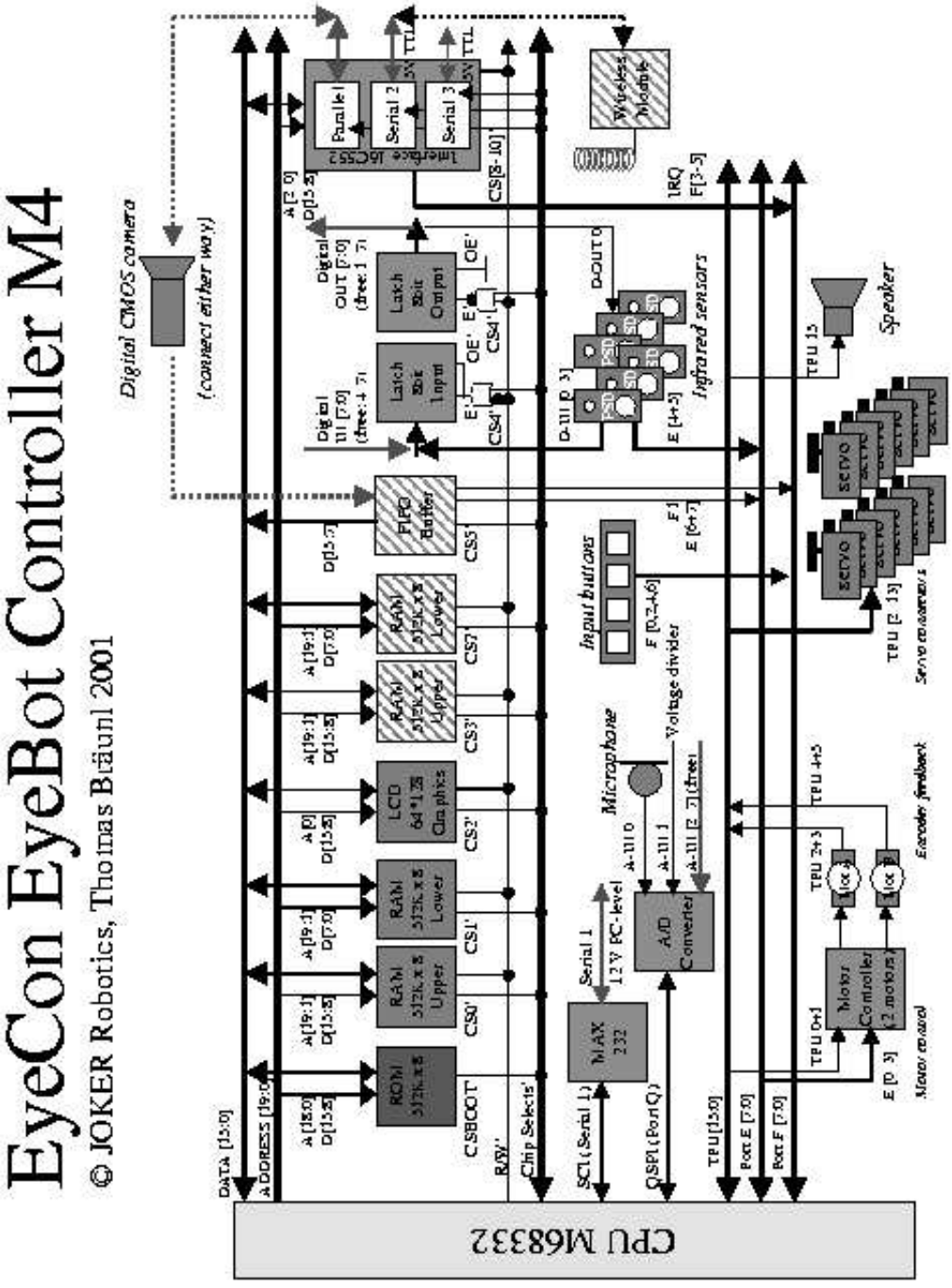

Figura 6.4: Diagrama esquemático do controlador do robô

```
0x00000000  128KB Área da RoBIOS
0x00020000  Até 2MB de memória RAM de usuário (CS0,1,3,7)
0x00200000  Fim da RAM
..........  endereço não utilizado
0x00a00000  Base TPU 2KB
0x00a00800  Fim da base TPU
..........  endereço não utilizado
0x00c00000  512KB Flash (CS2)
0x00c80000  Fim da Flash
..........  endereço não utilizado
0x00e00800  Latches  (CS4)
0x00e01000  FIFO ou Latches (CS5)
0x00e01800  Porta paralela/Câmera (CS8)
0x00e02000  Segunda porta serial (CS9)
0x00e02800  Terceira porta serial (CS10)
..........  endereço não utilizado
0x00fff000  Registradores internos do MC68332 (4KB)
0x01000000  Fim dos registradores e RAM endereçável
```

Figura 6.5: Mapa de memória do robô

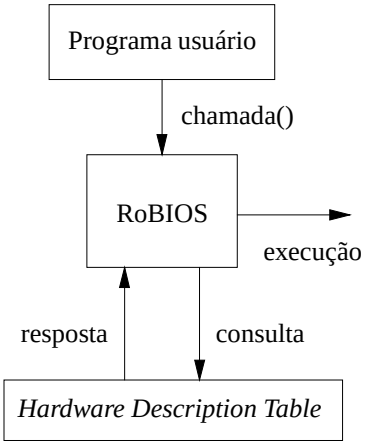

Figura 6.6: Fluxo de informação programa usuário-RoBIOS-HDT

```
S00E000045796556696577 2E686578C0
S31A000200004EF90002000A0002622423CF0002621C23FC00026510
S31A00020015A8000262200CB900020000000262206E0000142E7C2B
S31A0002002A0001FFF821FC0000000003C460000012203803C051FF
S31A0002003F802E4021FC000265A803C44E56FFF8223C00026590D3
S31A000200205 4203C000265A8B2806710204190814218 51C8FFFC4259
S31A0002006940538064F44878000048790002621848 6F0004487897
```

Figura 6.7: Trecho de um arquivo texto formato S-Record

6.8 Contribuições ao projeto

A maior contribuição ao projeto que mantém esta plataforma foi a criação de um programa para transmissão de imagens utilizando o módulo de rádio em ambiente UNIX. O programa foi criado utilizando-se a linguagem C (ANSI C) e a API de mais baixo nível de programação do sistema gráfico padrão dos Unices[3] *Xlib*. Com isto, tornamos este pequeno aplicativo muito mais portável que a versão existente que apenas "roda" em ambiente Windows.

Também escrevemos diversos outros programas, cada um deles explorando características e aspectos diferentes do robô. Futuramente, quando outros alunos iniciarem suas pesquisas com esta plataforma, já existirá uma base formada. Tempo será poupado e energias não serão disperdiçadas pois certo *know-how* foi adquirido o trabalho tenderá à evoluir mais rapidamente de onde paramos.

A figura 6.8 mostra quatro imagens capturadas pela câmera com a ajuda do programa *Xcameraviewer*. Este estará em breve disponível para *download* no próprio *site* oficial do projeto.

Figura 6.8: Algumas imagens em tamanho natural tiradas com o auxílio do programa Xcameraviewer

[3]Plural de UNIX.

Capítulo 7

Resultados

7.1 Testes com a câmera *on-board*

O objetivo principal dos testes com a câmera embarcada *EyeCam* do robô foi verificar a fidelidade com que as cores são captadas. Esta câmera é capaz de capturar imagens coloridas com resolução de $82x62 pixels$ e profundidade de cores de $24bps$, 8 para cada componente de cor[1]. Este teste é fundamental para a identificação e conseqüênte tomada de decisão dos objetos presentes no campo visual do robô. É com base nesta informação que os algoritmos de mais alto nível planejam as suas ações.

Os testes foram feitos com uma variação do programa *Xcameraviewer* (ver apêndice A). Ao invés das imagens serem transmitidas para serem exibidas numa janela gráfica, as magnitudes dos *pixels* em tons de cinza são transmitidos. Em frente à câmera foram colocadas imagens de cores totalmente homogêneas de uma única cor. A seguir, utilizando o enlace de rádio toda esta matriz de pontos (5084 pontos no total) foi transmitida. Alguns dos gráficos obtidos são mostrados a seguir. Todos os gráficos a seguir foram feitos utilizando uma imagem totalmente negra; ou seja, todos os valores dos pontos deveriam ser nulos[2].

As figuras 7.1 e 7.2 mostram no eixo das ordenadas a magnitude dos pontos e no eixo das abscissas o índice de cada um destes pontos. Esperava-se como resultado valores nulos ou muito próximos disto mas, para nossa surpresa, a quase totalidade dos pontos se concentrou na faixa que vai de zero a cinqüênta. Além disso, o desvio padrão também foi alto devido a diversos pontos fora desta faixa. As conseqüências deste fato são muito grandes! De imediato restringe, senão impossibilita, a identificação e o posterior tratamento da informação com base na magnitude dos *pixels*. Segundo, torna muito difícil qualquer "raciocínio" que utilize com base o sentido da visão do robô. Estes resultados apresentados nos gráficos 7.1 e 7.2 impossibilitaram que o código desenvolvido para a categoria de simuladores fosse portada para esta classe de robôs de pequeno porte, como era o desejo

[1] Vermelho,Verde e Azul (RGB em inglês).
[2] Cor preta vale 0 e cor branca vale 255.

inicial.

Outros testes foram feitos com imagens homogêneas coloridas. O resultado foi muito semelhante aos apresentados. A faixa de distribuição dos pontos é muito grande, freqüêntemente sobrepondo-se às outras cores e impedindo a diferenciação entre elas, como mostrado nas figuras 7.3 e 7.4.

Nesta figura 7.4 podemos ter a real noção da influência que apenas uma tonalidade tem sobre a câmera do robô. Os pontos distribuiram-se ao longo de toda escala de tons de cinza, que vai de 0 a 255. Talvez com a aplicação de técnicas de processamento de imagem mais elaboradas seja possível extrair alguma informação útil para a diferenciação cromática. Há também a limitação da própria CPU do robô que, por não ser computacionalmente muito potente, impede a aplicação de técnicas mais avançadas.

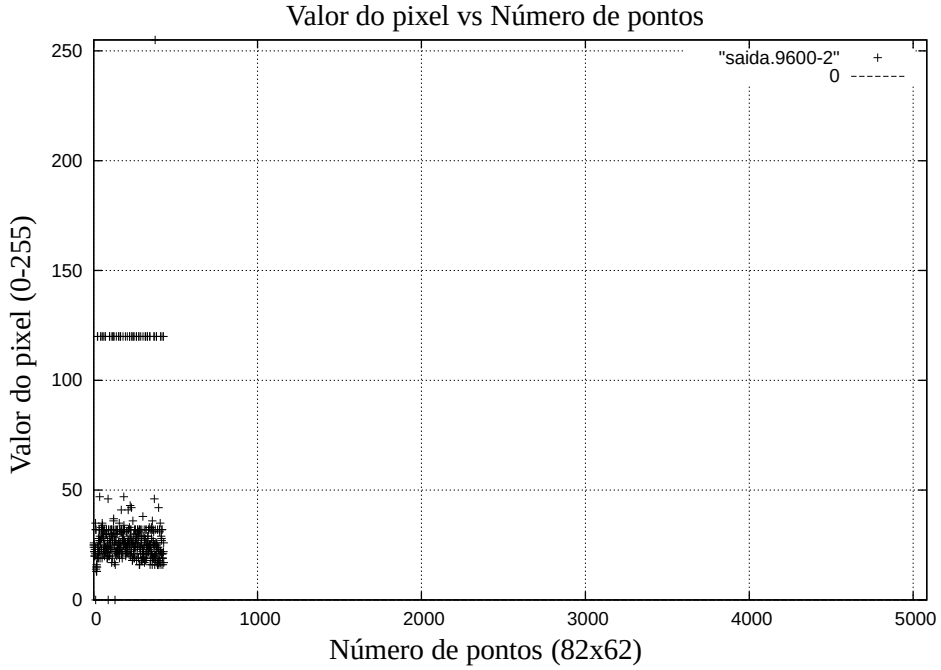

Figura 7.1: Resultado do teste com taxa de transmissão de 9600*bps*, *payload* de 1*byte* e superfície na cor preta

7.2 Testes com o módulo de rádio

O teste com o módulo de rádio foi conduzido com o objetivo de se avaliar quão confiável era a comunicação entre dois pontos, em geral entre um robô (cliente) e o PC (servidor). As figuras 7.1 e 7.2 nos mostram, além das informações já discutidas, a fragilidade do sistema de transmissão de rádio. Ambas transmissões foram configuradas para uma taxa de transferência de 9600*bps*, a única disponível. Em 7.1 preenchemos o campo destinado aos dados do *frame*, chamado de *payload*, com apenas 1*byte*. Este 1*byte* continha justamente a magnitude de um ponto da imagem. Mais uma vez

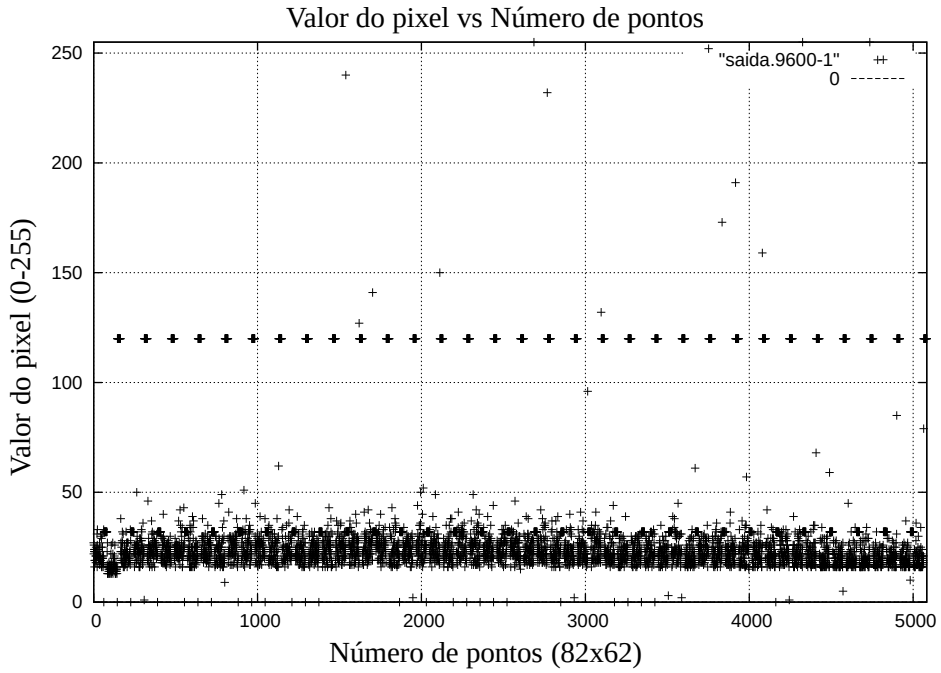

Figura 7.2: Resultado do teste com taxa de transmissão de 9600*bps*, *payload* de 1*byte*, superfície na cor preta e *delay* de 200*ms* entre cada transmissão

nos surpreendemos com a pequena quantidade de *frames* corretamente transmitidos. Com um canal real de comunicação de 9600*bps* estabelecido, apenas 429 dos 5084 pacotes transmitidos chegaram ao destino, menos de 10%!

Um segundo teste foi realizado. Todas as condições apresentadas anteriormente foram mantidas, com excessão de uma. Entre cada transmissão com origem no servidor (desempenhado pelo PC no nosso teste) foi colocado um atraso artificial de 200*ms*. O resultado é mostrado na figura 7.2. Com este atraso proposital todos os pacotes foram transmitidos e recebidos corretamente.

Outros testes foram feitos variando-se outros parâmetros que pudessem ter influência neste resultado. A figura 7.5 mostra o histograma do teste onde são comparadas as taxas de perdas de pacotes. A comunicação foi configurada para 9600*bps* e o campo *payload* preenchido com apenas 1*byte* de dados. A diferença é enorme; na primeira coluna estão plotados a magnitude dos pacotes que foram recebidos utilizando-se o artifício do atraso entre transmissões enquanto que na segunda coluna estão o número de pacotes transmitidos sem este atraso. A conclusão óbvia é que o módulo de rádio não suporta uma comunicação real com taxa de transferência de 9600*bps*. Conseguimos taxas melhores às custas de atrasos que prejudicam todo restante que depende deste sistema. Por último, avaliamos a influência do campo de dados (*payload*) no tempo e confiabilidade da transmissão. Em 7.6 está mostrado que a quantidade de dados que cada *frame* carrega não influi tanto na quantidade de pacotes recebidos. Houve uma queda nesta quantidade mas nada comparado à influência exercida pelo atraso temporal adicionado.

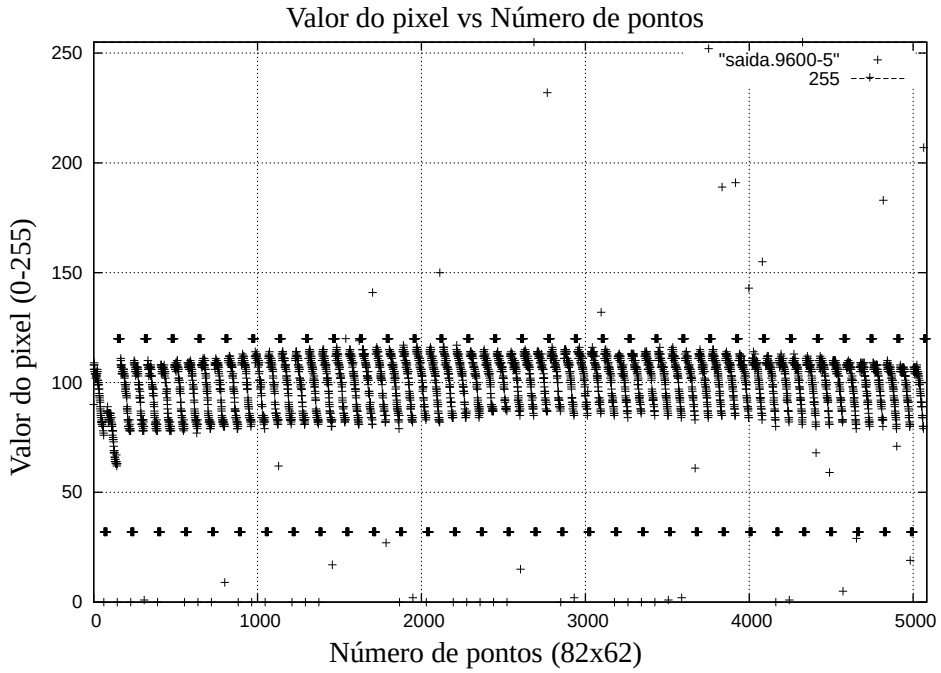

Figura 7.3: Resultado do teste com taxa de transmissão de 9600*bps*, *payload* de 1*byte*, superfície na cor branca e *delay* de 200*ms* entre cada transmissão

7.3 Testes com o dispositivo sensor de distância (PSD)

Os sensores de distância (*Position Sensitive Detector*) representam o principal meio de detecção de obstáculos e, por conseguinte, de estimação de distâncias. Trata-se de sensores óticos que utilizam sinais de baixa freqüência que são refletidos em superfícies de boa refletividade, tipicamente superfícies de cor branca.

Neste modelo de robô, os sensores retornam uma palavra binária de 8*bits*. Através de instruções podemos acessar o conteúdo do registrador onde este valor está armazenado. De posse desta medida, o sistema operacional do robô (RoBIOS) consulta a tabela de descrição de *hardware* (ver apêndice B) para fazer a conversão entre este valor cru (*raw value*) e o valor real da distância em *mm* medida. Esta tabela deve ser personalizada a fim de retornar valores precisos. A figura 7.7 mostra os resultados dos testes feitos com o sensor frontal do robô. Foram comparadas as distância de 10*cm* a 90*cm*, faixa onde a informação retornada pelo sensor é mais confiável segundo o próprio fabricante. Na primeira coluna de cada par de valores está o valor tomado como padrão enquanto que segundo o valor medido pelo sensor.

A interpretação dos histogramas 7.5, 7.6 e 7.7 sugere que estes sejam utilizados para distâncias até em torno de 50*cm*. Acima deste valor a informação retornada não é precisa e, muitas vezes, sem sentido. Mesmo assim, este sub-sistema se mostrou um dos melhores entre todos do robô.

O problema maior verificado foi com relação à disposição dos sensores infra-vermelhos no robô. Poderia-se mudar a localização destes de modo melhorar a navegação e evitar as situações mostradas

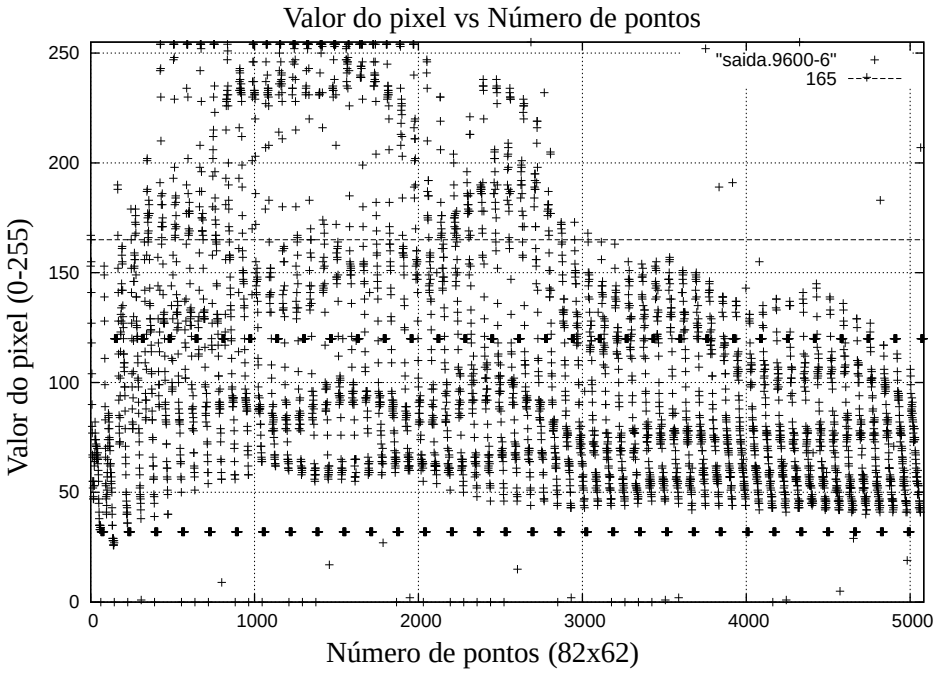

Figura 7.4: Resultado do teste com taxa de transmissão de 9600*bps*, *payload* de 1*byte*, superfície na cor laranja e *delay* de 200*ms* entre cada transmissão

nas figuras 7.10 em que o robô, devido ao seu formato, fica bloqueado num canto do campo, impossibilitado de sair desta posição por si só. Ou então, quando devido ao posicionamento do sensor infra-vermelho frontal, um objeto pode não ser detectado.

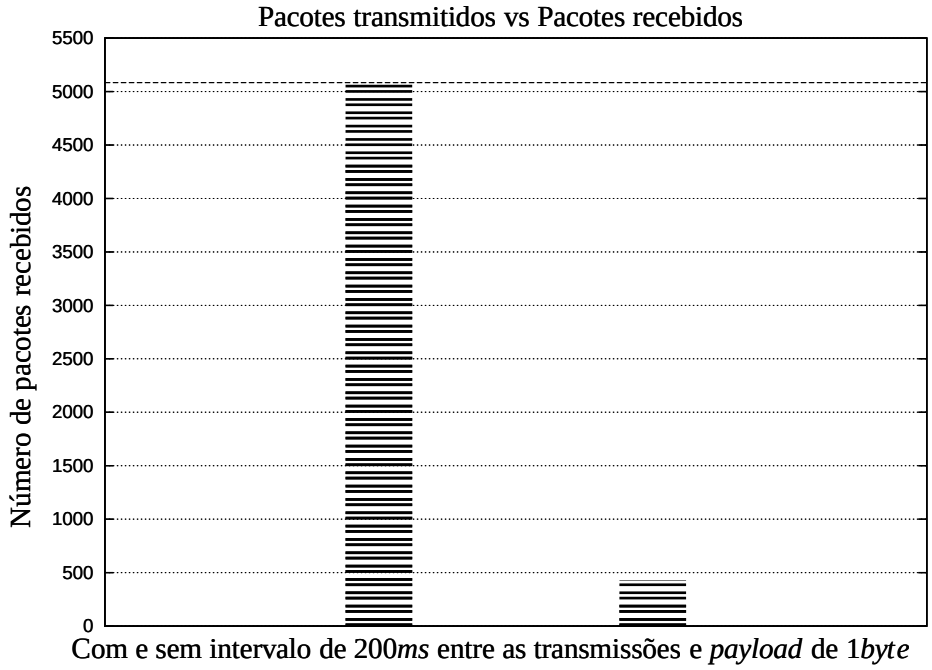

Figura 7.5: Teste de perdas de pacotes do módulo de rádio

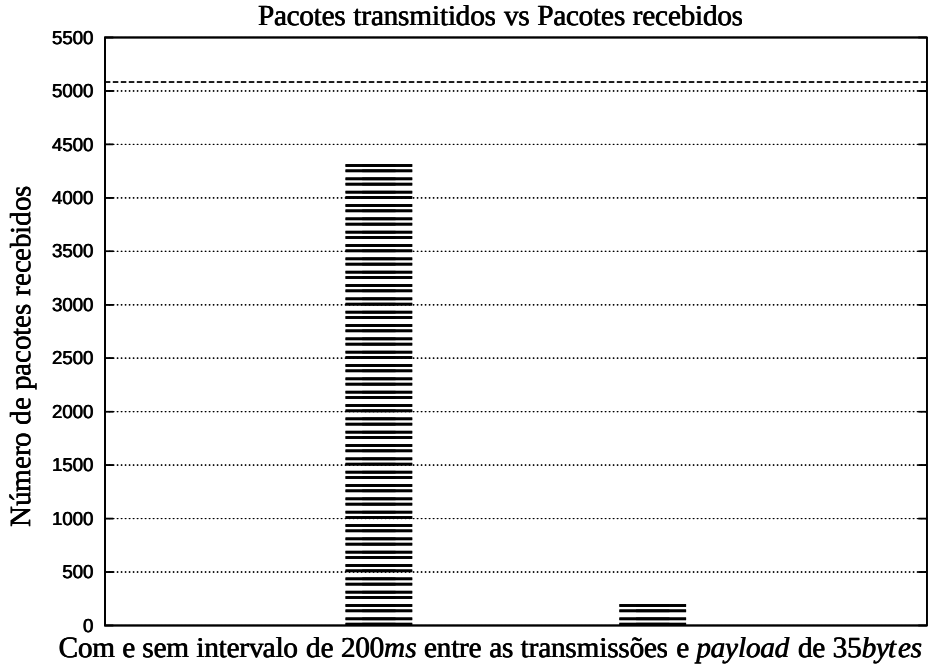

Figura 7.6: Teste de perdas de pacotes do módulo de rádio

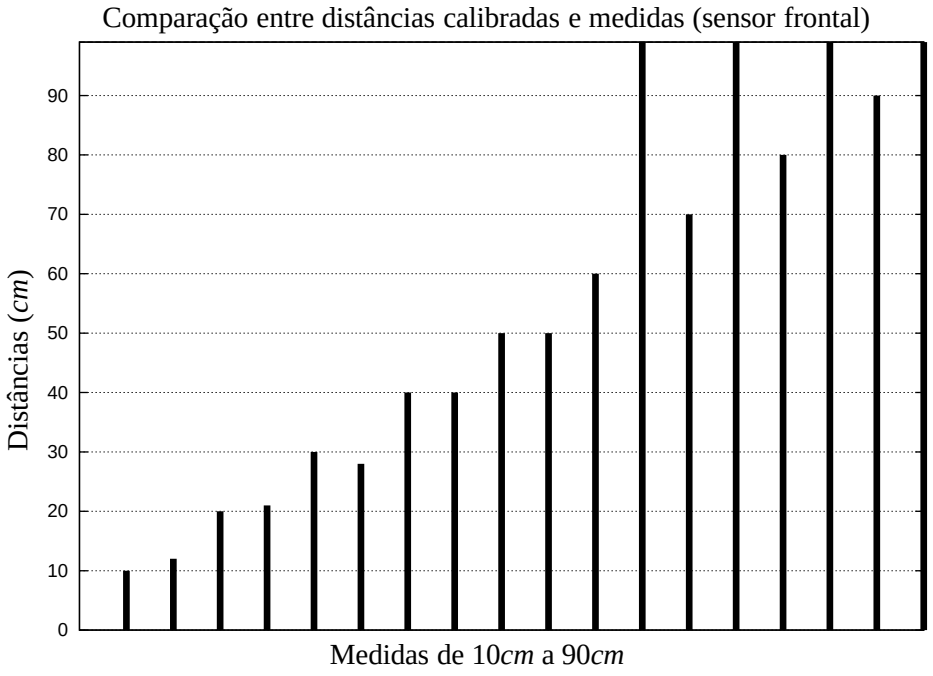

Figura 7.7: Teste de comparação de distâncias do sensor frontal

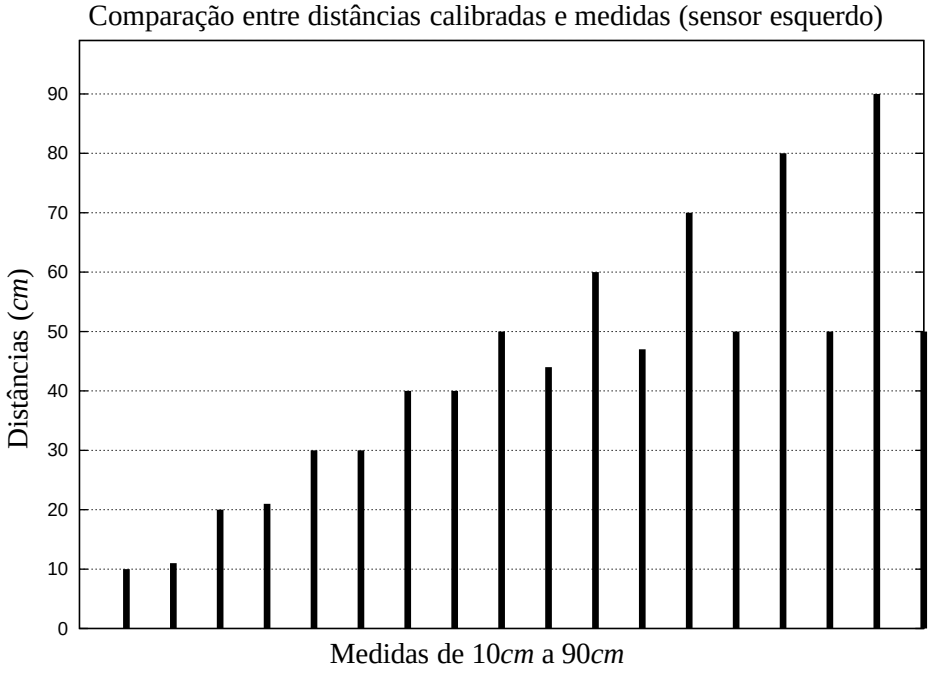

Figura 7.8: Teste de comparação de distâncias do sensor esquerdo

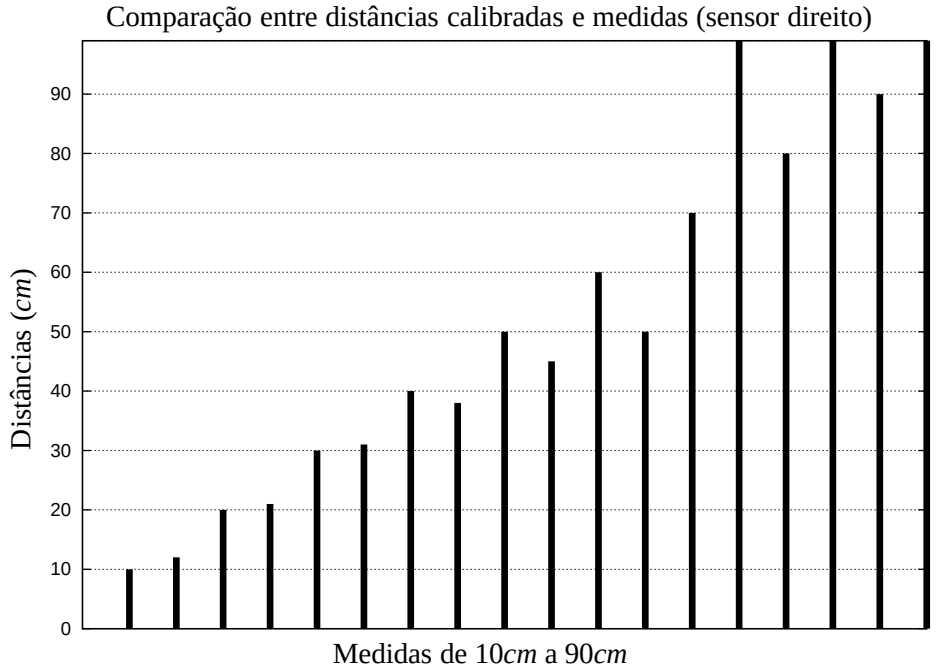

Figura 7.9: Teste de comparação de distâncias do sensor direito

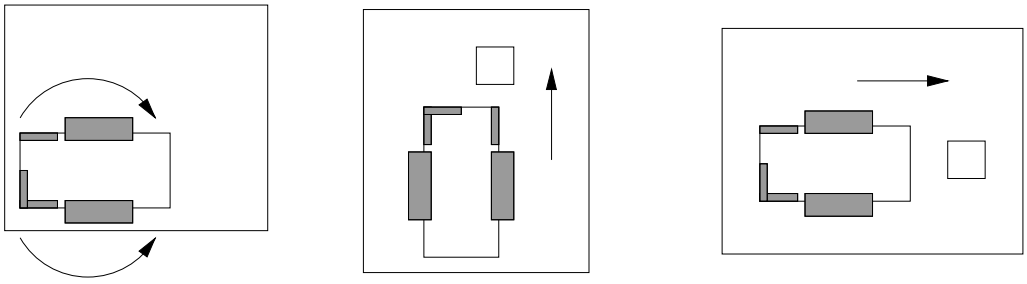

Figura 7.10: Situações indesejáveis freqüentemente encontradas

7.4 Testes com os *encoders*

Os testes com os *encoders* das rodas foram feitos com o objetivo de se verificar características como precisão e repetibilidade dos acionamentos do robô. Isto é fundamental para a navegação que utiliza a técnica *deduced-reckoning*, ou seja, localização e posicionamento com base nas informações cartesianas disponibilizadas pelos sensores óticos acoplados as rodas; tática muito comum em pequenos robôs móveis. Para realizar esta tarefa, utilizamos um pequeno programa (código 3) para obter estas medidas.

Alguns dos resultados obtidos estão mostrados na tabela 7.1.

Distância de $1m$ **e velocidade de** $1m/s$	
Valores ajustados	Valores medidos
$x : 1m$	$x = 0,99m$
$y : 0m$	$y = 0,09m$
$phi : 0rad$	$phi = 0,10rad$
Distância de $2m$ **e velocidade de** $0,5m/s$	
Valores ajustados	Valores medidos
$x : 2m$	$x = 1,98m$
$y : 0m$	$y = 0,16m$
$phi : 0rad$	$phi = 0,10rad$
Distância de $2m$ **e velocidade de** $1m/s$	
Valores ajustados	Valores medidos
$x : 2m$	$x = 1,97m$
$y : 0m$	$y = -0,15m$
$phi : 0rad$	$phi = 0,14rad$

Tabela 7.1: Resultados dos testes com os *encoders*

Pode-se perceber que, mesmo ajustando-se o robô numa posição com coordenadas *y* e *phi* (ângulo) nulos, no final da translação há um erro associado, pois estes valores são diferentes dos observados inicialmente. Apesar de pequena a distância percorrida, o erro é considerável. Pior, isto tem efeito cumulativo. Em testes mais longos realizados, ao final, os dados obtidos não tinham praticamente valor algum, tal a disparidade encontrada. Isto inviabiliza totalmente a navegação por *"dead-reckoning"*. Não há como planejar algo sobre bases tão frágeis.

Notou-se que com o aumento da velocidade de translação do robô, este tende a derrapar e girar sem controle. Mesmo à $1m/s$ isto foi observado. Os testes foram realizados em dois tipos de superfícies diferentes, uma mesa de madeira pintada com tinta sintética especialmente preparada para este fim e seguindo exatamente as especificações da *RoboCup* e outra superfície recoberta com piso cerâmico. Em ambas o resultado foi semelhante, giros sobre o próprio eixo e perda do controle. Duas hipóteses podem ser levantadas para explicar este fato: estas duas superfícies não são adequadas pois as rodas do robô não conseguem obter uma boa aderência ao piso ou então a diferença de velocidade entre as rodas faz com que surja uma componente resultante diferente de zero com o gradativo

aumento de energia aplicada a elas.

A interpretação dos dados mostrados acima prova que a utilização dos *encoders* das rodas para navegação, especialmente sob o método *deduced-reckoning*, deve ser evitada quando a velocidade for superior à $1m/s$, valor este que equivale à 50% da velocidade máxima possível para este modelo testado.

Código 3 Obtém informações dos *encoders* das rodas

```
#include <stdio.h>
#include "eyebot.h"

int main (void)
{
        int erro = 0;
        VWHandle v_omega;
        PositionType posicao;

        LCDMode(NONSCROLLING|NOCURSOR);
        LCDPrintf("   UFSC-Team!   \n");
        LCDMenu("NOP","NOP","NOP","FIM");

        v_omega = VWInit(VW,1);
        if (erro != 0) {
                printf("Erro: VWInit !\n");
                return 1;
        }

        AUBeep ();
        if (VWStartControl(v_omega,7.0,0.3,7.0,0.1) != 0) {
                printf("Um erro ocorreu durante a inicializacao!\n");
                return 1;
        }

        VWSetPosition(v_omega,0,0,0);
        VWDriveStraight(v_omega,1,1);
        VWDriveWait(v_omega);
        VWGetPosition(v_omega,&posicao);
        LCDPrintf("x:%f\n",posicao.x);
        LCDPrintf("y:%f\n",posicao.y);
        LCDPrintf("phi:%f\n",posicao.phi);
        VWStopControl(v_omega);
        VWRelease(v_omega);

        return 0;

}
```

Capítulo 8

Conclusões e Futuros Trabalhos

Visando criar condições para a realização do objetivo inicial de implementar a arquitetura para agentes cognitivos descrita no capítulo 5, estudou-se como obter, a partir das disponibilidades técnicas da plataforma, dados e meios de controle que permitam esta implementação.

O nível reativo implementado no processo Interface é responsável pela resposta em tempo real do agente. É através dele que informações como a percepção visual do agente são captadas e também comandos são enviados ao simulador para controlar o jogador. É composto por um conjunto de controladores nebulosos que, dependendo da função desempenhada por este (defesa, goleiro ou ataque), modela seu comportamento. As ações imediatas a serem tomadas são atribuições do único controlador ativo a cada momento.

O nível instintivo é implementado no processo Coordinator e responsável por identificar o estado em que a partida se encontra e atualizar a informação simbólica usada pelo nível cognitivo. Na prática consiste de um sistema especialista de ciclo único. Por fim, o nível decisório cognitivo é o responsável pelo planejamento em alto nível do agente, ou seja, por determinar as metas locais e globais do agente. Não tem efeito direto sobre o reativo, mas sim sobre o instintivo. É implementado através de um sistema especialista simbólico de maior complexidade que o encontrado no instintivo. Utilizado basicamente para planejar metas futuras e especificar requisições de cooperação.

Os resultados dos testes realizados na seção anterior mostram que, alguns dos sub-sistemas do robô não são adequados ao jogo de futebol, tornando todo este modelo inapto. Isto porque este problema exige, principalmente, velocidade e precisão das informações captadas. A câmera embarcada *EyeCam* mostrou-se pouco confiável na aquisição das imagens. Mesmo naquelas obtidas com o programa *Xcameraviewer*, houve sempre um desvio para a tonalidade verde das imagens capturadas. Uma hipótese para este fato seria a influência do ambiente, como iluminação fornecida por lâmpadas fluorescentes. No entanto, mesmo com iluminação incândecente o resultado observado foi o mesmo, levando a crer que o problema está relacionado com a própria câmera ou então que as condições para o correto funcionamento são muito exigentes, também dificultando a utilização deste tipo de robô já que, o problema do futebol de robô é acima de tudo desafiante, impreciso e imprevisível. Nos

testes de identificação utilizando como base os valores dos pontos das imagens as diferenças foram ainda maiores. Em nenhum deles foi possível utilizar esta informação de forma útil. O problema da fiel reprodução de cores por parte da câmera torna impossível a utilização destes dados sem um processamento adequado. Durante a realização deste trabalho não foram utilizadas qualquer espécie de algoritmos de tratamento de imagens, com excessão daquelas disponibilizadas pela própria API de programação da RoBIOS. Além disso, a propaganda feita por parte dos desenvolvedores de que seria possível realizar processamento em tempo real das imagens captadas não provou ser verdadeira por dois motivos: primeiro devido ao baixo desempenho do microcontrolador que não consegue processar a tempo todas as informações e segundo porque a própria câmera não é capaz de capturar imagens numa taxa maior do que a apurada, em torno de 4 *frames/s* nesta versão. A chamada quarta geração (nome código *MK*4) já possui mudanças no *hardware* que corrigem esta falha, segundo a equipe que desenvolve esta plataforma de robôs. Um provável trabalho futuro seria estudar mais a fundo a teoria de processamento e tratamento de imagens com o objetivo de melhorar a resposta da câmera, dentro do possível.

Outro ponto crítico trabalhado foi o módulo de rádio. Testes mostraram que não é possível utilizá-lo na taxa de transferência nominal de 9600*bps*. Em todos eles houve perdas significativas de pacotes, como comprovam as figuras 7.5 e 7.6. Duas hipóteses podem explicar este fato: a saturação do canal quando muitas mensagens são transmitidas num curto espaço de tempo ou então o mau dimensionamento dos *buffers* de recepção. Em ambos, porém, nota-se um equívoco com relação ao projeto. De modo algum isto poderia acontecer numa taxa baixa com esta. Um atraso temporal foi adicionado entre cada duas transmissões a fim de estabilizar e garantir a correta recepção dos pacotes. Isto, é claro, é uma medida paleativa. O protocolo de comunicação é não orientado a conexão e por este motivo não há garantias de que mensagens enviadas cheguem corretamente ao destino. Este recurso, caso necessário, deve ser disponibilizado pela aplicação.

Entre os pontos positivos desta plataforma estão a facilidade de programação e depuração. Pode-se programá-los utilizando linguagem de alto nível com a "C" ou então linguagem de montagem *assembly*. Todas as ferramentas de desenvolvimento são livres e distribuídas gratuitamente via Internet. Estes modelos são ideais para o ensino e pesquisa de conceitos ligados tanto à informática quanto à automação, ou seja, abrangente o suficiente para a condução de estudos em vários níveis.

Devido ao exposto acima, fica a sugestão de se criar um grupo multidisciplinar interessado em desenvolver pesquisas nesta área para se construir protótipos nossos, adaptados às nossas necessidades e abertos para a comunidade científica desenvolver trabalhos sobre ele. Nem sempre o melhor é a simples aquisição de produtos acabados. Isto mostrou-se atraente naquele momento em que não era possível o desenvolvimento por várias razões mas, futuramente, máquinas com tecnologia nacional devam se tornar o caminho natural.

Por fim, apesar dos objetivos iniciais não puderem ser alcançados totalmente, contribuímos em parte para o início de uma nova área de pesquisa, a robótica móvel com robôs de verdade que, diferentemente do ambiente simulado, é repleto de desafios. Seriam necessárias algumas mudanças

no *hardware* do robô para torná-lo mais confiável e preciso em suas medições. Desta forma a integração à arquitetura desenvolvida para o time de agentes de *software* (time simulado) poderia ser aproveitada. O time simulado baseia-se fortemente na informação visual retornada do simulador soccerserver. Como a câmera do robô provou não ser adequada, a migração do processo Interface para o robô não pode ser feita. Os outros dois níveis decisórios, Instintivo e Cognitivo, sempre necessitarão de uma máquina dedicada tal a exigência computacional requerida. Os testes anteriores deixaram claro a dificuldade da integração da arquitetura desenvolvida para a categoria de agentes simulados e esta de robôs reais, principalmente quando o sentido da visão era necessária. No entanto, quando esta não era fundamental, como no caso da detecção de obstáculos, controladores nebulosos utilizando os sensores de proximidade se mostraram bastante eficientes, permitindo uma navegação em ambientes desconhecidos satisfatória.

Apêndice A

Programa Xcameraviewer

```
/* Xcameraviewer
 * EyeCam's frame grabber - UNIX version
 *
 * Department of Automation and Systems
 * Control and Microinformatic Laboratory
 * Federal University of Santa Catarina
 * Florianopolis, SC - Brazil
 *
 * Copyright 2002,2003 Luciano Rottava da Silva (rottava@das.ufsc.br)
 *
 * This file is part of Xcameraviewer.
 *
 * Xcameraviewer is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * Xcameraviewer is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
 * 02111-1307  USA
 *
 */
```

```
/* Header files */
#include <stdio.h>
#include "eyebot.h"

/* Symbolic constants */
#define TRUE 1
#define nrows 62
#define ncolumns 82
#define nrgb 3
#define size 31

int main (void)
{
        int camera,row,col,rgb;
        BYTE r1[size],g1[size],b1[size];
        BYTE r2[size],g2[size],b2[size];
        BYTE pixel;
/* host (PC) ID */
        BYTE id_target = 0;
        image mono;
        colimage color;
        RadioIOParameters radioParams;

/* Setting radio parameters */
        RADIOGetIoctl(&radioParams);
        radioParams.interface = SERIAL2;
        radioParams.speed = SER9600;
        radioParams.id = 1;
        RADIOSetIoctl(radioParams);

/* Camera settings */
        camera = CAMInit(NORMAL);
        CAMMode(AUTOBRIGHTNESS);
        CAMSet(FPS15,0,0);

        LCDPrintf("EyeCam model: %d\n",camera);
        AUBeep();

/* Starting radio link */
        if (RADIOInit() != 0) {
```

```
                printf("Radio init error!\n");
                return 1;
        }
        else printf("Radio link OK!\n");

/* LCD settings */
        OSSleep(100);
        LCDClear();
        OSSleep(100);
        LCDMode(SCROLLING|NOCURSOR);
        LCDClear();
        LCDMenu(" ", " ", " ", "End");

/* Get one color frame ... */
        CAMGetColFrame(&color,0);
/* ... convert it to mono and put in display */
        IPColor2Grey(&color,&mono);
        LCDPutGraphic(&mono);

/* Now send this color frame to PC */
        LCDClear();
        LCDPrintf("-------------\n");
        LCDPrintf("Xcameraviewer\n");
        LCDPrintf("by L. Rottava\n");
        LCDPrintf("-------------\n\n");
        LCDPrintf("Sending...");

        OSSleep(50);
        for (col = 0 ; col < ncolumns ; col++) {
                for (row = 0 ; row < 31 ; row++)
                {
                        r1[row] = color[row][col][0];
                        g1[row] = color[row][col][1];
                        b1[row] = color[row][col][2];
                        r2[row] = color[row+31][col][0];
                        g2[row] = color[row+31][col][1];
                        b2[row] = color[row+31][col][2];
                }
                RADIOSend(id_target,size,&r1[size]);
                OSSleep(15);
                RADIOSend(id_target,size,&g1[size]);
```

```
                OSSleep(15);
                RADIOSend(id_target,size,&b1[size]);
                OSSleep(15);
                RADIOSend(id_target,size,&r2[size]);
                OSSleep(15);
                RADIOSend(id_target,size,&g2[size]);
                OSSleep(15);
                RADIOSend(id_target,size,&b2[size]);
                OSSleep(15);
        }

        LCDPrintf("done.\n");

/* Release all resources allocated by radio and camera */
        RADIOTerm();
        CAMRelease();

        return 0;
}
```

Código 4: Versão *wireless* do programa Xcameraviewer (cliente embarcado no robô).

```
/* Xcameraviewer
 * EyeCam's frame grabber - UNIX version
 *
 * Department of Automation and Systems
 * Control and Microinformatic Laboratory
 * Federal University of Santa Catarina
 * Florianopolis, SC - Brazil
 *
 * Copyright 2002,2003 Luciano Rottava da Silva (rottava@das.ufsc.br)
 *
 * This file is part of Xcameraviewer.
 *
 * Xcameraviewer is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * Xcameraviewer is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 */
```

```
/* Header files */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/keysym.h>
#include "eyebot.h"
#include "remote.h"

/* Symbolic constants */
#define TRUE 1
#define nrows 62
#define ncolumns 82
#define nrgb 3

void message (void)
{
        printf("\n-------------------------------------------------------------------------------\n");
        printf("This file is part of Xcameraviewer.\n\n");
        printf("Xcameraviewer is free software; you can redistribute it and/or modify\n");
        printf("it under the terms of the GNU General Public License as published by\n");
        printf("the Free Software Foundation; either version 2 of the License, or\n");
        printf("at your option) any later version.\n\n");
        printf("Xcameraviewer is distributed in the hope that it will be useful,\n");
        printf("but WITHOUT ANY WARRANTY; without even the implied warranty of\n");
        printf("MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n");
        printf("GNU General Public License for more details.\n\n");
        printf("You should have received a copy of the GNU General Public License\n");
        printf("along with this program; if not, write to the Free Software\n");
```

```
        printf("Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA\n");
        printf("\n------------------------------------------------------------------------\n");
}

int main (int argc, char *argv[])
{
        BYTE r1[31],g1[31],b1[31];
        BYTE r2[31],g2[31],b2[31];
/* EyeBot ID */
        BYTE id_source = 1;
        int size,row,col;
        RadioIOParameters radioParams;

/* Display variables */
        Display *display;
        char *display_name = NULL;

/* Window variables */
        Window parent,window;
        int x,y,screen_num;
        unsigned int width,height,border_width;
        unsigned long border,background;
        char *window_name = "Eye";

/* Graphics context variables */
        GC gc;
        XGCValues values;

/* Color support */
```

```
        Colormap colormap;
        XColor color;

/* Event variables */
        XEvent event;
        /*KeySym keysym_return;
        char buffer_return[10];*/

/* Print GPL message */
        message();

/* Set radio parameters */
        RADIOGetIoctl(&radioParams);
        radioParams.interface = SERIAL2;
        radioParams.speed = SER9600;
        radioParams.id = 0;
        RADIOSetIoctl(radioParams);

        if (RADIOInit() != 0)
        {
                printf("Radio init error!\n");
                return(1);
        }
        else
        {
                printf(" ** Messages **\n");
                printf("Radio init ok!\n");
        }
```

```
/* Open connection to X server */
        display = XOpenDisplay(display_name);
        if (display == NULL)
        {
                fprintf(stderr,"%s: couldn't connect to X server %s\n",argv[0],display_name);
                exit(1);
        }

        screen_num = XDefaultScreen(display);
        parent = DefaultRootWindow(display);
        x = y = 0;
        width = 82;
        height = 62;
        border_width = 0;
        border = BlackPixel(display, screen_num);
        background = WhitePixel(display, screen_num);

/* Create a simple window */
        window = XCreateSimpleWindow(display,parent,x,y,width,height,border_width,border,background);

/* Set window name */
        XStoreName(display,window,window_name);

/* Events handling */
        XSelectInput(display,window,StructureNotifyMask|ExposureMask);

        /*XSelectInput(display,window,NoEventMask);*/

/* Create and set colormap to the application */
```

```
        colormap = XCreateColormap(display,window,DefaultVisual(display,screen_num),AllocNone);
        XSetWindowColormap(display,window,colormap);

/* Create graphics context */
        gc = XCreateGC(display,window,0,&values);
        /*XSetForeground(display,gc,color.pixel);*/

/* Map the window */
        XMapWindow(display,window);

        for (col = 0; col < ncolumns; col++)
        {
                RADIORecv(&id_source,&size,&r1[size]);
                RADIORecv(&id_source,&size,&g1[size]);
                RADIORecv(&id_source,&size,&b1[size]);
                ;
                RADIORecv(&id_source,&size,&r2[size]);
                RADIORecv(&id_source,&size,&g2[size]);
                RADIORecv(&id_source,&size,&b2[size]);
                ;
                for (row = 0; row < 31; row++)
                {
                        color.red = r1[row] << 8;
                        color.green = g1[row] << 8;
                        color.blue = b1[row] << 8;
                        XAllocColor(display,colormap,&color);
                        XSetForeground(display,gc,color.pixel);
                        XDrawPoint(display,window,gc,col,row);
                        XFlush(display);
```

```
                }
                for (row = 31; row < 62; row++)
                {
                        color.red = r2[row-31] << 8;
                        color.green = g2[row-31] << 8;
                        color.blue = b2[row-31] << 8;
                        XAllocColor(display,colormap,&color);
                        XSetForeground(display,gc,color.pixel);
                        XDrawPoint(display,window,gc,col,row);
                        XFlush(display);
                }
        }

        printf("done\n");

        while (TRUE) {
                XMapWindow(display,window);
                XNextEvent(display,&event);
                switch(event.type)
                {
                        case Expose:
                                if(event.xexpose.count != 0) {
                                        for (col = 0; col < ncolumns; col++)
                                        {
                                                RADIORecv(&id_source,&size,&r1[size]);
                                                RADIORecv(&id_source,&size,&g1[size]);
                                                RADIORecv(&id_source,&size,&b1[size]);
                                                ;
                                                RADIORecv(&id_source,&size,&r2[size]);
```

```
                                        RADIORecv(&id_source,&size,&g2[size]);
                                        RADIORecv(&id_source,&size,&b2[size]);
                                        ;
                                        for (row = 0; row < 31; row++)
                                        {
                                                color.red = r1[row] << 8;
                                                color.green = g1[row] << 8;
                                                color.blue = b1[row] << 8;
                                                XAllocColor(display,colormap,&color);
                                                XSetForeground(display,gc,color.pixel);
                                                XDrawPoint(display,window,gc,col,row);
                                                XFlush(display);
                                        }
                                        for (row = 31; row < 62; row++)
                                        {
                                                color.red = r2[row-31] << 8;
                                                color.green = g2[row-31] << 8;
                                                color.blue = b2[row-31] << 8;
                                                XAllocColor(display,colormap,&color);
                                                XSetForeground(display,gc,color.pixel);
                                                XDrawPoint(display,window,gc,col,row);
                                                XFlush(display);
                                        }
                                }
                        }
                }
        }

/* Destroy GC */
```

```
        XFreeGC(display,gc);

/* Close connection to the X server */
        XCloseDisplay(display);

        return(0);
}
```

Código 5: Versão *wireless* do programa Xcameraviewer (servidor executado num PC).

Apêndice B

Tabela de Descrição de *Hardware*

A *Hardware Description Table (HDT)* é uma estrutura de dados que contém informações sobre cada um dos componentes do robô. Esta estrutura é utilizada pelo sistema operacional do robô (RoBIOS) para acessar, por exemplo, os motores e os sensores de proximidade. Um arquivo deve ser criado para conter todos os periféricos do robô. A seguir, este arquivo deve ser transferido para o robô via porta serial e armazenado numa região de memória especialmente reservada para isto.

A RoBIOS não terá como comandar um determinado componente caso sua entrada (rotinas de acesso na realidade) não esteja presente na HDT. Por este motivo a equipe que desenvolve estes robôs não abriu o código fonte do sistema operacional RoBIOS. Realmente não é fundamental ter acesso às rotinas de mais baixo nível da RoBIOS, apesar de interessante do ponto de vista das pesquisas.

Apenas para ilustrar melhor o que foi dito, o código a seguir apresenta a estrutura de dados do servomotor, e o código 8 apresenta uma HDT completa.

Código 6 Código da estrutura do servo

```
typedef struct
{
        int         driver_version;
        int         tpu_channel;
        int         tpu_timer;
        int         pwm_period;
        int         pwm_start;
        int         pwm_stop;
} servo_type
```

Esta estrutura é preenchida com os dados presentes na HDT, por exemplo:

Código 7 Rotina de acesso de uma estrutura servomotor

```
servo_type servo0 = {1, 0, TIMER2, 20000, 700, 1700};
```

```
/*-------------------------------------------------------------------------
Estrutura de dados HDT (Hardware Description Table) para EyeBot MK3
Adaptado para os robos do projeto UFSC-Team por Luciano Rottava da Silva
Versao para ser utilizada com RoBIOS 5.0
-------------------------------------------------------------------------*/

#define VERSION 1.0
#define NAME "Wiki"
#define ID 1

#include "hdt.h"
#include "hdt_sem.h"
#include "types.h"
#include "const.h"
#include "rs232.h"
#include "cam.h"
#include "librobi/librobi.h"

/* Informacoes importantes para a RoBIOS */
int      magic = 123456789;
extern  HDT_entry_type HDT[];
HDT_entry_type  *hdtbase = &HDT[0];

/*-------------------------------------------------------*/
/*------ TABELAS DE CONVERSAO (PSDs E MOTORES) ------ */
/*-------------------------------------------------------*/

/* Sensor de presenca (PSD) IR - lado esquerdo - intervalo 100 - 500 mm */
```

```
short distleft[128] =
        { 100,100,100,100,100,100,100,100,100,100,/* 1 - 20 */
          100,100,100,100,100,100,100,100,100,110,/* 21 - 40 */
          110,110,110,120,120,120,130,130,130,135,/* 41 - 60 */
          140,140,145,145,150,150,150,150,150,150,/* 61 - 80 */
          150,150,150,150,150,170,190,200,200,200,/* 81 - 100 */
          200,200,200,200,210,210,210,210,220,220,/* 101 - 120 */
          230,230,240,240,250,250,250,300,300,300,/* 121 - 140 */
          300,370,380,390,400,400,400,410,420,430,/* 141 - 160 */
          440,450,470,490,500,500,500,500,999,999,/* 161 - 180 */
          999,999,999,999,999,999,999,999,999,999,/* 181 - 200 */
          999,999,999,999,999,999,999,999,999,999,/* 201 - 220 */
          999,999,999,999,999,999,999,999,999,999,/* 221 - 240 */
          999,999,999,999,999,999,999,999};       /* 241 - 256 */

/* Sensor de presenca (PSD) IR - frente - intervalo 100 - 500 mm */
short distfront[128] =
        { 100,100,100,100,100,100,100,100,100,110,/* 1 - 20 */
          110,110,110,120,120,120,120,130,130,130,/* 21 - 40 */
          140,140,140,140,150,150,150,150,150,150,/* 41 - 60 */
          150,155,160,160,160,165,170,170,175,180,/* 61 - 80 */
          185,185,190,190,200,210,210,220,225,230,/* 81 - 100 */
          235,240,250,260,270,280,300,310,350,370,/* 101 - 120 */
          390,400,410,420,470,500,500,999,999,999,/* 121 - 140 */
          999,999,999,999,999,999,999,999,999,999,/* 141 - 160 */
          999,999,999,999,999,999,999,999,999,999,/* 161 - 180 */
          999,999,999,999,999,999,999,999,999,999,/* 181 - 200 */
          999,999,999,999,999,999,999,999,999,999,/* 201 - 220 */
          999,999,999,999,999,999,999,999,999,999,/* 221 - 240 */
```

```
          999,999,999,999,999,999,999,999};       /* 241 - 256 */

/* Sensor de presenca (PSD) IR - lado direito - intervalo 100 - 500 mm */
short distright[128] =
        { 100,100,100,100,100,100,100,100,100,100,/* 1 - 20 */
          100,100,100,110,110,120,120,130,130,130,/* 21 - 40 */
          140,140,140,140,150,150,150,150,150,150,/* 41 - 60 */
          150,155,160,160,160,165,170,170,175,160,/* 61 - 80 */
          170,170,170,180,180,180,190,190,200,200,/* 81 - 100 */
          200,200,210,210,220,230,230,240,250,250,/* 101 - 120 */
          260,260,270,280,300,300,310,320,330,350,/* 121 - 140 */
          350,360,360,380,390,400,400,450,470,500,/* 141 - 160 */
          500,999,999,999,999,999,999,999,999,999,/* 161 - 180 */
          999,999,999,999,999,999,999,999,999,999,/* 181 - 200 */
          999,999,999,999,999,999,999,999,999,999,/* 201 - 220 */
          999,999,999,999,999,999,999,999,999,999,/* 221 - 240 */
          999,999,999,999,999,999,999,999};       /* 241 - 256 */

BYTE motconv[101]= {0,1,2,3,4,5,6,7,8,9,10,
                    11,12,13,14,15,16,17,18,19,20,
                    21,22,23,24,25,26,27,28,29,30,
                    31,32,33,34,35,36,37,38,39,40,
                    41,42,43,44,45,46,47,48,49,50,
                    51,52,53,54,55,56,57,58,59,60,
                    61,62,63,64,65,66,67,68,69,70,
                    71,72,73,74,75,76,77,78,79,80,
                    81,82,83,84,85,86,87,88,89,90,
                    91,92,93,94,95,96,97,98,99,100};
```

```
/* Motores DC */
motor_type motor0 = {2,1,TIMER1,8196,(void*)OutBase,3,2,(BYTE*)&motconv};
motor_type motor1 = {2,0,TIMER1,8196,(void*)OutBase,0,1,(BYTE*)&motconv};
quad_type decoder0 = {1,2,3,MOTOR_LEFT,3235,2.0};
quad_type decoder1 = {1,5,4,MOTOR_RIGHT,3235,2.0};
vw_type drive = {1,DIFFERENTIAL_DRIVE,{QUAD_LEFT,QUAD_RIGHT,0.084}};

/* SERVOs */
servo_type servo0 = {2, 12, TIMER2, 20000, 700, 1700}; /* camera */
servo_type servo1 = {2, 13, TIMER2, 20000, 800, 1700}; /* kicker */

/* PSDs */
psd_type psdleft = {0,14,(BYTE*)InBase,0,AH,(BYTE*)OutBase,4,AH,(short*)&distleft};
psd_type psdfront = {0,14,(BYTE*)InBase,1,AH,(BYTE*)OutBase,4,AH,(short*)&distfront};
psd_type psdright = {0,14,(BYTE*)InBase,2,AH,(BYTE*)OutBase,4,AH,(short*)&distright};

/* Compasso Digital */
compass_type compass = {1,11,(void*)OutBase,5,(void*)OutBase,6,(void*)InBase,4};

/* Informacoes do controlador EyeBot */
info_type roboinfo = {1,PLATFORM,SER57600,RTSCTS,SERIAL1,NO_AUTOLOAD,0,AUTOBRIGHTNESS,
BATTERY_ON,33,VERSION,NAME,ID};

/* Waitstate para CPU de frequencia entre 16MHz - 20Mhz */
waitstate_type waitstates = {0,3,0,1,0,2};


/* ------ Lista HDT que contem todos componentes de hardware ------- */
HDT_entry_type HDT[] =
```

```
{
    {MOTOR,MOTOR_LEFT,"Esq.",(void *)&motor1},
    {MOTOR,MOTOR_RIGHT,"Dir.",(void *)&motor0},
    {QUAD,QUAD_LEFT,"Esq.",(void *)&decoder0},
    {QUAD,QUAD_RIGHT,"Dir.",(void *)&decoder1},
    {VW,VW,"Drive",(void *)&drive},
    {SERVO,SERVO0,"Camera",(void *)&servo0},
    {SERVO,SERVO1,"Chute",(void *)&servo1},
    {PSD,PSD_LEFT,"Esq.", (void *)&psdleft},
    {PSD,PSD_FRONT,"Frente", (void *)&psdfront},
    {PSD,PSD_RIGHT,"Dir.", (void *)&psdright},
    {COMPASS,COMPASS,"Compas",(void *)&compass},
    {WAIT,WAIT,"Espere",(void *)&waitstates},
    {INFO,INFO,"Info",(void *)&roboinfo},
    {END_OF_HDT,UNKNOWN_SEMANTICS,"End",(void *)0}
};
```

Código 8: Tabela de descrição de *hardware*).

Apêndice C

O Simulador SoccerServer

O ambiente de simulação utilizado nas pesquisas com agentes de *software* chama-se *SoccerServer*. Este sistema, composto por dois programas distintos[1], simulam os movimentos dos objetos jogadores e bola. Cada jogador é representado e controlado por um único programa cliente. O conjunto destes clientes forma as equipes. São, portanto, onze programas (clientes) por time, totalizando vinte e dois; exatamente como no futebol jogado por humanos. A figura C.1 ilustra a arquitetura interna do simulador.

A conexão entre os clientes e o servidor é feita através de *sockets* UDP/IP[2]. É através deste canal de comunicacão que as mensagens e os comandos são enviados e recebidos. Nenhuma outra forma de comunicação além desta é permitida. Não é possível, por exemplo, a troca direta de informações entre clientes. Isto é, na realidade, mais um "acordo de cavalheiros" do que propriamente uma limitação do sistema.

O acompanhamento da partida pode ser feito através do programa soccermonitor. Este programa exibe uma janela gráfica[3] onde é apresentado um campo de futebol virtual (ver figura C.2. Pode-se, inclusive, acompanhar o andamento das partidas via Internet, graças à arquitetura cliente/servidor do simulador. O programa foi construído em escala e, por este motivo, há muitas semelhanças com um campo verdadeiro. Todas as medidas e linhas de marcação foram preservadas. A maior parte das regras do jogo foram mantidas. Houve, evidentemente, adaptações para esta nova realidade. Talvez a modificação mais marcante ficou por conta da duplicação do tamanho das traves. O fato deste ser um problema bidimensional ao invés de tridimensional como no mundo real teve influência. A experiência prática expôs esta limitação e comprovou a dificuldade em se marcar gols mantendo-se as medidas originais.

---

[1] O servidor de conexões propriamente dito chamado *soccerserver* e uma representação gráfica do campo de futebol em 2D chamado *soccermonitor*.

[2] A escolha de um protocolo não orientado à conexão deve-se ao dinamismo deste tipo de jogo. Não há tempo para retransmissões em caso de falhas. Além disso, a possibilidade de mensagens serem perdidas também pode ser vista como uma *feature* que modela a incerteza e a imprecisão associadas ao domínio.

[3] Requer o sistema de janelas X Window System, padrão gráfico da maioria dos sistemas UNIX.

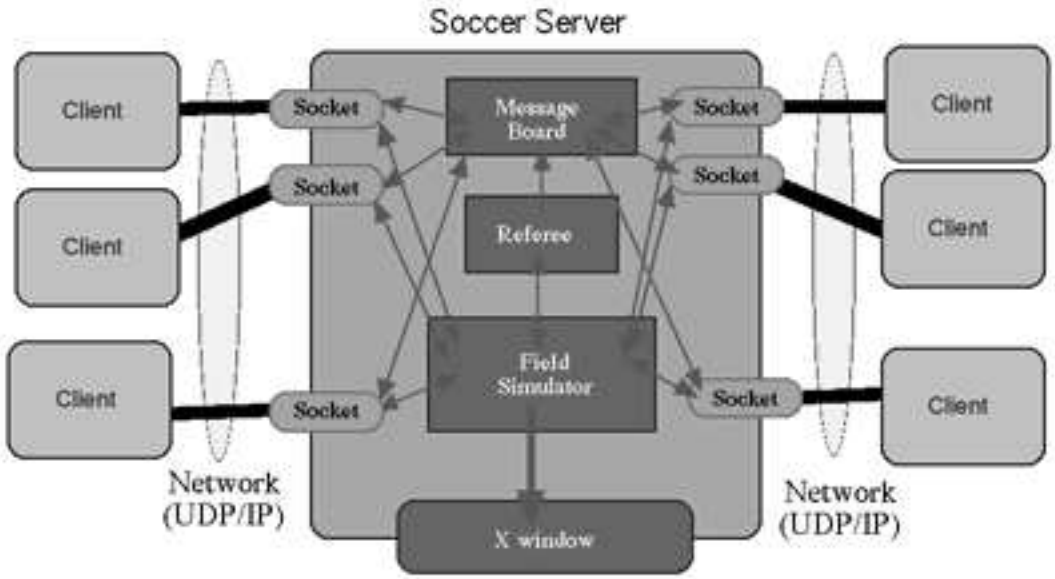

Figura C.1: Arquitetura interna do simulador SoccerServer

A partida acontece em dois tempos de 3000 ciclos de simulação; em torno de 5 minutos. Todo controle é feito pelo simulador. Para isto, ele lança mão dos chamados *modos de jogo*. Estes modos de jogo correspondem às diversas situações que podem ser encontradas em um partida de futebol real, como saídas de bola, impedimentos e faltas. Abaixo estão listadas todos as situações previstas pelo simulador:

1. *before_kick_off*: partida suspensa; modo anterior ao início do jogo.
2. *corner_kick_side*: indica que será cobrado um escanteio.
3. *free_kick_side*: cobrança de uma falta por parte de uma equipe; também se aplica ao tiro de meta.
4. *goal_kick_side*: modo de jogo ativo para que a equipe que sofreu o gol possa reiniciar a partida.
5. *kick_in_side*: cobrança de um lateral.
6. *kick_off_side*: partida temporariamente suspensa; modo ativo após a marcação de um gol.

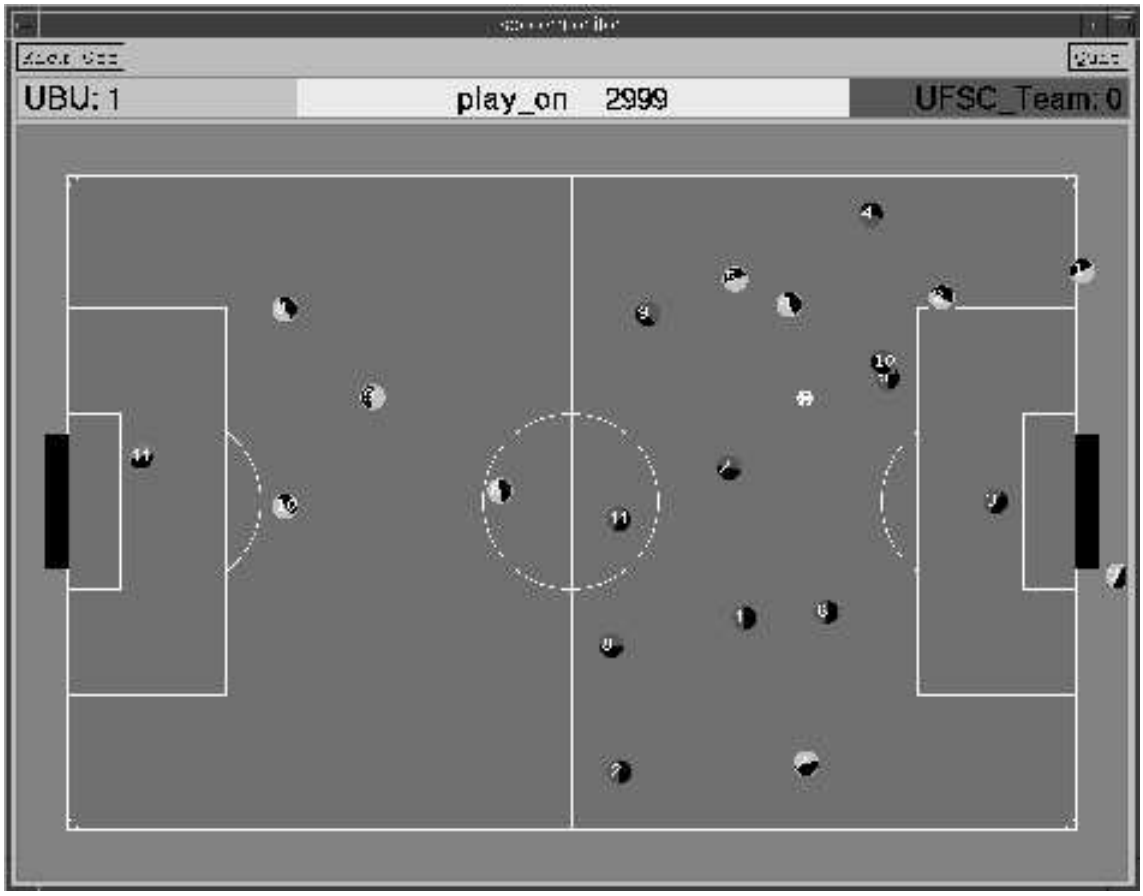

Figura C.2: Janela gráfica resultante da execução do programa soccermonitor

7. *offside_side*: indica que uma das duas equipes estava em condição de impedimento.
8. *play_on*: modo *default*; partida transcorrrendo normalmente.
9. *time_over*: fim de partida.

C.1 Breve Histórico

O primeiro sistema simulador SoccerServer foi desenvolvido por Itsuki Noda em setembro de 1993. Para sua programação foi utilizada um linguagem chamada de MWP, tipo de Prolog com suporte à multi-threads. Esta primeira versão exibia um campo virtual num terminal VT100 em modo texto. A primeira versão cliente/servidor foi escrita em julho de 1994 em LISP. Esta versão já "rodava" em ambientes X Window System, mas os objetos (jogadores e bola) ainda eram representados por caracteres alfanuméricos. Entre as inovações estava a utilização do protocolo UDP/IP para a comunicação com os clientes.

Em agosto de 1995, o simulador (batizado de versão 1), foi reescrito em C++ para atender as exigências de desempenho e anunciado num *workshop* da IJCAI'95. O desenvolvimento da segunda versão começou em janeiro do ano seguinte já tendo como objetivo principal tornar esta uma plataforma padrão para as pesquisas envolvendo simuladores. Nesta ocasião decidiu-se dividir o programa em duas partes distintas.

O esforço para se desenvolver um sistema cada vez mais fiel à realidade continua até hoje. As versões mais recentes exibem características que possibilitam aos clientes, e por conseqüência à partida em si, ações e comportamentos realmente próximos do real. A última versão estável disponível na Internet é a 9.2.2. Pode-se fazer o *download* de todo código fonte do simulador no seguinte endereço: `http://sserver.sourceforge.net/`.

C.2 As Regras do Jogo

Como em todo tipo de competição, aqui também existe um mediador, o árbitro. A diferença é que neste caso em especial, o árbitro também é um robô, um programa. O próprio simulador SoccerServer possui incorporado à si um módulo com as regras do jogo, aplicando-as sempre que necessário. Isto é feito de maneira totalmente automática e de forma continuada monitorando-se as posições de cada um dos jogadores. Como o simulador não é perfeito e devido à natureza imprecisa e incerta deste tipo de jogo, há também em toda partida oficial a presença de um juiz humano. As atribuições de cada um deles estão nas duas subseções seguintes.

C.2.1 Regras Julgadas pelo Simulador

Aqui estão detalhadas as regras que são aplicadas exclusivamente pelo simulador, sem interferência externa alguma. São elas:

- *Goal*

  Quando um time marca um gol, o árbitro anuncia o gol (através de um *broadcast* de uma mensagem para todos os cliente), atualiza o placar e suspende a partida por 5 segundos, move a bola para o centro do campo e muda o mode de jogo para *kick_off*.

- *Kick_off*

  Antes do início da partida ou log após acontecer um gol, todos jogadores devem estar em seus respectivos lados do campo. Se algum jogador estiver no lado do oponente, o árbitro o moverá para uma posição aleatória do seu campo. Os clientes têm 5 segundos para se posicionarem corretamente. Apenas durante este curto espaço de tempo é permitida a utilização do comando *move*. Esta instrução faz o reposicionamento do cliente de uma posição $(x1, y1)$ para outra $(x2, y2)$ sem, no entanto, ter que percorrer esta trajetória, ou seja, o jogador desaparece de um ponto e reaparece em outro.

- *Out_of_Field*

  Quando a bola está fora dos limites do campo, o árbitro move a bola para uma posição válida e mais próxima de uma linha de marcação (linha lateral, marca de escanteio, etc) e muda o modo de jogo para *kick_in*, *corner_in* ou *goal_in*. No caso de escanteio, o árbitro recoloca a bola a $(1m, 1m)$ para dentro do campo.

- *Clearance*

  Se o modo de jogo ativo no momento for um dos citados no item anterior, o árbitro remove todos jogadores defensores localizados dentro de um círculo centrado na bola com um raio de 9,15m. Os jogadores removidos são postos no perímetro deste círculo. Quando o mode de jogo é *offside*, todos jogadores defensores são movidos de volta a uma posição de não impedimento. Jogadores de ataque não podem re-entrar na área de pênalti enquanto o modo *goal_kick* estiver ativo. O mode de jogo *play_on* é ativo logo após a bola deixar a área. O mesmo acontece para os modos *kick_off*, *kick_in* e *corner_kick*.

- *Half-Time e Time-Up*

  O árbitro suspende a partida quando o primeiro tempo terminar. Se a partida permanecer empatada no final do segundo tempo, existe a prorrogação. O tempo de duração desta prorrogação não é fixo. O desempate acontece por morte súbita, ou seja, a equipe que marcar um gol primeiro vence a partida.

C.2.2 Regras Julgadas por um Humano

Além do árbitro robô, há também um juiz humano. Este se faz necessário devido ao fato de determinadas situações serem difíceis de serem avaliadas por serem subjetivas. Por este motivo o simulador oferece uma interface para intervenções. Deste modo o árbitro humano pode suspender uma partida e conceder *free_kicks* à equipe que julgar necessário. Apesar disto, é de praxe definir em quais situações este recurso será utilizado. À título de exemplo, os seguintes critérios costumam ser usados:

- envolver or sitiar a bola.
- bloquear a bola com vários jogadores (bola presa).
- explicitamente não (re)colocar a bola em jogo.
- intencionalmente bloquear a livre movimentação de jogadores adversários.
- excessivas invocações do comando *catch_ball* por parte do goleiro.
- sobrecarregar o servidor de conexões com mensagens com o objetivo de atrapalhar a partida.
- qualquer outra atitude julgada inadequada.

Referências Bibliográficas

Bares, J. e Whittaker, W. (1994). Dante ii. http://www.ri.cmu.edu/projects/project_163.html (acessado em 26/02/2003).

Barreto, J. M. (1999). *Inteligência Artificial: No limiar do século XXI*. Duplic, 2 edição.

Berry, A. (1999). Soccer robots with local vision. Dissertação de Mestrado, University of Western Australia.

Bittencourt, G. (2001). *Inteligência Artificial - Ferramentas e Teorias*. Editora da UFSC, 2 edição.

Brooks, R. (1986). A robust layered control system for a mobile robot. Em *IEEE Journal of Robotics and Automation*, volume 2, páginas 14–23.

Brooks, R. e Flynn, A. (1989a). Building robots: Expectations and experiences. Em *International Workshop on Intelligent Robot and Systems*.

Brooks, R. e Flynn, A. (1989b). Robot beings. Em *International Workshop on Intelligent Robot and Systems*.

Cruz, A. (1998). *Lógica Nebulosa*.

da Costa, A. L. (1996). Expert-coop: Um ambiente para desenvolvimento de sistemas multi-agents cognitivos. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

da Costa, A. L. (1997). Parla: A cooperation language for cognitive multi-agent systems. Em Spring-Verlag, editor, *Encontro Português de Inteligência Artificial (EPIA)*, páginas 207–215, Coimbra.

da Costa, A. L. e Bittencourt, G. (1998). Ufsc-team: A cognitive multi-agent approach to the robocup'98 simulator league. Em *Second RoboCup Workshop*, Paris.

da Costa, A. L. e Bittencourt, G. (1999). From a concurrent architecture to a concurrent autonomous agents architecture. Em *IJCAI'99 - Third RoboCup Workshop*, Estocolmo.

da Cunha, H. e Ribeiro, S. (1987). *Introdução aos Sistemas Especialistas*. Livros Técnicos e Científicos.

Electrolux (2003). Trilobite - robotic vacuum cleaner. http://trilobite.electrolux.com/ (acessado em 26/02/2003).

Ernst, G. e Simon, H. (1969). *GPS: A Case Study in Generality and Problem Solving*. Academy Press.

Everett, B. (2003). Robart. http://www.spawar.navy.mil/robots/land/robart/robart.html (acessado em 27/03/2003).

Everett, H. R. (1995). *Sensors for Mobile Robots:Theory and Application*. A K Peters Ltd., 1 edição.

Federation, R. (2003a). Robocup junior. http://www.robocupjunior.org (acessado em 21/02/2003).

Federation, R. (2003b). Robocup rescue. http://www.r.cs.kobe-u.ac.jp/robocup-rescue/ (acessado em 21/02/2003).

Ferreira, P. e Wainer, J. (2000). Scheduling meetings through multi-agent negotiation. Em *Simpósio Brasileiro de Inteligência Artificial*.

Frigo, L. B. (2002). Um ambiente interativo multiagente para o ensino de estrutura da informação. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Games, R. W. C. S. e Conferences (2002). Regulamento categoria smallsize (f180). http://www-2.cs.cmu.edu/%7Ebrettb/robocup/index.html (acessado em 13/02/2003).

Gonçalves, E. M. N. (2001). Otimização de controladores nebulosos e sistemas especialistas reativos utilizando algorítmos genéticos. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Jones, J. L., Flynn, A. M., e Seiger, B. A. (1999). *Mobile Robots: Inspiration to Implementation*. A K Peters, 2 edição.

Kitano, H. (2000). Robocup rescue: A grand challenge for multi-agent systems. Em *4th International Conference on MultiAgent Systems*, páginas 5–12.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., e Osawa, E. (1997). RoboCup: The robot world cup initiative. Em Johnson, W. L. e Hayes-Roth, B., editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, páginas 340–347, New York. ACM Press.

Kurzweil, R. (1990). *The Age of Intelligent Machines*. MIT Press.

Levine, W. S. (1996). *The Control Handbook*. IEEE Press.

Marchi, J. (2001). Navegação de robôs móveis autônomos: Estudo e implementação de abordagens. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Martinez, A., Tunstel, E., e Jamshidi, M. (1993). Fuzzy logic based collision avoidance for a mobile robot. Em *Third International Conference on Industrial Fuzzy Control and Intelligent Systems*, páginas 66–69.

Morrison, J. C. e Nguyen, T. T. (1996). On-board software for the mars pathfinder microrover. Em *Proceedings of the Second IAA International Conference on Low-Cost Planetary Missions*.

Motorola (1995). *68332 User's Manual*.

Motorola (1996). *M68332 32-Bit Modular Microcontroller*.

NASA (1999). Pioneer. http://www.frc.ri.cmu.edu/projects/pioneer/ (acessado em 26/02/2003).

NASA (2002). Mars exploration rover mission. http://mars.jpl.nasa.gov/missions/past/pathfinder.html (acessado em 27/03/2003).

NASA (2003). Mars exploration rover mission. http://mars.jpl.nasa.gov/mer/ (acessado em 21/02/2003).

Nehmzow, U. (2000). *Mobile Robotics: A Practical Introduction*. Springer-Verlag.

Nelson, W. R. (1982). An expert system for diagnosis and treatment of nuclear reactor accidents. Em *AAAI Proceedings*.

Noda, I., Asada, M., Matsubara, H., Veloso, M., e Kitano, H. (1999). Robocup as a strategic initiative to advance technologies. Em *IEEE International Conference on Systems, Man and Cybernetics*, volume 6, páginas 692–697.

Nof, S. Y. (1999). *Handbook of Industrial Robotics*. John Wiley & Sons, 2 edição.

Nunes, L. (2002). On learning by exchanging advice. Em *Second Symposium on Adaptative Agents and Multi-Agent Systems*.

Orth, A. (2001). Desenvolvimento de um sistema de visão para medir o desgaste de flanco de ferramentas de corte. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Pazos, F. (2002). *Automação de Sistemas e Robótica*. Editora Axcel.

Prouskas, K., Patel, A., Pitt, J., e Barria, J. (2000). A multi-agent system for intelligent network load control using a market-based approach. Em *4th International Conference on MultiAgent Systems*, páginas 231–238.

Rabuske, R. A. (1995). *Inteligência Artificial*. Editora da UFSC.

Rich, E. (1988). *Inteligência Artificial*. McGraw-Hill.

Russel, S. J. e Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Saygin, A. P. (2003). The turing test page. http://cogsci.ucsd.edu/ asaygin/tt/ttest.html (acessado em 28/02/2003).

Schalkoff, R. J. (1990). *Artificial Intelligence: An Engineering Approach*. McGraw-Hill.

Sony (2003). Aibo. http://www.us.aibo.com/ (acessado em 26/02/2003).

Stone, H. W. (1996). Mars pathfinder microrover: A small, low-cost, low-power spacecraft. Em *Proceedings of the 1996 AIAA Forum on Advanced Developments in Space Robotics*.

Tadokoro, S., Kitano, H., Takahashi, T., Noda, I., Matsubara, H., Shinjoh, A., Koto, T., Takeuchi, I., Takahashi, H., Matsuno, F., Hatayama, M., Ohta, M., Tayama, M., Matsui, T., Kaneda, T., Chiba, R., Takeuchi, K., Nobe, J., Noguchi, K., e Kuwata, Y. (2000a). The robocup-rescue: An it challenge to emergency response problem in disaster. Em *IEEE International Conference on Industrial Electronics Society*, volume 1, páginas 126–131.

Tadokoro, S., Kitano, H., Takahashi, T., Noda, I., Matsubara, K., Shinjoh, A., Koto, T., Takeuchi, I., Takahashi, H., Matsuno, F., Hatayama, M., Nobe, J., e Shimada, S. (2000b). The robocup-rescue project: A robotic approach to the disaster mitigation problem. Em *IEEE International Conference on Robotics and Automation*, volume 4, páginas 4089–4094.

Transrobotics (2003). Automatic guided vehicles. http://www.transbotics.com/ProductsServices/AGVs/AGVs.html.

Washington, R., Golden, K., Bresina, J., Smith, D. E., Anderson, C., e Smith, T. (1999). Autonomous rovers mars exploration. Em *Proceedings of the 1999 IEEE Aerospace Conference*.

Waterman, D. A. (1985). *A Guide to Expert Systems*. Addison-Wesley.

Zadeh, L. A. (1965). Fuzzy sets. Em *Information and Control*, volume 8, páginas 338–352.

Zlot, R. M., Stentz, A. T., Dias, M. B., e Thayer, S. (2002). Market-driven multi-robot exploration. Technical Report CMU-RI-TR-02-02, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.