Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Relatório de Projeto de Graduação II

Guiagem do Robô Móvel XR4000 para Inspeção via Internet de Tubulações Industriais Soldadas

POR

Sérgio Roberto Gonsalves Tourino

Relatório submetido como requisito para obtenção do grau de Engenheiro Mecânico

Orientador:

Prof. Alberto José Álvares

Brasília, 23 de junho de 2000.

Agradecimentos

Ao Departamento de Engenharia Mecânica da UnB e ao GRACO pelas condições de efetuação do trabalho.

Ao Prof. Alberto José Álvares pela orientação e incentivo.

À Prof. Flávia Guerra pelo auxílio prestado no ajuste do sistema de controle Fuzzy.

Aos colegas e amigos pelo apoio durante o desenvolvimento do projeto.

Ao meu irmão pelo auxílio em aspectos da Ciência da Computação.

E aos meus pais pelo apoio.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A tecnologia de teleoperação relaciona as técnicas de telecomunicações com o controle remoto de aparelhos. O crescimento do uso da Internet mostra que esta tecnologia pode ser usada para estabelecer sistemas de teleoperação entre máquinas e dispositivos, tais como robôs de soldagem ou robôs móveis. Este projeto tem como objetivo o controle de um robô móvel através da Internet para a inspeção remota de tubulações soldadas. O sistema será usado pelo usuário através da interface WWW, com o uso de um "applet" Java que mostra imagens do local remoto assim como informações sensoriais. O usuário, através de um "joystick" virtual, será capaz de controlar o robô e monitorar a qualidade da solda. Este sistema pode causar uma grande redução nos custos de inspeção a locais distantes. O projeto foi organizado nos seguintes tópicos: pesquisa bibliográfica, desenvolvimento da arquitetura do sistema, implementação, testes e otimização do mesmo.

Abstract

The teleoperation technology relates the telecomunication methods with the control of remote devices. The growth of Internet use shows that this technology can be used to stablish teleoperation among machines and devices, such as robots for welding or mobile ones. This project aims to control a mobile robot through the Internet for remotely inspecting welded pipes. The system will be used by an operator in a common WWW interface, through a Java applet that shows images from the remote site as well as sensorial information. The user, through a virtual joystick, will be able to control the robot and monitor the weld quality. This system can cause a substantial reduction in costs of inspection on long distance sites. The project is organized in the following steps: bibliographical research, development of a system architeture, implementation and ¯nally evaluation and optimization of the system.

Glossário

Abreviaturas e acrogramas adotados no texto

Sigla	*Significado (Inglês)*	*Significado (Português)*
API	Application Programming Interface	Interface de programação de aplicativos
CGI	Common Gateway Interface	
CPU	Central Processing Unit	Unidade central de processamento
DSP	Digital Signal Processing	Processamento digital de sinais
GUI	Graphical User Interface	Interface gráfica para usuário
HTML	Hiper Text Markup Language	Linguagem de hipertextos
IP	Internet Protocol	Protocolo Internet
IR	Infrared	Infravermelho
ISO	International Organization for Standardization	Organização Internacional de Padronização
JVM	Java Virtual Machine	Máquina virtual Java
MIME	Multipurpose Internet Mail Extensions	Extensões multipropósito para correio na Internet
NTSC	National Television Standards Committee	Comitê nacional de padrões de televisão
OSI	Open System Interconnect	Interconexão de sistemas abertos
PCI	Peripheral Component Interconnect	Interconexão de componentes periféricos
TCP	Transmission Control Protocol	Protocolo de transmissão de controle
TIG	Tungsten Inert Gas	Soldagem por arame de tungstênio
UDP	User Datagram Protocol	Protocolo de datagramas de usuário
WWW	World Wide Web	Rede mundial de computadores

Capítulo 1

Introdução

1.1 Estado Tecnológico

A existência de ambientes insalubres ou impróprios à ação humana, como centrais nucleares, levou ao desenvolvimento de sistemas remotos de operação. Entre esses sistemas encontra-se grande aplicação na soldagem remota [1], onde tochas TIG são utilizadas para a soldagem de tubulações em ambientes radioativos. Com uma outra abordagem tecnológica, a utilização de robôs móveis para visualização e monitoramento de ambientes perigosos também alcança grande desenvolvimento [2], como mostra a figura 1.1. Recentemente, pesquisadores [3] estudam o controle de robôs móveis através da utilização da Internet como veículo de transmissão de dados, com fins de pesquisa acadêmica.

Figura 1.1: O robô de inspeção RIES.

1.2 Proposta de Trabalho

O objetivo deste projeto é o desenvolvimento de um sistema de guiagem para o robô XR4000 por meio de teleoperação através da Internet, voltado para a inspeção de tubulações soldadas.

O desenvolvimento do projeto será baseado na arquitetura cliente-servidor aplicada à tecnologia de teleoperação via Internet. O robô, programado através da linguagem C, será comandado remotamente pelo usuário através de uma interface gráfica baseada na linguagem de programação Java sob o ambiente WWW.

O capítulo 2 apresentará uma revisão bibliográfica sobre a tecnologia de teleoperação, robótica móvel e comunicação de dados. A seguir, no capítulo 3, serão apresentados os requisitos de projeto, a arquitetura proposta para o sistema, além da interface de programação do robô XR4000 e aspectos tecnológicos de teleoperação via Internet. O capítulo 4 apresenta os resultados obtidos no projeto até o momento. Conclusões sobre o desenvolvimento do projeto são apresentadas no capítulo 5.

Capítulo 2

Revisão Bibliográfica

Neste capítulo serão abordados tópicos sobre teleoperação, robótica móvel e comunicação de dados via Internet.

2.1 Teleoperação

A teleoperação é definida como o controle contínuo e direto de um teleoperador (máquina remota). Inicialmente desenvolvida para a manipulação de materiais radioativos, a teleoperação permite que um operador exerça força e realize movimentos sobre uma máquina remota, e ainda receba realimentação sensorial, geralmente através de dados visuais, sonoros ou táteis. Com a introdução da tecnologia de teleoperação, foi possível o desenvolvimento de interfaces capazes de prover uma interação satisfatória entre homem e máquina, permitindo que serviços de grande destreza fossem realizados.

Um grande número de esquemas de classificação para descrever a teleoperação foram propostos. Um desses categoriza os sistemas de teleoperação tomando como base o grau de automação do sistema. Em um espectro variando da mínima para a máxima autonomia, a teleoperação pode ser classificada como [4]:

- controle manual sem auxílio computacional;
- controle manual com significativo auxílio ou transformação computacional;
- controle supervisório com predomínio do controle realizado pelo operador humano;
- controle supervisório com predomínio do controle realizado pelo computador;
- controle completamente automático, onde os operadores humanos observam o processo sem intervenções.

A teleoperação requer a sinergia entre homem e máquina. Diversos modelos relacionam esta interface, sendo os seguintes os tipos principais [5]:

- Modelo Mestre-Escravo

  Este modelo descreve o mais tradicional sistema de teleoperação. Neste caso, o operador humano observa o ambiente de trabalho remoto através de um sistema de vídeo e manipula o braço robótico mestre por meio de um "console", que controla o braço escravo no local remoto. A estrutura mestre-escravo provê uma interface intuitiva para o controle remoto de sistemas. Embora esse modelo esteja muito evoluído, com o uso de vídeos estéreis e realimentação de força, a maior desvantagem desse sistema é a sua falta de destreza e o cansaço físico que impõe ao operador.

- Modelo de Telepresença

  Em um esforço de alcançar uma alta fidelidade nos canais de comunicação entre o mestre e escravo do modelo anterior, foram desenvolvidos sistemas antropomórficos para teleoperação, de forma a oferecer uma melhor forma de transmissão das capacidades humanas de solução de problemas e de manipulação em ambientes hostis. Com a meta de prover um sistema transparente de interface homem-máquina, os sistemas de telepresença utilizam-se de "displays" montados na cabeça, sensores de movimento montados no corpo do operador, realimentação de força, entre outras tecnologias. A meta final desses sistemas é fazer o operador sentir-se presente no local de trabalho remoto, obtendo-se assim melhores condições de realização de tarefas. O custo técnico da implementação de tais sistemas, entretanto, não é justificado. O cansaço provocado ao operador devido ao volume e peso do equipamento, e ainda a falta de necessidade de um sistema de telepresença, reduzem o uso desse modelo principalmente à pesquisa e não para utilizações práticas.

- Modelo Professor-Aluno

  Dado que o aprendizado de sistemas computacionais é uma das áreas mais difíceis na inteligência artificial, o modelo professor-aluno define como função do professor ao operador humano, e assume que o "aluno" robô possui inteligência suficiente para reconhecer e atuar em uma situação já aprendida. Embora essas tecnologias ainda devam ser desenvolvidas e integradas para realizar esse modelo, as vantagens potenciais oferecidas por este conceito, em termos de conforto e efetividade, são substanciais.

- Modelo Supervisor-Companheiro

  De acordo com esse modelo, um robô baseado em sensores não deve simplesmente repetir os movimentos do operador humano, como no modelo mestre-escravo. Neste caso, o operador humano serve como um supervisor, ao invés de projetar-se no ambiente remoto. Com a companhia do operador humano, o sistema robótico incorpora capacidades computacionais, como precisão e capacidades sensoriais, para a realização das tarefas. A comunicação homem-robô pode ser facilitada com o uso de gráficos interativos, controle com vários graus de liberdade, e interfaces híbridas.

Uma vez estabelecida e efetivada a comunicação entre homem e máquina, deve-se observar a relação entre a máquina remota e seu ambiente. Dessa forma, são propostos esquemas de classificação de modelos de ambientes:

- Ambiente Remoto Totalmente Modelado

  Nesta categoria, os objetos sendo manipulados, o ambiente e os procedimentos operacionais são repetitivos ou variantes, mas previsíveis. No primeiro caso, um robô programável pode realizar suas tarefas com mínima intervenção humana. No último, um operador humano escolhe estratégias de ação pré-programadas, consideradas como uma linguagem de programação do sistema.

- Ambiente Remoto Parcialmente Modelado

  Nesta categoria incluem-se operações em ambientes "humanos", como plataformas espaciais ou plantas industriais, onde todos os potenciais procedimentos operacionais das tarefas não podem ser antecipados em suficientes detalhes para uma pré-programação efetiva. Nesses casos, entretanto, é possível algum conhecimento geométrico do ambiente, podendo ser modelado *a priori*, mesmo que as relações espaciais entre o local e seus objetos não sejam conhecidas exatamente.

- Ambiente Remoto Desconhecido

  Esta categoria difere da anterior pelo fato de que pouca ou nenhuma informação sobre o ambiente e seus objetos é conhecida *a priori*. Exemplos desse tipo são os de robótica submarina, mineração, limpeza de resíduos nucleares e robótica militar.

2.2 Robótica Móvel

2.2.1 O Robô Móvel Nomad XR4000

O robô móvel NomadXR4000 (figura 2.1), produzido pela Nomadic Technologies [6], é um sistema avançado de robótica móvel, compreendendo sistemas de controle, sensores, comunicação e programação necessários para pesquisas e desenvolvimentos na área de manipulação robótica, visão computacional, navegação por sensores e aprendizado. Suas especificações físicas são listadas a seguir:

- Dimensões: 620 mm (diâmetro) × 850 mm (altura);
- Peso: 150 Kg (com baterias);
- Máxima aceleração translacional: $5 m/s^2$;
- Máxima aceleração rotacional: $2 rad/s^2$;
- Número de rodas: 4;
- Capacidade das baterias: 1575 $W:h$;
- CPU: até 3 Pentium ou Pentium Pro;

Figura 2.1: O robô móvel Nomad XR4000.

Nas seções seguintes serão especificados os subsistemas componentes do robô XR4000 [7].

Subsistema Motor

O sistema motor do robô XR4000 baseia-se num sistema holonômico, provendo três graus de liberdade (x, y e μ), com baixa complexidade mecânica e boa confiabilidade. Suas rodas possuem eixos de translação e rotação independentes, somando oito

motores para as quatro rodas do sistema. O controle é realizado por três DSP (Digital Signal Processing) e um microcontrolador dedicado, para o controle dos oito eixos e para cálculo da sua posição estimada (\dead reckoning").

Subsistema Sensorial

O robô móvel ISR4000 possui, como padrão, três tipos de sensores: sensores táteis, sensores por ultrasom ou sonar, e sensores por infravermelho. A seguir será descrito o funcionamento de cada um dos sensores.

1. Sensores Táteis

   Também denominados de sensores de colisão, são sensores que retornam informações sobre contato físico com objetos no ambiente. Embora seja desejável que os sensores de proximidade (sonar e infravermelho) sintam e previnam o contato com obstáculos, isso não é garantido. Os sensores de colisão do Nomad SR4000, sistema *Sensus 150*, possuem dois níveis de sensibilidade, para contatos fracos e fortes.

2. Sonares

   O sistema *Sensus 250* consiste em dois anéis contendo 24 sonares, provendo informações sobre a distância de objetos \distantes" (de 150 a 7000 mm). Essa informação é calculada através do conhecimento da velocidade do som e do tempo de viagem do pulso emitido e reº etido. Matematicamente:

   $$d_{objeto} = v_{som} . \frac{t_{viagem}}{2}$$

   O sonar baseia-se no emissor ultrasônico Polaroid *6500*, que emite 16 ciclos de onda quadrada de $49{:}4\,kHz$ através de um transdutor eletrostático, que também funciona como receptor após o disparo do pulso. Como pode ser visto na Figura 2.2, o transdutor não emite energia uniformemente em todas as direções, sendo maior na direção normal ao sensor.

   Devido a características do circuito eletrônico utilizado no sensor, três erros de medida podem ocorrer:

   - Duração do Pulso Transmitido
   - Amplificador de Ganho Variável com o Tempo
   - Carga Capacitiva do Circuito de \Threshold"

Como os sensores baseiam-se na emissão de ondas sonoras, a determinação da distância de um objeto depende das características desses objetos:

- Objetos *Refletores*, com dimensões superiores ao comprimento da onda sonora (6.95 *mm* a 20º *C*).
- Objetos *Difratantes*, com dimensões inferiores ao comprimento da onda sonora. Embora objetos dessa dimensão sejam raros, superfícies rugosas como concreto podem apresentar características difratantes, reduzindo a efetividade do sensor.

A figura 2.2 apresenta dados típicos de retorno de sensores ultrasônicos.

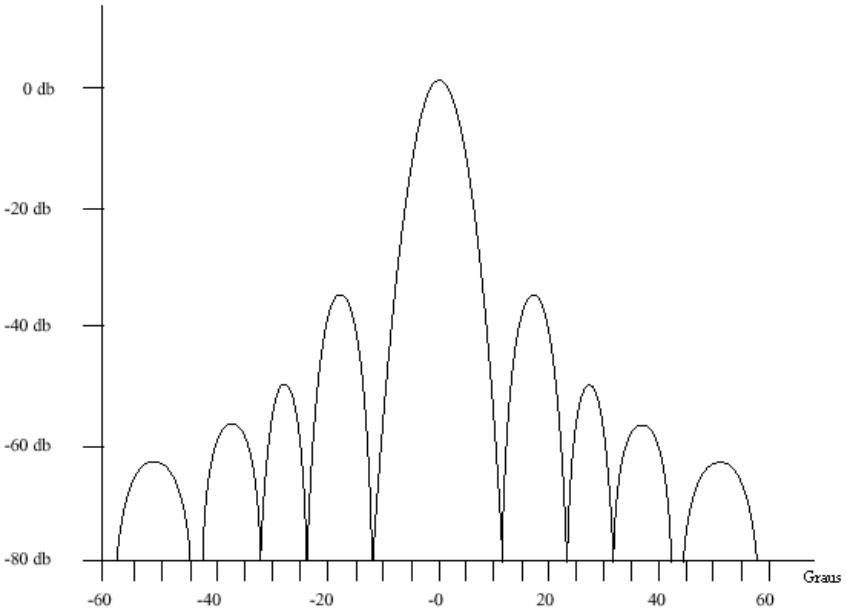

Figura 2.2: Recepção do sensor ultrasônico.

3. Sensores por Infravermelho

O sistema de sensores por luz infravermelha, sistema *Sensus 350*, consiste em dois anéis de 24 sensores, provendo informações de presença de objetos próximos (menos de 500 mm de distância).

Cada sensor é composto de dois emissores infravermelhos SIEMENS SFH 34-3 GaAs e um receptor SIEMENS SFH 2030F (fotodiodo). Para obter a energia refletida por um objeto, são tomadas duas leituras: uma com os emissores desligados - obtendo-se o nível de radiação do ambiente, e uma com os emissores ligados - obtendo-se a energia refletida mais a radiação ambiental. A energia referente ao objeto é obtida da subtração das leituras obtidas:

$$E_{objeto} = E_{ligado} - E_{desligado}$$

Diversos fatores influenciam os dados obtidos pelo sensor:

- Geometria

  O princípio de funcionamento do sensor é baseado no fato de que quanto maior a distância do objeto, menor será a energia refletida captada pelo receptor. Além disso, o ângulo do objeto também é de grande importância. Para um objeto de superfície normal ao eixo do emissor, a energia refletida irá quase toda para o receptor. Ao contrário, para um objeto de superfície quase paralela ao eixo do emissor, a energia que chegará ao receptor será mínima, indicando uma distância maior que a verdadeira.

- Iluminação

  A presença de uma grande quantidade de energia infravermelha no ambiente, como no caso de locais iluminados por lâmpadas incandescentes, pode saturar o receptor, reduzindo sua sensitividade. Em geral, o uso de lâmpadas fluorescentes quase não afeta as medidas de sensores infravermelhos.

- Cor e Superfície

  A "cor" do objeto (sua refletividade à luz infravermelha) possui grande influência nas leituras obtidas (figura 2.3). Além disso, a rugosidade superficial também influi na reflexão do sinal, sendo maior para superfícies lisas que rugosas.

Subsistema de Visão

Como auxílio ao subsistema sensorial, o robô XR4000 possui o sistema *Sensus 460*, que consiste numa câmera de vídeo colorida, padrão NTSC, e uma placa de captura de vídeo, padrão PCI, com $45\ Mbytes/s$ de transferência de dados. Esse sistema é auxiliado por uma unidade "Pan-Tilt", controlada por uma porta serial RS-232, para movimentação da câmera.

Esse sistema é adequado, devido à sua operação em tempo real, para movimentação visual (através das imagens), análise de movimentos, assim como aplicações diversas.

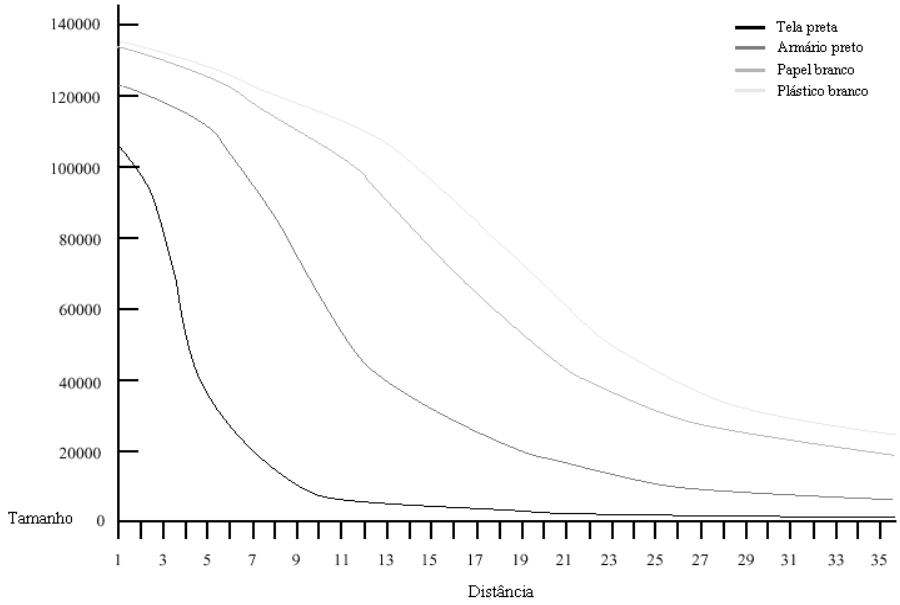

Figura 2.3: Recepção do sensor infravermelho.

Subsistema de Comunicação

A comunicação do robô XR4000 com a rede local é realizada através da interface Proxim RangeLan 2 (freqüência de 2.4 GHz), um rádio Ethernet que provê ao sistema completo acesso à rede (incluindo serviços como \telnet", \ftp" e TCP/IP \sockets").

2.2.2 Navegação de Robôs Móveis

Navegação é a ciência (ou arte) de direcionar o curso de um robô móvel à medida que o mesmo atravessa o ambiente [8]. Em todo sistema de navegação deseja-se alcançar o destino sem que o robô se perca ou colida em algum obstáculo. A navegação envolve três tarefas: mapeamento, planejamento e direção. Um processo de alto nível, denominado de planejamento de tarefas, especifica o destino e restrições para o sistema, como o tempo.

De forma simples, o problema da navegação é encontrar um caminho ligando o local atual inicial até uma meta, e atravessá-lo sem colisões.

A primeira parte da navegação, mapeamento, é realizada através do uso ou de mapas pré-armazenados ou de mapas \aprendidos" pelo sistema sensorial à medida que o robô atravessa o ambiente. A modelagem do ambiente é realizada pela análise dos dados sensoriais para a construção e modificação dos mapas.

O planejamento é feito através da procura de caminhos possíveis no mapa construído. A melhor alternativa é então escolhida de forma a atender às restrições impostas pela tarefa.

A terceira tarefa realizada na navegação é a guiagem do robô através do caminho definido anteriormente. O movimento do robô é então controlado utilizando-se o seu modelo cinemático e dinâmico. Durante a movimentação, o processo de percepção examina continuamente os dados sensoriais a fim de detectar colisões potenciais. Quando uma colisão é possível, o processo de percepção inicia uma ação evasiva. O nível final de segurança em todo robô móvel é obtido através da detecção de colisão por sensores de toque ou contato.

A figura 2.4 mostra a interrelação entre as funções de navegação de um robô móvel.

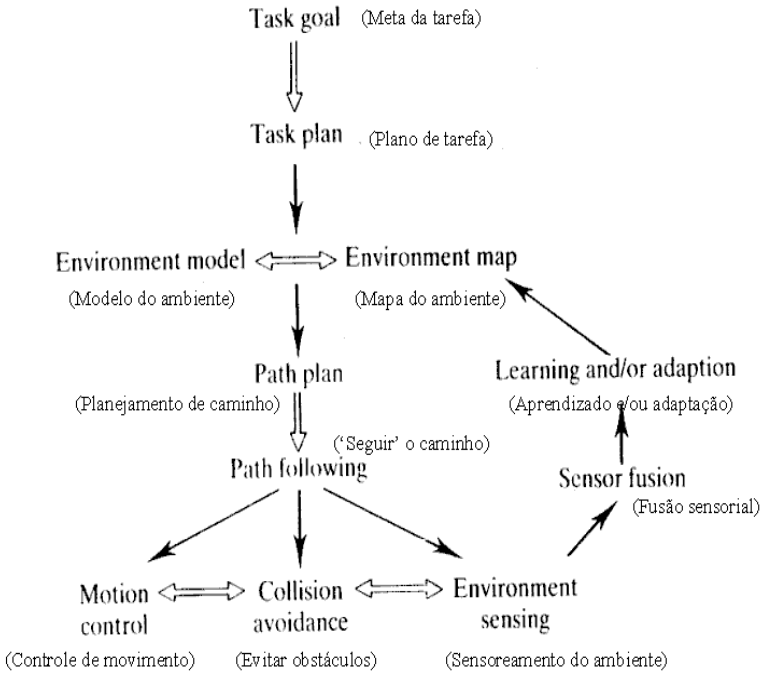

Figura 2.4: Hierarquia das funções de navegação [8].

Pesquisas recentes em navegação buscam a integração entre sistemas de odometria (\dead-reckoning") e sensores de obstáculos (sonares ou laser), de forma a gerar um mapeamento confiável do ambiente do robô [9].

2.2.3 Teleoperação de Robôs Móveis via Internet

O desenvolvimento de sistemas de teleoperação via Internet de robôs móveis é amplamente explorado por diversos pesquisadores [10, 11, 12, 13]. Diversas especificações para esses sistemas foram definidas [10], gerando as seguintes regras práticas:

- o sistema deve ser imune a atrasos de transmissão;
- altos tráfegos na rede não devem impedir a conexão com o usuário;

- a transmissão de vídeo deve ser fluente suficiente para prover um grau razoável de realismo no controle;
- a interface com o usuário deve ser simples de se usar;
- a estratégia de controle do sistema deve ser intuitiva.

Grande parte dos trabalhos apresenta soluções baseadas nos conceitos de CGI, Java e CORBA, de forma a prover diversos graus de interatividade com o usuário. O requisito de imunidade a atrasos requer um mínimo grau de autonomia do sistema. A autonomia pode ser implementada nos seguintes níveis básicos [13]:

- evitar obstáculos;
- navegação;
- planejamento de caminhos.

O requisito mínimo para sistemas teleoperados é a capacidade do mesmo de evitar obstáculos, de forma que o comando remoto não seja totalmente responsável pela segurança do robô e de seu ambiente. A navegação e o planejamento de caminhos são geralmente utilizados quando o ambiente possui obstáculos estáticos (como mobília e equipamentos) e deseja-se simplificar o controle por parte do usuário.

O desenvolvimento de sistemas de teleoperação via Internet requer, por parte do sistema, uma autonomia mínima que assegure a sua segurança, e por parte do usuário, uma interface de controle simples e intuitiva.

2.3 Comunicação de Dados via Internet

Nesta seção serão apresentados os conceitos de modelo cliente-servidor, o modelo ISO-OSI para comunicação de dados, e o conceito de canais de comunicação.

2.3.1 O Modelo Cliente-Servidor

O modelo da arquitetura cliente-servidor é um modelo de sistemas distribuídos que mostra como os dados e processamentos são distribuídos entre um conjunto de processadores [14]. Os principais componentes desse modelo são:

- um conjunto de servidores independentes que oferecem serviços para outros subsistemas;
- um conjunto de clientes que requisitam serviços oferecidos pelos servidores;

- uma rede de computadores que permite que os clientes acessem esses serviços.

Os clientes devem saber os "nomes" dos servidores disponíveis e os serviços que os mesmos provêm. Entretanto, servidores não precisam conhecer ou a identidade dos clientes ou quantos clientes tentam obter seus serviços. Um exemplo de um sistema desenvolvido a partir do modelo cliente-servidor é mostrado na figura 2.5.

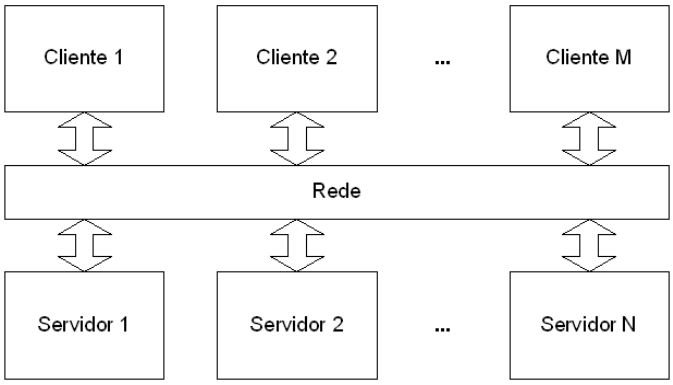

Figura 2.5: Exemplo de aplicação do modelo cliente-servidor.

A mais importante vantagem do modelo cliente-servidor é a sua fácil distribuição ou ampliação. É simples adicionar um novo servidor e integrá-lo gradualmente com o resto do sistema, ou mesmo atualizar os servidores sem afetar outras partes do sistema.

2.3.2 O modelo ISO-OSI

Um sistema de comunicações representa a forma na qual se realizam as trocas de dados e a passagem das informações de controle. O deslocamento desses dados é geralmente definido a partir do modelo de comunicação OSI (Open System Interconnection), definido pela ISO.

O modelo OSI divide em sete camadas o trabalho de deslocamento de dados de um ponto a outro. São camadas hierarquizadas que contribuem para o agrupamento ou separação de um pacote de dados. Esses pacotes de dados, além da informação propriamente dita a ser transmitida, levam em seu interior dados relativos à origem e destino na rede, assim como o protocolo empregado na codificação das informações transportadas.

As camadas do modelo OSI são as seguintes:

- Camada Física

  É a camada responsável pela geração e recepção dos impulsos físicos (elétricos) envolvidos na comunicação de dados. Essa camada apenas trata da conexão

das demais camadas com o sistema de comunicação, não incluindo portanto a implementação física da rede de computadores.

- Camada de Enlace de Dados

  É o nível que realiza o primeiro tratamento dos sinais, agrupando-os em pacotes de bits e adicionando-lhes sistemas de verificação de erros.

- Camada de Rede

  Quando uma rede de computadores torna-se complexa devido ao seu tamanho, utiliza-se como solução a sub-divisão desta rede, através da criação de sub-redes locais. A camada de rede gerencia o transporte de dados através da rede, realizando o roteamento dessas informações.

- Camada de Transporte

  É um nível de transição que realiza o gerenciamento final dos pacotes de roteamento e a recuperação de erros das camadas anteriores.

- Camada de Sessão

  A camada de sessão é o nível que mantém as transmissões orientadas à conexão, realizando a abertura e fechamento de conexões entre máquinas da rede.

- Camada de Apresentação

  É uma camada pouco utilizada responsável pela conversão dos dados recebidos para a próxima camada, a camada de aplicação.

- Camada de Aplicação

  A camada de aplicação trata dos assuntos de segurança e disponibilidade de recursos, como transferência de arquivos e protocolos de terminais.

2.3.3 TCP e UDP

O TCP (Transmission Control Protocol) e o UDP (User Datagram Protocol) são protocolos que percorrem encapsulados nos pacotes IP (Internet Protocol), sendo responsáveis pela comunicação de dados baseados em datagramas e baseados em fluxo [15].

Um protocolo baseado em datagrama é aquele que atribui um tamanho máximo à quantidade de dados enviada em uma única transmissão (como o UDP). Um protocolo baseado em fluxo (como o TCP) realiza a "quebra" da informação em pacotes menores

e trata da reorganização desses pacotes, de forma que, para o usuário, a transmissão é realizada sem limites de quantidade de dados.

De forma comparativa, podemos diferenciar o UDP do TCP da seguinte forma:

1. UDP
   - baseado por datagrama;
   - menor consumo de recursos da rede;
   - menor confiabilidade;
   - quantidade de dados enviada por transmissão limitada.
2. TCP
   - baseado em fluxo;
   - maior consumo de recursos da rede;
   - maior confiabilidade;
   - quantidade ilimitada de dados transmitidos.

2.3.4 Canais de Comunicação

Canais de comunicação (geralmente denominados de "sockets") são a forma de se realizar a comunicação entre dois computadores em uma rede [16, 17]. Os canais ligam dois computadores através de seus endereços IP e através de uma "porta" de comunicação. Esta comunicação pode ser realizada tanto através do protocolo UDP como pelo protocolo TCP. Comandos como "telnet" e "ftp" utilizam internamente em seu funcionamento o conceito de canais de comunicação.

De forma genérica um canal de comunicação é aberto especificando-se o endereço da máquina destino e a porta de comunicação a ser utilizada. Para o caso de uma comunicação TCP, a máquina remota recebe o endereço da máquina local, o que permite que uma conexão baseada em fluxo seja realizada entre as máquinas. A figura 2.6 mostra um esquema do processo.

Figura 2.6: Esquema de uso de canais de comunicação.

2.3.5 O servidor Inetd

Nos sistemas UNIX, como o Linux, existe um processo responsável pela maior parte das conexões realizadas através de canais de comunicação. Esse programa, denominado inetd, aguarda conexões de rede e as direciona para programas servidores, como o telnetd, responsável por conexões telnet.

Como mostra a figura 2.7, o inetd, ao receber uma conexão da rede, cria uma cópia de si, através do comando fork, e então executa o programa servidor através do comando exec [18].

A configuração do inetd é armazenada no arquivo \etc\inetd.conf. A sintaxe dos dados contidos no arquivo segue a seguinte forma [15]:

serviço canal protocolo espera usuário localização argumentos

Os termos referidos acima possuem os seguintes significados:

serviço é o nome do servidor, listado no arquivo /etc/services.

canal descreve o tipo de canal utilizado, ou seja, stream ou dgram.

protocolo é o protocolo utilizado pelo serviço, seja tcp ou udp.

espera configura o inetd a travar a porta (wait) a novas conexões ou não travar a porta (nowait) a novas conexões.

usuário indica sob qual usuário o servidor deve ser ativado (geralmente o super-usuário root).

localização indica a localização e nome do servidor a ser ativado pelo inetd.

argumentos descreve os argumentos a serem passados ao programa servidor.

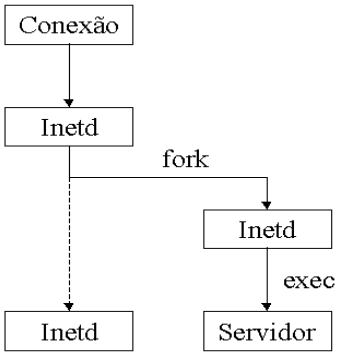

Figura 2.7: Funcionamento do *daemon* inetd.

Assim, por exemplo, o servidor telnet é geralmente configurado no arquivo inetd.conf da seguinte forma:

```
telnet  stream  tcp  nowait  root  /usr/etc/in.telnetd  in.telnetd
```

Deve-se lembrar que o servidor deve estar listado no arquivo /etc/services, onde é definida a porta que o servidor atende. No caso do servidor de telnet, sua configuração é a seguinte:

```
telnet  23/tcp
```

onde temos a utilização da porta 23 sob o protocolo tcp.

A utilização do inetd é vantajosa sobre a programação convencional utilizando diretamente os canais de comunicação pois reduz a carga sobre o sistema operacional e facilita o desenvolvimento de programas que realizam comunicação via Internet.

2.4 Lógica Nebulosa ou *Fuzzy*

Nesta seção serão apresentados conceitos básicos sobre a lógica *Fuzzy*, sua metodologia e formas de aplicação mais usuais.

2.4.1 Introdução à Lógica *Fuzzy*

De forma simples, pode-se caracterizar a lógica *Fuzzy* como "um tipo de lógica que reconhece mais do que simples valores de verdadeiro ou falso. Com a lógica fuzzy, proposições podem ser representadas com graus de verdade e falsidade. Por exemplo, a frase, hoje está ensolarado, pode ser 100% verdadeira se não há nuvens, 80% verdadeira se há poucas nuvens, 50% verdadeira se está nublado e 0% verdadeira se chove durante todo o dia." [19].

Ou seja, a lógica *Fuzzy* provê um método de traduzir expressões verbais vagas, imprecisas e qualitativas em valores numéricos, permitindo que computadores sejam utilizados como sistemas inteligentes baseados na experiência humana [20].

Aspectos Qualitativos da Lógica *Fuzzy*

Vista de forma qualitativa, a lógica *fuzzy* pode ser apresentada da seguinte forma:

- Diferentemente da lógica aristotélica (bivalente - verdadeiro ou falso), a lógica *fuzzy* é multivalente, ou seja, a verdade é graduada, recebendo usualmente um valor no intervalo $[0; 1]$, sendo 0 representando completamente falso e 1 completamente verdadeiro;

- Expressões verbais, imprecisas, qualitativas, inerentemente humanas, podem ser tratadas através do formalismo da lógica *fuzzy*;
- As implicações lógicas, ou inferência lógica, as entradas e saídas, ou antecedentes e conseqüentes, são associados a graus de verdade no intervalo $[0; 1]$;
- A utilização da lógica *fuzzy* facilita a interface computador-homem, por permitir ao primeiro uma melhor compreensão da linguagem inerentemente imprecisa dos seus operadores.

Conjuntos *Fuzzy*

Seja um conjunto universo E e x um elemento desse conjunto. O conjunto A é o conjunto de pares ordenados tais que:

$$f[x; \mu_A(x)]g; 8x \in E$$

onde $\mu_A(x)$ é o grau de pertinência de x em A. O valor de $\mu_A(x)$, na lógica *fuzzy*, pode ser qualquer valor no intervalo $[0; 1]$.

Operações com Conjuntos

Assim como nos conjuntos convencionais, os conjuntos *fuzzy* possuem operadores de união, interseção e complemento. A seguir serão definidas as principais operações utilizadas.

- União

  A função de pertinência $\mu_{A \cup B}(u); u \in U$, da união $A \cup B$, é definida como:

  $$\mu_{A \cup B}(u) = \mu_A(u) \perp \mu_B(u) \equiv max[\mu_A(u); \mu_B(u)]$$

  onde $\perp$ é a co-norma triangular de união generalizada. Diversas normas $\perp$ podem ser utilizadas, dentre as quais:

União	$x \perp y = max(x; y)$
Soma algébrica	$x \perp y = x + y - xy$

- Interseção

  A função de pertinência $\mu_{A \cap B}(u); u \in U$, da interseção $A \cap B$, é definida como:

  $$\mu_{A \cap B}(u) = \mu_A(u) \top \mu_B(u) \equiv min[\mu_A(u); \mu_B(u)]$$

onde t é a norma triangular de interseção generalizada. Diversas normas t podem ser utilizadas, dentre as quais:

Interseção $x \top y = min(x; y)$

Produto algébrico $x \top y = xy$

Produto drástico $x \top y = \begin{cases} x; & y = 1 \\ y; & x = 1 \\ 0; & x; y < 1 \end{cases}$

- Complemento

A função de pertinência do complemento é dada como segue:

$$\mu_{A'}(x) = 1 - \mu_A(x)$$

Funções de Pertinência

Uma função de pertinência *fuzzy* é uma função numérica gráfica ou tabulada que atribui valores de pertinência para valores discretos de uma variável, dentro de seu universo de discurso (o universo de discurso representa o intervalo numérico de todos os valores possíveis que uma variável pode assumir).

As funções de pertinência assumem diversas formas, sendo as mais utilizadas a triangular e a trapezoidal. A figura 2.8 apresenta as principais funções de pertinência geralmente utilizadas em projetos de controladores *fuzzy*.

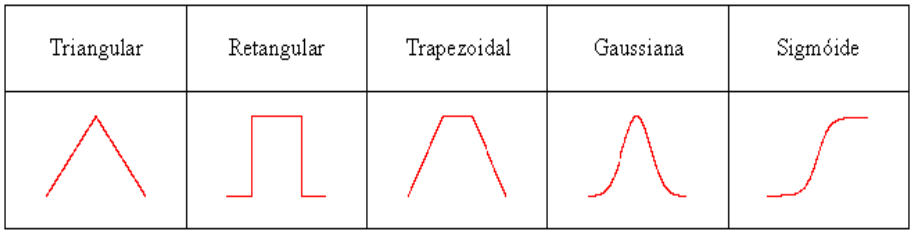

Figura 2.8: Funções de pertinência *fuzzy*.

A quantidade de funções de pertinência a ser utilizada em um universo de discurso e seus formatos são geralmente baseados na experiência. A pesar disso, algumas dicas práticas podem ser utilizadas:

- o número de funções de pertinência utilizado varia geralmente entre 2 e 7. Quanto maior o seu número, maior a precisão mas também é maior o custo computacional;

- o formato das funções de pertinência é geralmente triangular ou trapezoidal, mas caso seja necessário um comportamento mais suave do sistema, devem ser utilizadas funções como a gaussiana ou a sigmóide;
- a precisão do sistema é dependente do grau de superposição entre as funções de pertinência. Geralmente aconselha-se para testes iniciais 50% de superposição, e para o sistema otimizado um mínimo de 25% e um máximo de 75% de superposição em sistemas de malha fechada;
- caso haja dados experimentais, é possível utilizar sistemas neuro-*fuzzy* para gerar automaticamente as funções de pertinência;
- as funções de pertinência devem preferivelmente abranger todo o universo de discurso da variável analisada.

Fuzzy-ação

A *fuzzy*-ação é o mapeamento do domínio de valores numéricos (V) reais (como valores obtidos de um sensor) para valores *fuzzy* (VF), definidos pelas funções de pertinência. Como pode ser visto na figura 2.9, o valor *fuzzy* para a variável V é dado por 0:3 para a função Me por 0:7 para L.

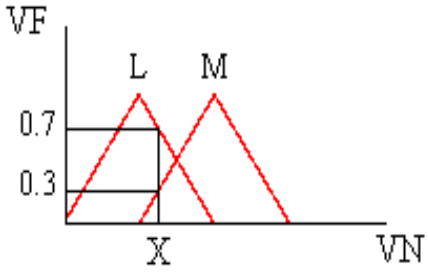

Figura 2.9: *Fuzzy*-ação de valores numéricos.

Regras

As regras *fuzzy* seguem dois tipos de implicações: *modus ponens* (modo afirmativo) e o *modus tollens* (modo negativo). A tabela 2.1 mostra a forma como essas implicações são utilizadas.

Uma regra *fuzzy* utilizando a implicação *modus ponens* toma a seguinte forma:

SE X=A ENTAO Y=B

Modo	*Modus ponens*	*Modus tollens*
Premissa 1	u=Ì	y=naoÌ
Premissa 2	se u=Ì entaoo y=B	se u=Ì entaoo y=B
ConseqüÊncia	y=B	y=naoB

Tabela 2.1: Métodos de implicaçoes *fuzzy*.

Generalizando, as regras *fuzzy*, utilizando as variáveis e suas funçoes de pertinência, possuem a seguinte sintaxe:

SE < variavel_1 = fp_1 > E/OU <variavel_2 = fp_2> E/OU ...
ENTAO < variavel_S = fp_S >

onde variavel_n refere-se a uma variável de entrada, fp_n a uma funçaoo de pertinência desta variável n, e variavel_S e fp_S referem-se a variável de saída e uma de suas funçoes de pertinência.

Inferência

A inferência *fuzzy* baseia-se na transformaçaoo do espaco de entradas $A(x)$ para o espaço de saidas $B(y)$ através de uma relaçaoo matricial $R(x;y)$. Matematicamente:

$$B(y) = A(x) \pm R(x;y)$$

onde $\pm$ é o operador composicional de inferência. Os operadores mais utilizados saao os seguintes:

Max-min $\quad \mu_B(y) = max\{min[\mu_A(x); \mu_R(x;y)]\}$
Max-produto $\quad \mu_B(y) = max\{\mu_A(x).\mu_R(x;y)\}$

*Desfuzzy*ficaçaoo

A *desfuzzy*ficaçaoo o valor das variáveis lingüÍsticas de saída será traduzido para um valor discreto numérico, com o objetivo de obter-se o melhor valor que represente o valor desejado. Existem diversos métodos de *desfuzzy*ficaçaoo, cada um apropriado para uma aplicaçaoo determinada:

² Centro da Área ou Método do Centróide

O método do centróide, o valor numérico retornado é calculado a partir da seguinte expressaoo:

$$u^* = \frac{\sum_{i=1}^{N} u_i \mu_{sai}(u_i)}{\sum_{i=1}^{N} \mu_{sai}(u_i)}$$

- Centro do Máximo ou *Desfuzzyficacao* pelas Alturas

  O método dos máximos realiza uma média ponderada dos máximos das funções de pertinência, segundo a seguinte fórmula:

$$u^* = \frac{\sum_{i=1}^{N} u_i \cdot \sum_{k=1}^{n} \mu_{O;k}(u_i)}{\sum_{i=1}^{N} \sum_{k=1}^{n} \mu_{O;k}(u_i)}$$

- Média do Máximo

  Esta abordagem utiliza o valor médio entre os máximos das funções de pertinência, segundo a seguinte expressão:

$$u^* = \sum_{m=1}^{M} \frac{u_m}{M}$$

  onde u_m é o m-ésimo elemento no universo de discurso onde $\mu_{sai}(u_i)$ possui um máximo.

2.4.2 Sistemas de Controle *Fuzzy*

Embora o controle convencional, baseado em controladores PID e funções de transferência, seja adequado para a maior parte das situações, as implementações por lógica *fuzzy* freqüentemente são mais eficientes, devido às seguintes características:

- o controle *fuzzy* nasce da experiência ao invés de modelos matemáticos, portanto uma implementação linguística é muito mais fácil e rápida de ser definida;
- condições raras ou excepcionais no controle podem ser incorporadas com pouco custo computacional, permanecendo o sistema ainda transparente e compreensível;
- a implementação da lógica *fuzzy* é freqüentemente mais eficiente em termos de codificação e tempo computacional de execução.

Um controlador *fuzzy* é geralmente composto dos seguintes blocos funcionais:

- interface de *fuzzyficacao*;
- base de conhecimento (regras);
- sistema de inferência;
- interface de *desfuzzyficacao*.

2.4.3 Aplicações da Lógica *Fuzzy*

A lógica *fuzzy* é aplicada em diversas áreas, seja comercial ou de pesquisa. Entre algumas aplicações comerciais podem ser citadas:

- Câmeras de Vídeo

  O controle de foco e abertura do diafragma de câmeras pode ser obtido através da lógica *fuzzy*, garantindo assim uma melhor qualidade das imagens obtidas e permitindo equipamentos mais simples e baratos.

- Máquinas de Lavar Roupa

  O controle dos ciclos da máquina é obtido através do sensoreamento de diversas características para a lavagem, como a temperatura da água, peso das roupas e grau de sujeira. Aplicando-se o conhecimento empírico humano a um controlador *fuzzy* é possível obter a lavagem das roupas garantindo-se melhor utilização do equipamento.

- Sistemas de Ar-Condicionado

  A utilização de um controlador *fuzzy* para sistemas de ar-condicionado permite economia energética de 10 a 30%, devido ao bom ajuste do sistema às diversas variáveis de entrada (sensores de temperatura e umidade) e aos atuadores (válvulas de água fria, abertura das tubulações, pressão das bombas, entre outros).

No âmbito acadêmico, destacam-se como aplicações da lógica *fuzzy* o controle de motores, análise de sinais, diagnósticos médicos e o controle de robôs móveis (velocidade e trajetória, principalmente) [21].

Capítulo 3

Metodologia

Neste capítulo serão apresentados uma análise dos requisitos de projeto, a interface de programação do robô Pioneer R4000, a aplicação de teleoperação via Internet e a arquitetura do sistema a ser desenvolvido.

3.1 Requisitos do Projeto

Nesta seção serão analisados os requisitos básicos a serem obtidos pelo projeto, assim como a definição específica do trabalho a ser realizado.

Segundo Beitz [22], a metodologia de projeto em engenharia deve seguir a partir da definição das necessidades e características desejáveis de um produto ou solução. Com base nesta metodologia, foram definidos alguns requisitos de projeto, de forma a guiar o desenvolvimento do trabalho. Na tabela 3.1 encontram-se os requisitos definidos para o sistema de guiagem do robô móvel Pioneer R4000. Na tabela apresentada, *N* representa um aspecto *N*ecessário à solução, enquanto *D* representa um aspecto *D*esejável à mesma.

Seguindo a metodologia de projeto definida por Beitz [22], a formulação do problema deve ser realizada através dos seguintes passos:

1. Eliminação de preferências pessoais;

   Neste caso todos os requisitos desejáveis são eliminados, restando apenas aqueles necessários à solução do problema.

2. Eliminação de requisitos sem relação com funções ou restrições;

   Após esse passo restam os requisitos de *movimentação remota*, *"feedback"* e *evitar obstáculos*.

3. Generalizar os resultados do passo anterior;

GRACO	Lista de Especificações para o Sistema de Guiagem	
Mudança	N/D	Especificação
		1. Cinemática
06/09/1999	D	Movimentos polares
	N	Baixa velocidade (segurança)
		2. Energia
	D	Monitoração nível baterias
		3. Sinais
	N	Uso de sonares e IR
	D	Nível baterias
	N	Movimentação remota
	N	Teleoperação via Internet
	N	\Feedback" com imagens e dados sensoriais
		4. Segurança
	N	Evitar obstáculos
	D	Impedir controle simultâneo
	D	Utilizar dados criptografados
		5. Ergonomia
	N	Interface \amigável"
		6. Operação
	N	Utilização simples
	D	Conversação com sintetizador de voz
	N	Autonomia no ambiente
	D	Controle da câmera de vídeo
		7. Manutenção
	N	Documentação dos programas
	D	Recarga automática das baterias

Tabela 3.1: Requisitos para o sistema.

- pós a generalização dos requisitos anteriores, obtêm-se: *envio e recepção de sinais do robô*; e *evitar obstáculos*.

4. Formulação do problema.

- formulação genérica do problema é então estabelecida: *enviar e receber sinais de um robô autônomo*.

- pós a formulação do problema, recomenda-se a ampliação da mesma, através de especificações mais amplas:

- \comunicar com um robô autônomo";
- \comunicar com um robô";
- \comunicar com um computador";
- \telecomunicaçã o";

Conclui-se, atravé s da metodologia de projeto de Beitz [22], que a soluçã o do presente projeto encontra-se principalmente em uma soluçã o de *telecomunicaçõ es*, sem no entanto ignorar ou subestimar aspectos de engenharia existentes no sistema.

3.2 A Interface de Programaçã o do Robô Nomadic XR4000

O robô NomadicXR4000 é programado atravé s de uma interface baseada na arquitetura de \software" desenvolvida pela Nomadic, denominada *Xtreme Robotics Development Environment*", ou NomadicRex [23]. NomadicRex é uma arquitetura multi-processo, fazendo uma aplicaçã o NomadicRex ser constituí da de vá rios processos funcionando simultaneamente. Há trê s tipos de processos na arquitetura:

- *Nrobot*, que é o processo servidor que se comunica com o \hardware" do robô, enviando e recebendo comandos do usuá rio;
- *Ngui*, que realiza uma interface grá fica entre o robô e o usuá rio;
- Processos criados pelo usuá rio, acessando processos do robô atravé s de *Nrobot*.

O robô é programado atravé s da utilizaçã o de uma biblioteca em linguagem C [24] fornecida pela Nomadic, denominada Nclient.h. Esta biblioteca contém a especificaçã o de comandos e estruturas de dados responsá veis pela comunicaçã o *robô-usuá rio*.

3.2.1 Comunicaçã o com o Robô

O programa desenvolvido pelo usuá rio deve inicialmente conectar-se com o robô, atravé s do comando N_InitializeClient. Esse comando abre um canal de comunicaçã o (denominado \socket") entre o programa e o robô. A sintaxe deste comando é:

```
int N_InitializeClient (const char *scheduler_hostname,
                        unsigned short scheduler_socket)
```

onde *sheduler_hostname* contém o nome do robô na rede (no robô utilizado, lower.graco.unb.br), e *scheduler_socket* indica o canal de comunicação TCP/IP (no caso, 7073).

Após aberto o canal de comunicação, o programa deve iniciar a comunicação através do comando N_ConnectRobot. Findo o trabalho com o robô, deve-se utilizar do comando N_DisconnectRobot para fechar a comunicação com o mesmo. Esses comandos possuem as seguintes sintaxes:

```
int N_ConnectRobot (long RobotID)

int N_DisconnectRobot (long RobotID)
```

onde *RobotID* é o número de identificação do robô (usualmente com o valor 1).

3.2.2 A Estrutura de Dados do Robô

A programação do robô XR4000 é realizada através da leitura e escrita de dados sobre uma estrutura de dados, denominada N_RobotState. Essa estrutura contém toda a interface de comunicação entre o "software" e o "hardware" do robô, sendo definida da seguinte forma:

```
struct N_RobotState
{
N_CONST long RobotID;
N_CONST char RobotType;
struct N_Integrator Integrator;
struct N_AxisSet AxisSet;
struct N_LiftController LiftController;
struct N_Joystick Joystick;
struct N_SonarController SonarController;
struct N_InfraredController InfraredController;
struct N_BumperController BumperController;
struct N_Compass Compass;
struct N_LasetSet LaserSet;
struct N_S550Set S550Set;
struct N_BatterySet BatterySet;
struct N_Timer Timer;
};
```

O acesso a N_RobotState é obtido através do comando N_GetRobotState, cuja sintaxe é mostrada abaixo, declarado no início do programa.

```
struct N_RobotState *N_GetRobotState (long RobotID);
```

A seguir, o usuário é capaz de ler dados da estrutura | com a família de comandos Get... | ou atualizar dados da estrutura | com a família de comandos Set....

Nos tópicos seguintes serão mostrados os comandos e estruturas responsáveis pelo controle do robô XR4000.

Movimentação do Robô

O controle cinemático do robô | ou seja, posição, velocidade e aceleração | é obtido através da estrutura N_AxisSet, que contém referência a cada um dos eixos de movimento do robô (X, Y e μ):

```
struct N_AxisSet
{
BOOL Global;
unsigned char Status;
N_CONST unsigned int AxisCount;
struct N_Axis Axis[N_MAX_AXIS_COUNT];
};
```

Na variável Global podem ser utilizados dois valores: *TRUE*, significando modo global, onde o robô possui uma referência fixa no solo para a realização dos movimentos, e *FALSE*, representando o movimento das juntas, na qual a referência é fixada no corpo do robô.

A estrutura N_Axis contém informações sobre cada um dos eixos de movimentação:

```
struct N_Axis
{
BOOL DataActive;
BOOL TimeStampActive;
BOOL Update;
unsigned long TimeStamp;
char Mode;
long DesiredPosition;
long DesiredSpeed;
```

```
long Acceleration;
long TrajectoryPosition;
long TrajectoryVelocity;
long ActualPosition;
long ActualVelocity;
BOOL InProgress;
};
```

Observa-se dos nomes das variáveis, como DesiredPosition, que a utilização desta estrutura é de fácil compreensão. O acesso a essa estrutura é realizado através dos comandos Set_Axes e Get_Axes, que possuem as seguintes sintaxes:

```
int N_SetAxes (long RobotID);

int N_GetAxes (long RobotID);
```

Posição do Robô

O robô XR4000 possui internamente um sistema de auto-localização no ambiente, baseado na geometria e movimentos realizados. Dessa forma, o sistema calcula continuamente as posições x, y e μ com base numa referência inicial no piso (essa estimativa de posição é denominada "dead-reckoning"). Embora essa informação seja afetada por problemas como o escorregamento das rodas no solo e da imperfeição do modelo cinemático, esses dados são úteis como primeira aproximação da posição do robô no ambiente.

Essas informações são registradas na estrutura N_Integrator, definida como:

```
struct N_Integrator
{
BOOL DataActive;
BOOL TimeStampActive;
long x;
long y;
long Steering;
long Rotation;
unsigned long TimeStamp;
};
```

Os comandos utilizados para obter esses dados ou atualizá-los são, respectivamente, N_GetIntegratedConfiguration e N_SetIntegratedConfiguration, cujas sintaxes são dadas a seguir:

```
int N_GetIntegratedConfiguration (long RobotID);

int N_SetIntegratedConfiguration (long RobotID);
```

Sensores

Os três tipos de sensores do XR4000 — tátil, de infravermelho e ultrasom — são controlados de forma semelhante no seu aspecto geométrico. Como pode ser visto na figura 3.1, os sensores do robô são dispostos no anel na forma de conjuntos contendo oito sensores (numerados de 0 a 7). Esses seis conjuntos, numerados de 0 a 5, são localizados nas três portas do robô, sendo três na parte superior (0, 2 e 4), e três na parte inferior (1, 3 e 5). A seguir cada uma das interfaces dos sensores é mostrada separadamente.

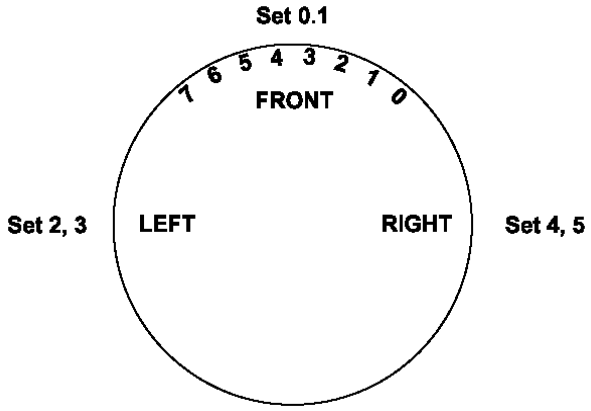

Figura 3.1: "Anel" de sensores do robô XR4000.

² Sensores Táteis

A informação sobre colisões do robô com o ambiente é registrada na estrutura N_BumperController, definida como:

```
struct N_BumperController
{
N_CONST unsigned int BumperSetCount;
struct N_BumperSet BumperSet[N_MAX_BUMPER_SET_COUNT];
};
```

A estrutura N_BumperSet contém referência a cada um dos conjuntos de sensores, sendo definida como:

```
struct N_BumperSet
{
BOOL DataActive;
BOOL TimeStampActive;
N_CONST unsigned int BumperCount;
struct N_Bumper Bumper[N_MAX_BUMPER_COUNT];
};
```

A leitura de cada um dos sensores de colisão, contida na estrutura N_Bumper, está contida na variável Reading, podendo ter os valores N_BUMPERNONE (sem colisão), N_BUMPERLOW (colisão fraca) ou N_BUMPERHIGH (para colisão forte). Essa estrutura é definida como:

```
struct N_Bumper
{
char Reading;
unsigned long TimeStamp;
};
```

Os dados da estrutura N_BumperController são atualizados através da chamada da função N_GetBumper, cuja sintaxe é dada a seguir:

```
int N_GetBumper (long RobotID);
```

2 Sensores por Infravermelho

A estrutura N_InfraredController, definida abaixo, contém a informação dos sensores infravermelhos.

```
struct N_InfraredController
{
BOOL InfraredPaused;
N_CONST unsigned int InfraredSetCount;
struct N_InfraredSet InfraredSet[N_MAX_INFRARED_SET_COUNT];
};
```

A variável InfraredPaused controla o funcionamento dos sensores, ativando-os quando possui um valor FALSE. A estrutura N_InfraredSet contém informações sobre os conjuntos de sensores, possuindo a seguinte definição:

```
struct N_InfraredSet
{
BOOL DataActive;
BOOL TimeStampActive;
N_CONST unsigned int InfraredCount;
struct N_Infrared Infrared[N_MAX_INFRARED_COUNT];
};
```

A estrutura N_Infrared possui os dados de leituras de cada um dos sensores, individualmente. Sua definição é dada por:

```
struct N_Infrared
{
long Reading;
unsigned long TimeStamp;
};
```

O valor da variável Reading varia de 0 a 255, sendo 0 equivalente a nenhuma recepção de luz refletida, e 255 como uma reflexão máxima de luz, o que indiretamente significa a proximidade do objeto no ambiente.

A atualização da estrutura N_InfraredController é realizada através da função N_GetInfrared, que possui a seguinte sintaxe:

```
int N_GetInfrared (long RobotID);
```

2 Sensores por Ultrasom

Os dados obtidos através do sistema de ultrasom são armazenados na estrutura N_SonarController, definida como:

```
struct N_SonarController
{
N_CONST unsigned int SonarSetCount;
struct N_SonarSet SonarSet[N_MAX_SONAR_SET_COUNT];
BOOL SonarPaused;
};
```

A estrutura N_SonarSet armazena as informações relativas aos conjuntos de sensores, possuindo a seguinte definição:

```
struct N_SonarSet
{
unsigned int FiringOrder[N_MAX_SONAR_COUNT + 1];
long FiringDelay;
long BlankingInterval;
BOOL DataActive;
BOOL TimeStampActive;
N_CONST unsigned int SonarCount;
struct N_Sonar Sonar[N_MAX_SONAR_COUNT];
};
```

A variável FiringOrder armazena a seqüência de disparo dos sonares, permitindo que diversas estratégias de atualização dos dados sejam obtidas. A variável FiringDelay define o intervalo de tempo, em milisegundos, entre os disparos dos sensores, regulando assim a velocidade dos mesmos para obter dados sobre a distância de objetos do ambiente.

A estrutura N_Sonar registra em sua variável Reading a distância, em milímetros, do objeto mais próximo a um determinado sensor. Essa estrutura é definida como:

```
struct N_Sonar
{
long Reading;
unsigned long TimeStamp;
};
```

A estrutura N_SonarController pode ser lida ou atualizada através de três funções principais: N_GetSonar, que recebe os dados das leituras dos sonares; N_GetSonarConfiguration, que lê os dados de configuração dos sensores; e N_SetSonarConfiguration, que atualiza a configuração dos mesmos. Essas funções possuem as seguintes sintaxes:

```
int N_GetSonar (long RobotID);

int N_GetSonarConfiguration (long RobotID);

int N_SetSonarConfiguration (long RobotID);
```

Baterias

O sistema de alimentação do robô, composto de quatro baterias, pode ser verificado pelo usuário através da estrutura N_BatterySet, definida como:

```
struct N_BatterySet
{
struct N_Battery Battery[N_MAX_BATTERY_COUNT];
BOOL DataActive;
};
```

Utilizando o comando N_GetBattery obtém-se na estrutura o valor da tensão das baterias, dadas em milivolts.

```
int N_GetBattery (long RobotID);
```

Sintetizador de Voz

O robô XR4000 possui um sintetizador de voz, permitindo que textos em inglês sejam transformados em fonemas para a saída de áudio. Para isso é utilizado o comando N_Speak, cuja sintaxe é dada a seguir:

```
int N_Speak (unsigned long RobotID, char *Text);
```

3.3 Teleoperação via Internet

A teleoperação baseada na Internet pode ser realizada através de várias metodologias. Dentre as quais destacam-se:

- **Acesso Remoto via "Telnet"**

  Uma forma de se obter o acesso a um sistema teleoperado via Internet é a conexão direta do usuário via interface "telnet", amplamente disponível em ambientes de rede. A simplicidade de operação é perdida pelo fato de se necessitar de contas de usuários nas máquinas servidoras, o que é inviável dentro de um sistema de ampla abrangência, e ser mais susceptível a falhas de segurança.

- **Programação CGI - "Common Gateway Interface"**

  A programação CGI com base em páginas HTML ("Hiper Text Markup Language") é a abordagem mais utilizada no momento para o controle de sistemas através da Internet, baseada na interface WWW ("World Wide Web") [25]. A sua desvantagem é a limitação de interatividade com o usuário, e pelo fato de sobrecarregar o servidor.

- Programação Java

  A linguagem Java, através de seus aplicativos WWW "applets", são a forma mais atual de programação para a Internet. Suas vantagens incluem a interatividade com o usuário, a fácil programação e a sua natureza voltada para a Internet. Sua desvantagem principal é a velocidade de operação, além do tempo para a inicialização dos "applets".

A seguir será explicitada em maiores detalhes a linguagem Java como interface de programação para teleoperação via Internet.

3.3.1 A Linguagem de Programação Java

Java, além de uma linguagem de programação, é também uma plataforma computacional [26]. Como linguagem de programação de alto nível, apresenta as seguintes características:

- Simples

  Ao contrário de outras linguagens de programação orientadas a objetos (como o C++), Java apresenta simplificações no armazenamento e gerenciamento de dados que permitem maior facilidade de aprendizado e desenvolvimento de programas.

- Orientada a objetos

  A programação orientada a objetos é a técnica que enfoca os dados e suas relações com os demais dados, permitindo uma programação mais "intuitiva" e com maiores possibilidades de reutilização de código.

- Distribuída

  A integração de Java com a Internet ou sistemas de comunicação via redes é facilitada pela ampla disponibilidade de bibliotecas destinadas a esse fim.

- Robusta

  Ao contrário de linguagem como o C e C++, Java não utiliza ponteiros, de forma que um programa apresenta mínimas chances de ter sua área de código corrompida por manipulações indevidas de memória.

- Segura

  Como Java é destinada à utilização em ambientes de rede, a segurança é uma grande ênfase no projeto da linguagem.

- Arquitetura neutra

  Programas Java são desenvolvidos para funcionar sobre a Máquina Virtual Java (JVM - Java Virtual Machine), de forma que o seu funcionamento independe da arquitetura do computador em que é utilizado.

- Portável

  Como Java é uma linguagem que independe da arquitetura, não possui detalhes de implementação que restringem a sua portabilidade entre diversas máquinas e sistemas operacionais.

- Interpretada

  A compilação de um programa Java gera um arquivo contendo \bytecodes", que são compreendidos e interpretados em tempo real pela JVM.

- Alta \Performance"

  Quando necessário, os códigos Java podem ser convertidos diretamente para os códigos nativos da máquina utilizada, aumentando sua eficiência computacional.

- Multiprocessada

  Programas Java podem ser desenvolvidos utilizando o conceito de processamento múltiplo, na qual o programa é sub-divido em tarefas que podem ser realizadas \simultaneamente" pelo computador.

- Dinâmica

  A utilização de bibliotecas na linguagem Java é simplificada, não necessitando de se recompilar os programas já existentes quando o código de uma biblioteca é atualizado.

Uma plataforma é o ambiente baseado em \hardware" ou \software" na qual um programa funciona. A plataforma Java, diferente das demais, é composta por:

- Máquina Virtual Java (JVM);
- Interface de Programação para Aplicações - Java API (\Java Application Programming Interface");

O Java API é composto de diversas bibliotecas ou pacotes, responsáveis pelas interfaces gráficas para usuários (GUI - \Graphical User Interface"), além de outras utilidades gerais.

3.4 Arquitetura do Sistema

Nesta seção é apresentada a arquitetura do sistema a ser desenvolvido, através do desenvolvimento do modelo de comunicação cliente-servidor.

Com base nos modelos apresentados anteriormente, o sistema de teleoperação a ser desenvolvido pode ser classificado como:

- Controle manual com auxílio computacional;
- Modelo mestre-escravo / supervisor-companheiro;
- Ambiente remoto parcialmente modelado;

O desenvolvimento da arquitetura do sistema terá como base os aspectos teóricos acima apresentados.

A figura 3.2 apresenta um esquema simplificado da arquitetura do sistema de guiagem para o robô móvel Nomad XR4000. Pode ser observado que, com base no modelo, o robô representa o servidor, enquanto que o usuário é o cliente, sendo interligados através da rede Internet.

Figura 3.2: Arquitetura simplificada.

Como foi visto anteriormente, a programação do robô é realizada na linguagem C, enquanto a programação voltada para a Internet é feita em Java. Isto não se constitui um problema, visto que ambas as linguagens são aptas a se comunicarem através de \sockets".

3.4.1 Módulos Principais do Sistema

A figura 3.2 apresentou um esquema simplificado da arquitetura do sistema. Na figura 3.3 é apresentada a estrutura detalhada do sistema, baseada no trabalho de Álvares [27], onde são mostrados os seus principais módulos.

Com base na figura 3.3 podem ser definidos os seguintes módulos:

- Robô XR4000

  Neste módulo estão os subsistemas de servidor de vídeo e servidor de controle, ambos programados na linguagem C. O servidor de vídeo é responsável pela

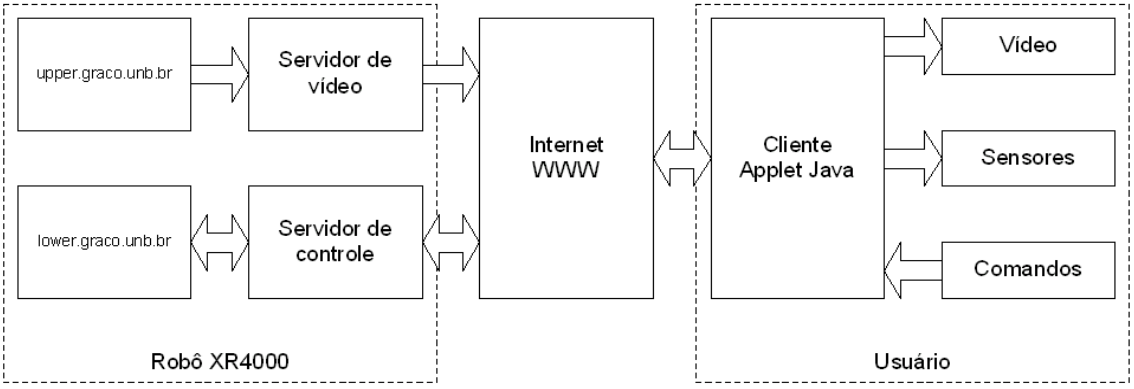

Figura 3.3: Arquitetura detalhada.

captura de vídeo e o envio dos dados para a Internet. O servidor de controle funciona de modo bidirecional, recebendo comandos através da rede e enviando dados sensoriais e posicionais do robô para a Internet. O servidor de controle implementa também o sistema de navegação e modelagem de ambiente que auxilia o operador remoto na execução das tarefas.

- Internet

  É o módulo que representa a conexão Internet desenvolvida através de canais de comunicação ("sockets") entre os módulos do *Robô XR4000* e o módulo do *Usuário*.

- Usuário

  O módulo do usuário é composto por três subsistemas programados na linguagem Java: o de vídeo, o de sensores e posição, e o de comando. O sistema de vídeo apresenta ao usuário os dados recebidos através da rede de forma gráfica. O sistema de sensores mostra dados sensoriais (sonares, IR e colisão), além de dados de "dead reckoning". O sistema de comando é responsável pelo envio de comandos do usuário para o robô, tais como a direção e velocidade do movimento desejado.

Deve-se observar que a comunicação entre os servidores e o cliente deverá ser realizada através do desenvolvimento de um protocolo de comunicação, específico para a aplicação de guiagem do robô móvel.

O desenvolvimento do projeto será realizado através da especificação e implementação dos módulos anteriormente definidos.

3.4.2 A Interface com o Usuário

A utilização do robô RR4000, além de ser realizada através da programação em C, pode ser realizada através da interface gráfica Ngui, provida pela Nomadic. Como

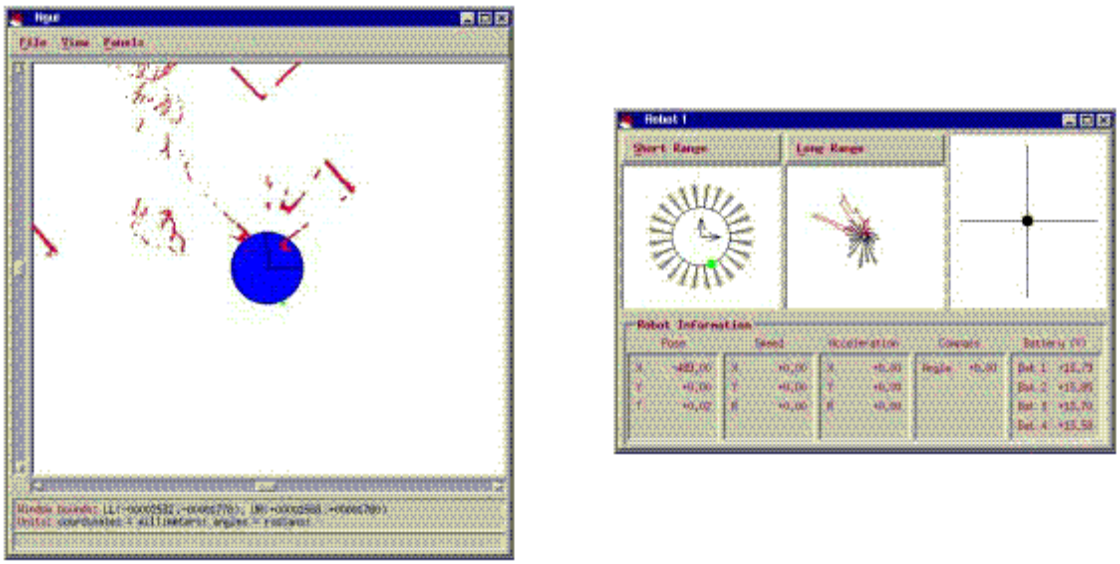

Figura 3.4: Interface com o usuário provida pela Nomadic.

pode ser visto na figura 3.4, Ngui é composto por duas telas:

- uma tela contendo a descrição do ambiente com base nos dados sensoriais (à esquerda);
- uma tela contendo dados instantâneos dos sensores IR e sonares, um "joystick" controlado pelo "mouse" e dados sobre odometria e estado das baterias do robô.

Deve-se notar que a interface Ngui não provê ao usuário informações visuais da câmera do robô, o que deve ser obtido através da utilização de um programa independente. Este programa de captura, denominado vu4color, realiza a animação dos "frames" obtidos pela placa de captura. No entanto, devido à velocidade de processamento da imagem, ocorre um grande atraso desde a captura até a visão da mesma pelo usuário, comprometendo a utilização do sistema.

A proposta de interface com o usuário (módulo *Usuário* definido anteriormente) baseia-se na união das características já apresentadas pela interface Ngui, adicionando-lhe as imagens da câmera e a utilização do sistema de navegação do sistema.

A figura 3.5 mostra uma idéia da interface a ser desenvolvida.

Podem ser vistos na figura os seguintes campos:

- Mapa

  Contém a interpretação dos dados sensoriais na forma gráfica, permitindo ao usuário uma noção do ambiente na qual o robô se encontra.

- Vídeo

  Apresenta as imagens de vídeo capturadas pela câmera do robô.

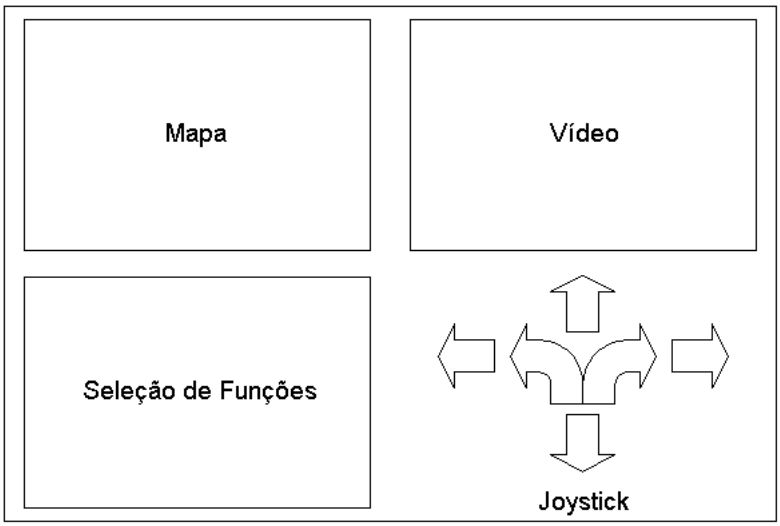

Figura 3.5: Interface com o usuário proposta para o sistema.

- Seleção de funções

  Através de um menu o usuário será capaz de escolher dados a serem visualizados, como dados sensoriais de IR ou sonares, ou dados de odometria, ou carga das baterias, além dos controles de velocidade e aceleração do robô.

- "Joystick"

  É o "joystick" virtual, composto por botões para movimentação nas direções X, Y e rotação μ.

3.5 O Sistema *Xfuzzy*

Diversos *softwares* estão disponíveis para o desenvolvimento de aplicações de lógica *fuzzy*. O ambiente Xfuzzy é um destes aplicativos e o *Xfuzzy*, desenvolvido no Instituto de Microeletrônica de Sevilha, Espanha [28].

Xfuzzy é um ambiente de projeto que auxilia na especificação, verificação e implementação final de sistemas *fuzzy*. O desenvolvimento dos sistemas é baseado na linguagem XFL, que especifica as variáveis do sistema, assim como seus universos de discurso e funções de pertinência, obtendo-se assim uma descrição detalhada do sistema *fuzzy* implementado. Com o sistema definido, é possível realizar análises e simulações sobre o projeto, permitindo dessa forma a otimização do sistema e a obtenção do código-fonte correspondente em linguagem C.

3.5.1 Definição do Sistema *Fuzzy*

O sistema *fuzzy* é definido através da linguagem XFL através de dois passos: declarações de tipo e especificação dos módulos do sistema. Esta primeira parte relaciona a definição das variáveis de entrada e saída do sistema, incluindo seus universos de discurso e suas funções de pertinência. A segunda parte define a estrutura e conteúdo da base de regras que controla o comportamento do sistema.

O modelo do sistema *fuzzy* é desenvolvido a partir dos seguintes passos:

- as variáveis de entrada são *fuzzificadas*, adotando para cada uma um valor as suas funções de pertinência;
- as regras da base de inferência são aplicadas, resultando nos valores correspondentes das funções de pertinência das variáveis de saída;
- as variáveis de saída são *desfuzzificadas*, obtendo-se assim os valores de saída do sistema.

O sistema é definido a partir de *menus* contidos no programa *Xfuzzy*. A seguir será apresentada a especificação do sistema com base na linguagem XFL.

A Especificação do Sistema via XFL

A linguagem XFL contém basicamente quatro estruturas que definem um sistema *fuzzy*: a definição de tipo, as funções de pertinência, a base de regras e as regras do sistema. A seguir são detalhados cada uma dessas estruturas.

- Definição de Tipo

  A definição de tipo de uma variável *fuzzy* é obtida através da seguinte sintaxe:

  ```
  type Identificador : TipoPredefinido [Cardinalidade] (Faixa)
  Funcao_de_Pertinencia_1
  Funcao_de_Pertinencia_2
  ...
  ```

  Identificador refere-se ao nome do tipo a ser criado, como por exemplo, *velocidade*. TipoPredefinido pode ser *integer* ou *real*, a depender do tipo de valores a serem utilizados no sistema. Cardinalidade refere-se a quantos pontos de precisão são necessários no intervalo de valores da variável. Faixa é o intervalo de valores da variável. Seja por exemplo:

```
type speed : real [1501] ( 0.000000 < 1500.000000 ) {
```

Neste caso o tipo speed foi definido como do tipo real, contendo 1501 valores possíveis entre os extremos 0 e 1500.

- Funções de Pertinência

A definição das funções de pertinência contidas em um tipo é obtida através da seguinte sintaxe:

```
Identificador  Classe  (Lista)
```

Identificador refere-se ao nome da função de pertinência, como por exemplo, *devagar*. Classe define o tipo de função de pertinência a ser utilizado. A figura 2.8 mostrou alguns dos tipos predefinidos disponíveis no sistema *Xfuzzy*. Lista define as características da função de pertinência definida. Seja por exemplo:

```
slow trapezoid (220.0, 260.0, 360.0, 470.0)
```

Neste caso foi definida a função de pertinência slow do tipo trapezoid com valores de inicio e fim de respectivamente 220:0 e 470:0, possuindo pertinência 1 entre 260:0 e 360:0.

- Base de Regras

A definição de uma base de regras possui a seguinte sintaxe geral:

```
Identificador (Lista)
```

Identificador define o nome da base de regras, e Lista define as variáveis de entrada e saída do sistema. A definição dessas variáveis possui as seguintes sintaxes:

```
Tipo ? Identificador
```

para variáveis de entrada, onde Tipo refere-se ao tipo definido anteriormente e Identificador define o nome da variável, e

```
Tipo ! Identificador
```

para variáveis de saída, com os termos Tipo e Identificador os mesmos significados que os de variáveis de entrada. A seguir tem-se um exemplo:

```
system(sonartype ? sonar, distancetype ? distance, speedtype ! speed)
 rulebase {
```

Neste caso o sistema *fuzzy* denominado system possui duas variáveis de entrada (sonar e distance) e uma variável de saída (speed), de tipos sonartype, distancetype e speedtype respectivamente.

- Regras

  Cada regra individual possui a seguinte sintaxe:

  ```
  if Antecedente -> Consequente
  ```

  Antecedente é uma expressão contendo as variáveis de entrada do sistema e suas funções de pertinência. Consequente é uma expressão que define as variáveis de saída com base em sua função de pertinência. Seja o seguinte exemplo:

  ```
  if sonar is verynear & distance is verynear -> speed is veryslow
  ```

  Neste exemplo tem-se que speed será veryslow se simultaneamente sonar for verynear e distance for verynear.

3.5.2 Análise do Sistema *Fuzzy*

Tendo sido gerada a definição do sistema *fuzzy* é possível realizar a análise do mesmo, através da visualização da superfície de saída para duas variáveis *fuzzy*, ou testando-se o algoritmo através de um simulador contido no ambiente *Xfuzzy*.

A superfície de saída é gerada através do *software* Gnuplot, podendo-se visualizar a resposta do sistema obtido para diversas entradas. Na figura 3.6 é mostrada a saída de um exemplo provido no ambiente *Xfuzzy*.

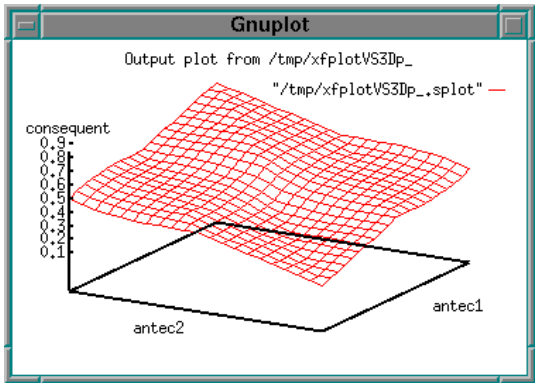

Figura 3.6: Superfície de resposta de um sistema *fuzzy*.

3.5.3 Geração de Código em Linguagem C

Uma das grandes vantagens de se utilizar um *software* de desenvolvimento de sistemas *fuzzy* é a possibilidade de se transformar de forma automática o algoritmo gerado em código-fonte. O ambiente *Xfuzzy* é capaz de gerar código em linguagem C (ANSI), de forma a ser implementado em qualquer plataforma que possua a linguagem C para programação. Assim podem ser implementados sistemas com lógica *fuzzy* tanto em microprocessadores dedicados como em computadores de grande porte.

Capítulo 4

Resultados Obtidos

Neste capítulo serão apresentados os principais resultados obtidos no desenvolvimento do projeto. Os diagramas de fluxo de dados dos sistemas são listados no apêndice B. Os códigos-fonte dos programas e sistemas apresentados são listados no apêndice E.

4.1 Servidores Localizados no Robô

Como foi visto na figura 3.3, o sistema é composto por servidores localizados no robô e por clientes localizados no *browser* do usuário. No robô foram desenvolvidos seis servidores, três localizados na máquina upper.graco.unb.br, responsável pela captura de vídeo e controle do *pan-tilt*, e três localizados na máquina lower.graco.unb.br, responsável pelos sensores e movimentação do robô.

4.1.1 Servidor de Controle do *Pan-Tilt*

Este servidor é responsável pelo controle da orientação da câmera de captura de vídeo com relação a dois eixos ortogonais: *pan* (rotação em torno do eixo *x*) e *tilt* (rotação em torno do eixo *y*). O mecanismo utilizado consiste no modelo PTU-46-70, da *Directed Perception*[29], comandado através de uma interface serial.

O servidor de controle, denominado camserver, aceita conexões da rede através do programa inetd, tendo sido configurado na porta 8086. Uma vez realizada a conexão, o servidor aguarda por comandos enviados pelo cliente remoto, executando-os caso a sintaxe e a faixa de valores das variáveis esteja correta.

O servidor camserver aceita os seguintes comandos:

- direction

Este comando define a direção de movimento do *pan-tilt*, realizando um deslocamento angular definido pelos comandos xchange e ychange. Os valores válidos para esse comando são apresentados na tabela 4.1.

upleft	up	upright
left	center	right
downleft	down	downright

Tabela 4.1: Direções de movimentação do *pan-tilt*.

A figura 4.1 apresenta um esquema do funcionamento do *pan-tilt* para o comando direction=upright.

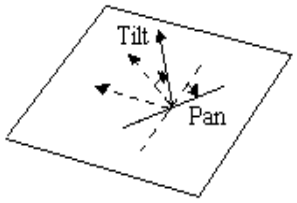

Figura 4.1: Movimentação do sistema *pan-tilt*.

- xabsolute

  Este comando define a posição absoluta para a rotação em torno do eixo *x* (*pan*). Para uma posição angular de 3000 unidades, por exemplo, utiliza-se o comando xabsolute=3000.

- yabsolute

  Este comando define a posição absoluta para a rotação em torno do eixo *y* (*tilt*). Para uma posição angular de -1000 unidades, por exemplo, utiliza-se o comando xabsolute=-1000.

- xchange

  O comando xchange define o deslocamento angular para o eixo *x* a ser realizado por comandos da família direction. Para que o comando direction=left realize um deslocamento angular de 2000, por exemplo, deve-se usar xchange=2000.

- ychange

  O comando ychange define o deslocamento angular para o eixo *y* a ser realizado por comandos da família direction. Para que o comando direction=down realize um deslocamento angular de -500, por exemplo, deve-se usar ychange=500.

- start

  Este comando ativa a movimentação do *pan-tilt*, segundo os parâmetros de movimentação definidos anteriormente. Caso algum valor definido esteja fora dos limites do atuador, o programa reajusta o valor para o limite do sistema.

- goodbye

  Este comando finaliza a conexão entre o servidor e o cliente.

4.1.2 Servidor de Imagens

O servidor de imagens desenvolvido realiza a captura das imagens obtidas através da placa de captura de vídeo *Meteor Matrox*, comprime essa imagem no formato JPEG e realiza a formatação dos dados enviados ao *browser* para que o mesmo realize a animação das imagens recebidas. A seguir cada uma dessas funções do servidor de imagens é pormenorizada.

Captura de Vídeo através da Placa *Meteor*

A placa de captura de vídeo *Matrox Meteor* possui três modos de captura de vídeo [30], apresentados resumidamente a seguir:

- Interface convencional read

  Este modo de captura é o mais fácil de se usar, embora seja o mais lento em termos de velocidade de captura. Neste modo o usuário abre o dispositivo, ajusta parâmetros relativos a imagem e através do comando read() realiza a leitura da imagem gravando-a em um *buffer* de memória.

- Captura simples por memória mapeada

  Neste modo de captura utiliza-se uma área de memória fixa como região de armazenamento dos dados lidos pela placa de captura. O usuário abre o dispositivo, define a forma e parâmetros de captura, realiza o mapeamento de memória e realiza a chamada dos dados pelo comando ioctl.

- Captura sincronizada contínua por memória mapeada

  Esta forma de captura continua de imagens é capaz de gerar imagens até uma taxa de 32 imagens/segundo. O programa do usuário é notificado através de um sinal disparado pelo *kernel* do sistema operacional, podendo então realizar a transferência dos dados para a memória mapeada e utilizar a imagem capturada.

O formato utilizado para a captura de imagens do robô móvel é a captura sincronizada, por permitir maiores taxas de imagens, o que garante um melhor *feedback* para o usuário.

Compressão das Imagens para o Formato JPEG

A compressão das imagem capturadas pela placa *Meteor* é necessária dada a relativa baixa velocidade de transmissão de dados via Internet e para permitir, ao se reduzir o tamanho da imagens, uma maior velocidade de animação.

Entre os diversos métodos de compressão de imagens destaca-se o formato JPEG [31], amplamente utilizado em programas de *WebCam* e por navegadores como o *Netscape*. O formato JPEG é um padrão de compressão com perda, ou seja, a imagem obtida não contém toda a informação original, mas apenas uma aproximação adequada para a sua visualização. Dessa forma, o formato JPEG, embora adequado para utilização em teleoperação, é inadequado para o processamento digital de imagens, no qual são obtidos parâmetros de controle com base nas imagens capturadas.

A compressão JPEG é realizada através de uma biblioteca na linguagem C, denominada jpeglib, desenvolvida pela IJG (*Independent JPEG Group*). De forma simples, a compressão de uma imagem para o formato JPEG deve seguir os seguintes passos:

- Alocação e inicialização de um objeto de compressão JPEG;
- Especificação do destino para a imagem JPEG (geralmente um arquivo ou o dispositivo padrão de saída - stdout);
- Definição de parâmetros para a compressão, incluindo o tamanho da imagem e campos de cores;
- Compressão propriamente dita, dada pelos seguintes comandos:

```
jpeg_start_compress();
while ()
  jpeg_write_scanlines();
jpeg_finish_compress();
```

- Desalocação do objeto de compressão JPEG.

A qualidade da imagem comprimida pode ser alterada com base no parâmetro *quality*, que varia de 0 a 100 (0 para imagens de baixa qualidade e 100 para imagens

de alta qualidade). A tabela 4.2 é mostrada a relação entre o valor de qualidade e o espaço consumido pela imagem comprimida. Deve-se observar que o valor da qualidade não corresponde linearmente a uma qualidade visual da imagem, como pode ser visto na figura 4.2. Optou-se utilizar um valor de qualidade de 30, como melhor compromisso qualidade-tamanho.

Quality	Tamanho (bytes)
0	2604
10	4393
20	6086
30	7427
40	8524
...	...
90	21051
100	59549

Tabela 4.2: Tamanho da imagem como função da qualidade.

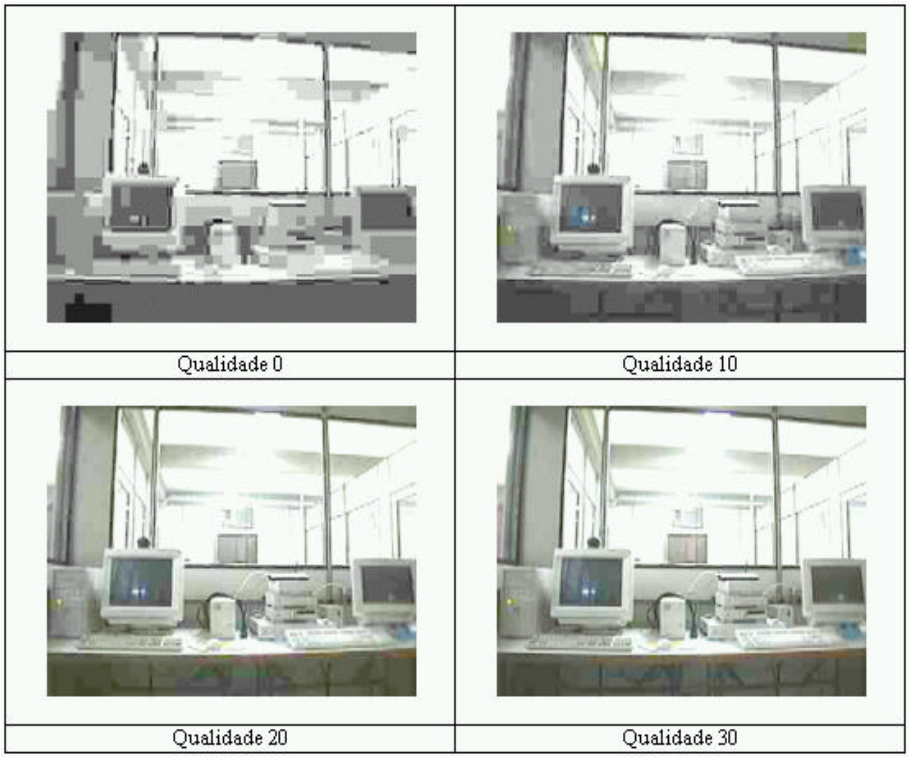

Figura 4.2: Níveis de qualidade de imagens JPEG.

Mecanismo de Animação do *Netscape*

Entre as diversas formas de se realizar a animação de imagens obtidas de uma câmera, a mais simples e que não requer a utilização de *plugins* é a utilização dos mecanismos de envio de dados do Netscape [32]. Esses dois mecanismos de envio de dados são os seguintes:

- *Server push*

  Essa forma de transmissão de dados é obtida através do envio contínuo de dados de um servidor para o *browser*, mantendo-se a conexão entre os dois aberta. Assim, quando o servidor envia um dado, o navegador exibe este dado. Após um intervalo de tempo qualquer, o servidor envia o dado seguinte, e o navegador o exibe, e assim sucessivamente.

- *Client pull*

  Nesta forma de transmissão o servidor envia dados que contêm uma mensagem ao navegador indicando \leia este dado novamente em 3 segundos". Após o tempo definido o navegador realiza a operação definida previamente, lendo o dado novamente do servidor.

Com base nestas tecnologias de transmissão de dados, é possível a animação de imagens geradas a partir de um servidor, sendo lidas continuamente pelo cliente.

O mecanismo *Server push*, em contraste com o *Client pull*, aproveita uma conexão aberta que é utilizada para o envio de diversos blocos de dados. Isso permite que o servidor tenha maior controle de quando e como os dados são enviados. Além disso, esse método é mais eficiente por manter uma conexão HTTP continuamente aberta, embora isso represente para o servidor um maior gasto de recursos computacionais. Uma outra vantagem do sistema *Server push* é que o usuário pode cancelar o envio de novos dados através do botão *Stop* do navegador.

O envio de dados através do servidor HTTP é formatado através de uma mensagem MIME no inicio do documento, relatando ao navegador a formatação do documento sendo lido. Geralmente essa mensagem MIME indica que o documento possui apenas uma parte, mas em casos especiais é possível enviar uma mensagem indicando que o mesmo possui várias partes. Esse recurso é obtido através da mensagem MIME multipart/mixed. Essa mensagem geralmente toma a seguinte forma:

```
Content-type: multipart/mixed; boundary=qualquer_texto

--qualquer_texto
```

```
Content-type: text/plain

Dados para o primeiro bloco.

--qualquer_texto
Content-type: text/plain

Dados para o segundo e último bloco.

--qualquer_texto--
```

Para o mecanismo *Server push* utiliza-se o comando MIME variante denominado multpart/x-mixed-replace. Neste caso, cada novo bloco de dados irá sobrepor os dados anteriores (*replace*). A seguir tem-se um exemplo deste mecanismo:

```
Content-type: multipart/x-mixed-replace; boundary=qualquer_texto

--qualquer_texto
Content-type: text/plain

Dados para o primeiro bloco.

--qualquer_texto
Content-type: text/plain

Dados para o segundo e último bloco.

--qualquer_texto--
```

Com base nessa tecnologia é possível realizar a animação de imagens de qualquer formato suportado pelo *browser* sem a necessidade de *plugins*, obtendo-se dessa forma maior abrangência e simplicidade do sistema de vídeo.

Resultados Obtidos com o Servidor de Imagens

O servidor de imagens desenvolvido, denominado JPGPush, é capaz de prover de forma satisfatória uma realimentação sensorial para a navegação do robô. A freqüência de imagens geradas alcança uma taxa de 3 imagens/segundo, sendo esse valor limitado pela memória de *buffer* do *browser* utilizado (Netscape). A taxa de animação

de imagens pelo Netscape depende do tamanho da imagem gerada. Optou-se nesse trabalho um valor médio de tamanho de imagem (240 x 160 pixels) e uma taxa de 3 imagens/segundo. A figura 4.3 mostra um gráfico de taxa de produção de imagens em função do tempo.

4.1.3 Servidor de Controle de Movimentos

O servidor de controle de movimentos, denominado robserver, realiza a movimentação do robô com base nos comandos dados pelo cliente e dados sensoriais do robô, utilizando-se da lógica *fuzzy* como núcleo do controle de velocidade do sistema. Utilizando os dados sensoriais (sonares e IR) e um controlador *fuzzy*, o sistema realiza o ajuste de velocidade do robô em tempo real, de forma a minimizar a possibilidade de colisões. Os dados sensoriais são utilizados de forma conservativa, ou seja, para uma dada direção de movimento é utilizado como informação o dado do sensor que possui a menor distância captada. Este servidor possui os seguintes comandos:

- direction

  Este comando define a direção de movimento do robô, seja translacional (front, back, left e right) ou rotacional (clockwise ou counterclockwise). Assim o comando direction=front define que a próxima movimentação do robô será realizada para a frente.

- distance

  Este comando define a distância linear a ser percorrida pelo robô em sua próxima movimentação. Assim, por exemplo, o comando distance=2000 indica que o

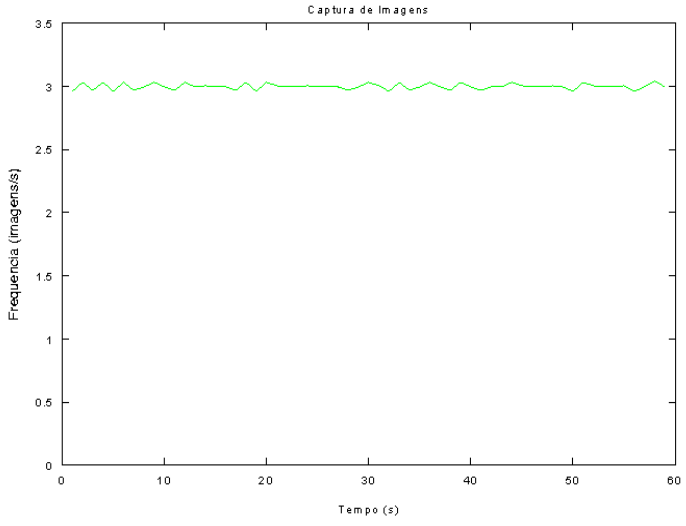

Figura 4.3: Taxa de produção de imagens no servidor.

robô percorrerá 2000 *mm* em seu movimento.

- teta

  Este comando define a rotação a ser realizada pelo robô em sua movimentação. Assim o comando teta=500 indica que o robô efetuará um giro de 500 *mrad* (miliradianos).

- omega

  Este comando define a velocidade de rotação a ser efetuada pelo robô. Como exemplo, o comando omega=200 define uma velocidade de 200 *mrad/s* (miliradianos/s).

- sensors

  Este comando liga (sensors=on) ou desliga (sensors=off) os sensores ultrasônicos. Caso os sensores estejam desligados o sistema não permite a movimentação do robô.

- start

  Este comando realiza efetivamente a movimentação do robô de acordo com os parâmetros de distância e velocidade predefinidos, desde que os sensores estejam ligados. O núcleo do sistema de movimentação utiliza-se de um algoritmo *fuzzy* para o controle da velocidade do robô como base da distância definida e dos dados sensoriais.

- goodbye

  Este comando finaliza a conexão entre o servidor e o cliente.

4.1.4 Servidor de Dados Sensoriais

O servidor de dados sensoriais, denominado senserver, é responsável pela captura de dados do robô e o seu envio para o cliente. Diversas informações podem ser obtidas através deste servidor, entre as quais: dados sensoriais dos sonares, dados dos sensores de IR, estado da carga das baterias, posição estimada (odometria) e velocidade atual dos eixos do robô. Dois comandos são disponibilizados no servidor:

- get

  O comando get, a depender de seu parâmetro, realiza a leitura de dados do robô, enviando-os de forma formatada para o cliente. Na tabela 4.3 são descritos os parâmetros utilizados no comando e suas respectivas funções. Assim, por exemplo, o comando get=IR retorna os dados dos sensores de infravermelho.

Parâmetro	Função
battery	Retorna o estado de tensão (mV) das baterias do robô.
IR	Retorna dados relativos aos sensores por IR (intensidade refletida).
odometry	Retorna a posição $(X;Y)$ e orientação μ estimada do robô.
sonar	Retorna dados de distância (mm) dos sensores ultrasônicos.
speed	Retorna as velocidades instantâneas dos três eixos de movimento do robô $(X;Y;\mu)$.

Tabela 4.3: Parâmetros utilizados no comando get.

- goodbye

  Este comando finaliza a conexão entre o servidor e o cliente.

4.1.5 Servidor de Parada

Este servidor, denominado de stopserver, foi desenvolvido com a finalidade de prover uma segurança extra para o robô. Caso o usuário verifique que o servidor de movimentos robserver está funcionando inadequadamente, é possível paralisar o movimento do robô. Este servidor possui dois comandos:

- stop

  O comando stop=on realiza a parada "instantânea" do robô através do envio de um sinal para o servidor robserver.

- goodbye

  Este comando finaliza a conexão entre o servidor e o cliente.

4.2 A Interface via Internet

A interface de controle do sistema é baseada na linguagem de programação Java, atualmente a plataforma de desenvolvimento de *software* mais adequada para aplicações na Internet. O sistema desenvolvido consiste em dois *applets*, ou seja, programas carregados pelo *browser* que funcionam na máquina remota do cliente. Esses *applets* conectam-se com os servidores localizados no robô móvel, permitindo assim o envio e recebimento de dados entre o usuário e o robô. A seguir são pormenorizados os *applets* desenvolvidos.

4.2.1 *Applet* de Controle do *Pan-Tilt*

O *applet* de controle do *pan-tilt* é responsável pelo posicionamento da câmera de vídeo do robô, possibilitando uma visualização do ambiente no qual o robô se encontra, seja na direção de seu movimento como em outras direções. Inicialmente a câmera é posicionada horizontalmente e direcionada para a posição frontal do robô. O *applet* de controle (figura 4.4) é composto pelos seguintes subsistemas:

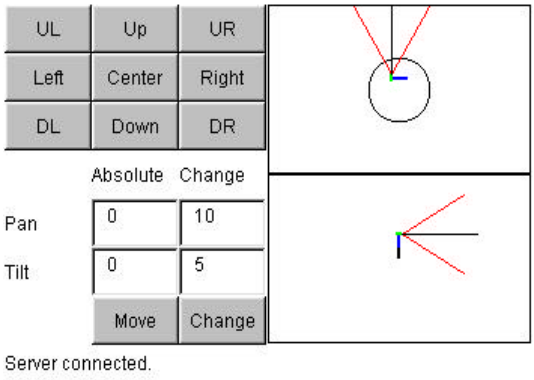

Figura 4.4: *Applet* de controle do *pan-tilt*.

- *Joystick*

  O *joystick* consiste num conjunto de nove botões, responsáveis pelo posicionamento relativo da câmera. O botão central, denominado center, realiza a centralização da câmera, reposicionando-a novamente para a posição horizontal e direcionada para a parte frontal do robô.

- Controles

  A seção de controle realiza a configuração do deslocamento angular do *pan-tilt* e da posição absoluta do mesmo.

- Gráficos de Orientação

  Os gráficos de orientação apresentam de forma simples a orientação tomada pela câmera com relação ao robô. Na parte superior é mostrada uma vista de cima do sistema, permitindo a visualização da rotação *pan*. Na parte inferior apresenta-se uma vista lateral do sistema, permitindo a visualização da rotação *tilt*. O formalismo matemático utilizado para o desenvolvimento desses gráficos é apresentado no apêndice A.

4.2.2 *Applet* de Controle de Movimentos

O *applet* de controle de movimentos é responsável pela interface entre o usuário e os servidores robserver, sensenver e stopserver localizados no robô. Com base nesta interface, o cliente é capaz de posicionar o robô no plano, realizar ajustes de deslocamento linear e angular assim como receber dados sensoriais e o estado geral das baterias do robô. O *applet* de controle (figura 4.5) é composto pelos seguintes subsistemas:

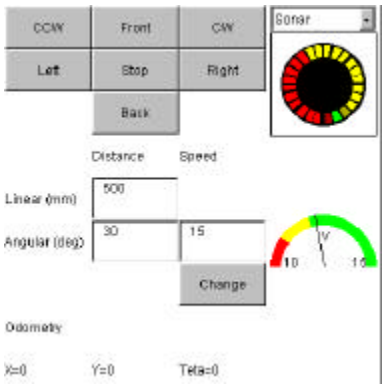

Figura 4.5: *Applet* de controle de movimentos.

- "Joystick"

  O *joystick* consiste num conjunto de sete botões, responsáveis pelo posicionamento relativo do robô (translação $X;Y$ e rotação μ). O botão central, denominado stop, realiza a parada "instantânea" do robô, conectando-se com o servidor stopserver.

- Controles

  A seção de controle realiza a configuração do deslocamento linear e angular do robô, além da velocidade de rotação do robô.

- Dados Sensoriais

  O *applet* apresenta um modelo do robô (vista superior) com um código de cores dos sensores de IR ou sonares. O código de cores segue o padrão do sistema de trânsito, ou seja, vermelho indica proximidade, amarelo média distância e verde longa distância. Nesta parte também apresenta-se o dado de tensão das baterias do robô, indicando ao usuário a autonomia do robô.

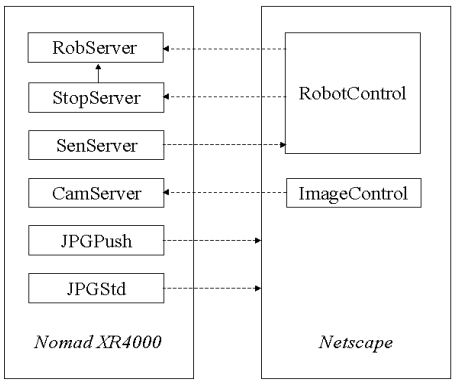

Figura 4.6: Diagrama esquemático do sistema.

4.2.3 Imagens da Câmera do Robô

A recepção das imagens animadas pelo servidor é responsabilidade do navegador *Netscape*. Durante o funcionamento do sistema o usuário inicialmente visualiza a animação gerada pelo sistema, sendo capaz de movimentar e explorar o ambiente remoto através dessas imagens. Caso seja necessário, o usuário pode realizar um "zoom" da imagem pressionando o *mouse* sobre a animação, sendo então mostrada uma imagem estática ampliada do servidor de imagens do robô (servidor JPGStd).

4.2.4 Esquema Geral do Sistema

A Figura 4.6 apresenta um resumo dos servidores e clientes desenvolvidos no projeto, ressaltando o fluxo de dados entre os seus diversos componentes e entre o robô XR4000 e o *browser* Netscape.

4.2.5 Página HTML do Sistema de Teleoperação

Os *applets* desenvolvidos e as imagens geradas foram reunidos numa página HTML, responsável pela interação com o usuário. A Figura 4.7 mostra o resultado final do sistema de teleoperação.

4.3 Controlador de Velocidade por Lógica *Fuzzy*

Aplicações de teleoperação necessitam de uma certa autonomia para se garantir a segurança do sistema, visto que o tempo entre o envio de um comando e o recebimento das imagens ou dados sensoriais pode ser da ordem de segundos. Uma aplicação simples de controle de velocidade é necessário, de forma a se garantir uma velocidade de movimentação do robô adequada ao ambiente no qual ele se encontra. Para um

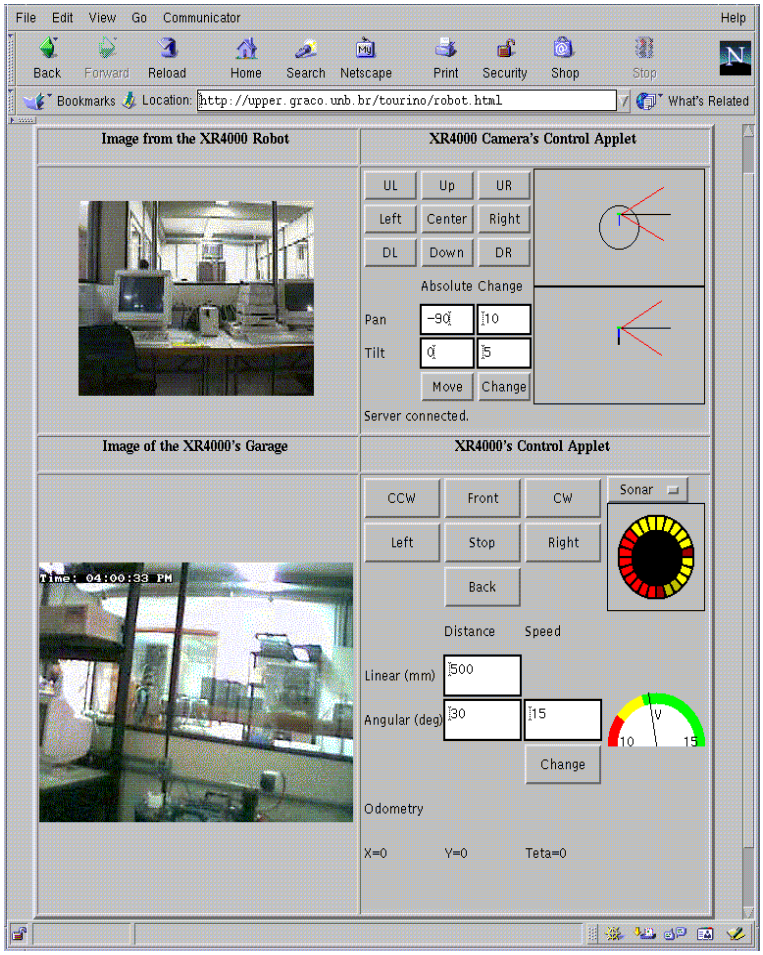

Figura 4.7: Página HTML de teleoperação do sistema.

ambiente livre de obstáculos é possível o aumento da velocidade do sistema, enquanto que para um ambiente congestionado deve-se utilizar menores velocidades para que se evite a colisão ou se minimize os efeitos de uma colisão acidental.

Neste trabalho utilizou-se da lógica *fuzzy* como forma de controle de velocidade do robô móvel, baseando a velocidade de saída na distância a ser percorrida (definida pelo usuário) e na distância dos obstáculos presentes no ambiente (aleatórios e transientes). Matematicamente o controlador *fuzzy* realiza o seguinte mapeamento de espaços:

$$v = f(d_d; d_o)$$

onde:

- v velocidade do robô
- d_d distância definida pelo usuário
- d_o distância do obstáculo mais próximo na direção do movimento.

4.3.1 Definição dos Conjuntos *Fuzzy*

O sistema *fuzzy* desenvolvido utiliza duas variáveis de entrada (sonar e distance) e uma variável de saída (speed). A seguir são explicitadas as definições utilizadas para as variáveis:

- distance

  Foram utilizadas funções de pertinência de forma triangular, discretizando os valores dados pelo usuário ($0mm < d_d \leq 5000mm$) em cinco conjuntos *fuzzy*. Utilizou-se superposição nestes conjuntos (figura 4.8), de forma a suavizar a *fuzzy-ficação* dos dados.

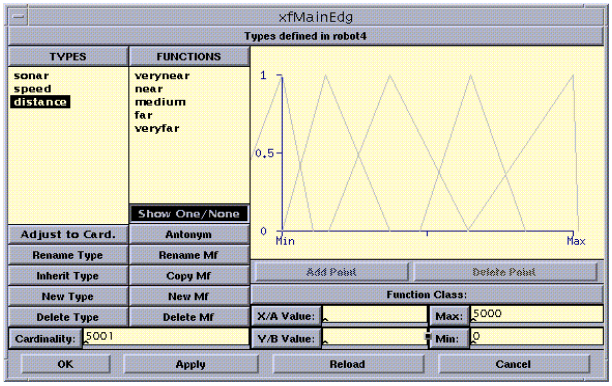

Figura 4.8: Funções de pertinência para a variável distance.

- sonar

  Foram também utilizadas funções de pertinência de forma triangular, discretizando os valores lidos pelo sensor ultrasônico ($0mm < d_d \leq 8000mm$) em cinco conjuntos *fuzzy*. Utilizou-se superposição nestes conjuntos (figura 4.9), de forma a suavizar a *fuzzy-ficação* dos dados.

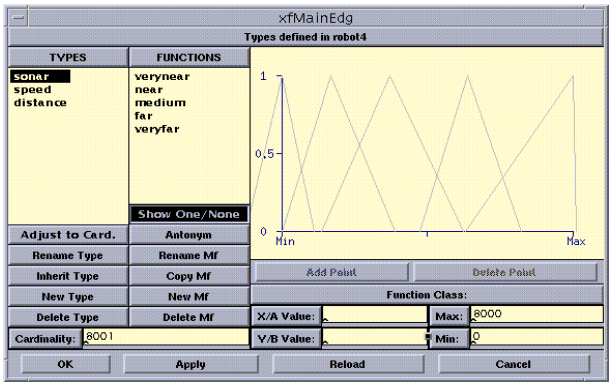

Figura 4.9: Funções de pertinência para a variável sonar.

² speed

Utilizou-se como função de pertinência de saída a forma trapezoidal (figura 4.10), com o objetivo de criar cinco níveis de velocidade preferenciais (através de *defuzzificação* pelo método do centróide). A velocidade de saída encontra-se na faixa $0\,mm/s \leq v \leq 1500\,mm/s$.

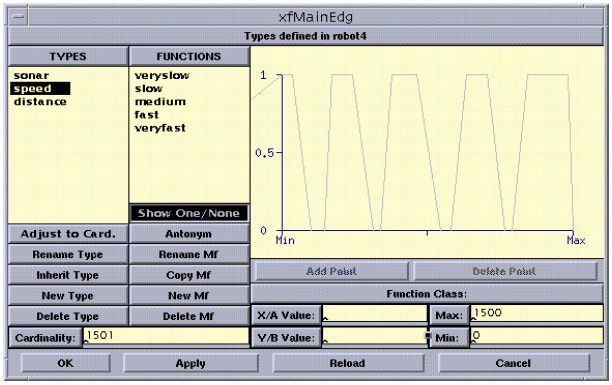

Figura 4.10: Funções de pertinência para a variável speed.

4.3.2 Definição das Regras

A definição das regras do controlador *fuzzy* baseia-se no fato de que o robô deve se mover de forma mais lenta quando próximo de obstáculos ou próximo de seu destino. Com base nesta idéia, a matriz contida na tabela 4.4 apresenta as regras utilizadas numa forma condensada. A figura 4.11 apresenta as regras no sistema *Xfuzzy*.

Sonar \ Distance	MN	N	M	F	MF
MN	VS	VS	VS	VS	VS
N	VS	S	S	S	S
M	VS	S	M	M	M
F	VS	S	M	F	F
MF	VS	S	M	F	MF

Sonar, Distance		Speed	
VN	Muito perto	VS	Muito devagar
N	Perto	S	Devagar
M	Médio	M	Médio
F	Longe	F	Rápido
VF	Muito longe	VF	Muito rápido

Tabela 4.4: Regras utilizadas no controlador.

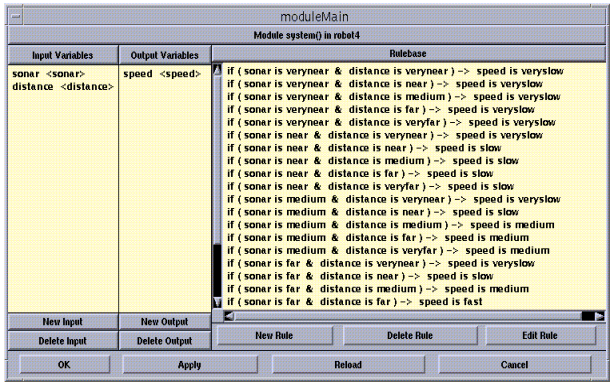

Figura 4.11: Regras definidas no controlador.

4.3.3 Superfície de Resposta do Controlador

Com base nas definições de conjuntos *fuzzy*, regras e método de *desfuzzificação* (método do centróide), foi obtida a superfície de resposta do controlador (figura 4.12).

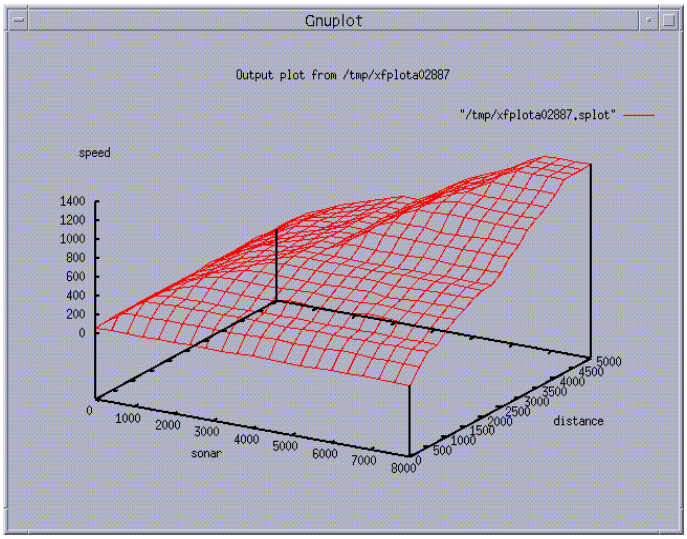

Figura 4.12: Superfície de resposta do controlador *fuzzy*.

Observa-se no gráfico apresentado uma relativa suavidade na resposta do controlador, não apresentando grandes variações de velocidade para pequenas variações das variáveis de entrada. Verifica-se ainda que a resposta do controlador segue as regras definidas, possuindo baixas velocidades para condições de distância ou leituras de sonar próximas.

4.3.4 Resultados Obtidos com o Controlador

O sistema de movimentação do robô possui sensores que disponibilizam ao usuário o estado atual de seus atuadores, ou seja, posição, velocidade e aceleração. Para

a análise do controlador *fuzzy* desenvolvido utilizou-se dos dados de velocidade do sistema, assim como de dados do relógio interno do robô.

Através da metodologia de projeto fatorial a dois níveis (três variáveis independentes e duas repetições) foram estudadas, de forma qualitativa, a influência das variáveis distância percorrida, aceleração e dos dados sensoriais sobre a forma das curvas de velocidade do robô no tempo. Essa análise visou um maior conhecimento das capacidades e limitações do controlador desenvolvido. A tabela 4.5 apresenta um resumo dos resultados obtidos nos experimentos. Gráficos obtidos no experimento são apresentados no apêndice C.

Experimento	Corridas	Distância	Sensor	Aceleração	Variação?	Oscilação?
1	1,13	-	-	-	Não	Pequena
2	2,6	-	-	+	Não	Grande
3	4,12	-	+	-	Sim	Grande
4	10,11	-	+	+	Sim	Grande
5	5,8	+	-	-	Não	Grande
6	15,16	+	-	+	Não	Grande
7	7,9	+	+	-	Sim	Grande
8	3,14	+	+	+	Sim	Grande

Tabela 4.5: Resultados do projeto fatorial realizado.

Variação refere-se à observação se as duas curvas obtidas para cada configuração apresentam grande dispersão, enquanto Oscilação refere-se à observação se cada uma das curvas possui oscilação pronunciada de velocidade no tempo.

Efeitos da Distância Percorrida

A distância percorrida, variada entre $2m$ (-) e $4m$ (+), não apresenta influência nos parâmetros analisados, embora o seu aumento permita, devido ao maior tempo de viagem, uma maior permanência do sistema na velocidade máxima permitida pelo controlador.

Efeitos da Aceleração dos Atuadores

A aceleração do robô, variada entre $200mm/s^2$ (-) e $800mm/s^2$ (+) não apresentou influência perceptível sobre os parâmetros analisados, atuando somente no tempo total de viagem do robô.

feitos dos Dados Sensoriais

Os sensores ultrasônicos são grandemente influenciados por efeitos de reflexão das ondas no ambiente. Como mostra a figura 4.13, o sistema sensorial apresenta um grande ruído, demonstrado por grandes variações dos dados no tempo. Essa variação pode representar uma influência na resposta de velocidade do controlador *fuzzy*.

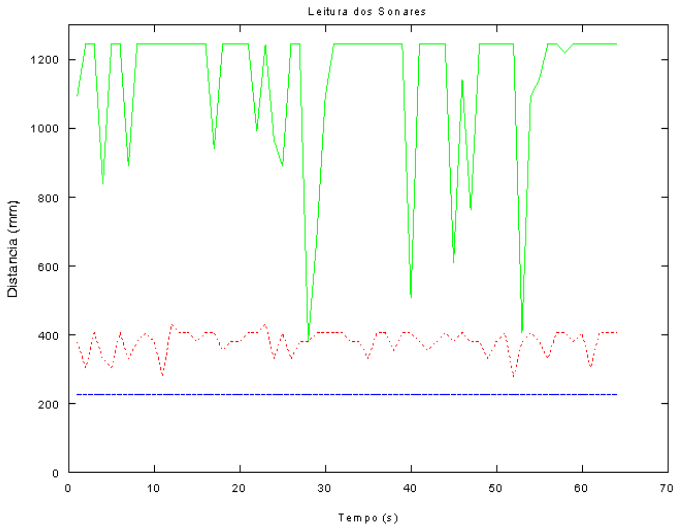

Figura 4.13: Leitura de sonares em função do tempo.

O efeito dos dados sensoriais foi analisado desligando-se sua ação sobre o controlador (-) ou ligando-os ao controlador (+). Verificou-se claramente, como previsto anteriormente, uma grande influência sobre a variação de velocidade no sistema de controle *fuzzy*, o que indica que para uma melhor utilização desses dados sensoriais deva-se utilizar um filtro nos dados sensoriais, de forma a estabilizar a saída do controlador.

4.4 Análise dos Resultados do Projeto

Nesta seção é apresentada a simulação da utilização do sistema de teleoperação do robô XR4000, realizada no laboratório do GRUCO, e propostas de melhorias a serem realizadas no sistema.

4.4.1 Simulação de Inspeção de Tubulações Soldadas

Foi realizada uma simulação da aplicação prática do sistema de teleoperação desenvolvido através da navegação do robô no ambiente do laboratório do GRUCO, a aproximação do robô ao objetivo e a visualização de uma estrutura soldada.

Figura 4.14: Movimento de aproximação do robô.

O objetivo da simulação foi, a partir da origem do "estacionamento" do robô, chegar à estrutura soldada e capturar uma imagem da solda, de forma a permitir a análise remota da mesma. Foram capturadas quatro imagens durante o processo de navegação, apresentando o processo de aproximação do robô à célula de soldagem de quadros de bicicletas presente no laboratório. Estando o robô adequadamente posicionado, a câmera do mesmo foi ajustada de forma a obter uma boa orientação e então a imagem ampliada do quadro soldado foi gravada. As figuras 4.14 e 4.15 apresentam o desenvolvimento da simulação.

A pesar da simplicidade da simulação apresentada, esta mostra as principais capacidades do sistema desenvolvido, ou seja, a navegação através de imagens do ambiente remoto e o controle *fuzzy* de velocidade em função dos obstáculos no ambiente.

4.4.2 Discussão dos Resultados

A tecnologia de teleoperação via Internet encontra-se em seu estágio inicial de desenvolvimento, tendo sua aplicabilidade ainda reduzida devido à baixa velocidade de transmissão de dados da atualidade. N este trabalho foi apresentado o desenvolvimento de um sistema de guiagem do robô N R4000 via Internet para a inspeção de tubulações industriais soldadas. Este sistema de teleoperação utiliza como ferramenta de comunicação de dados a linguagem de programação Java, atualmente a plataforma

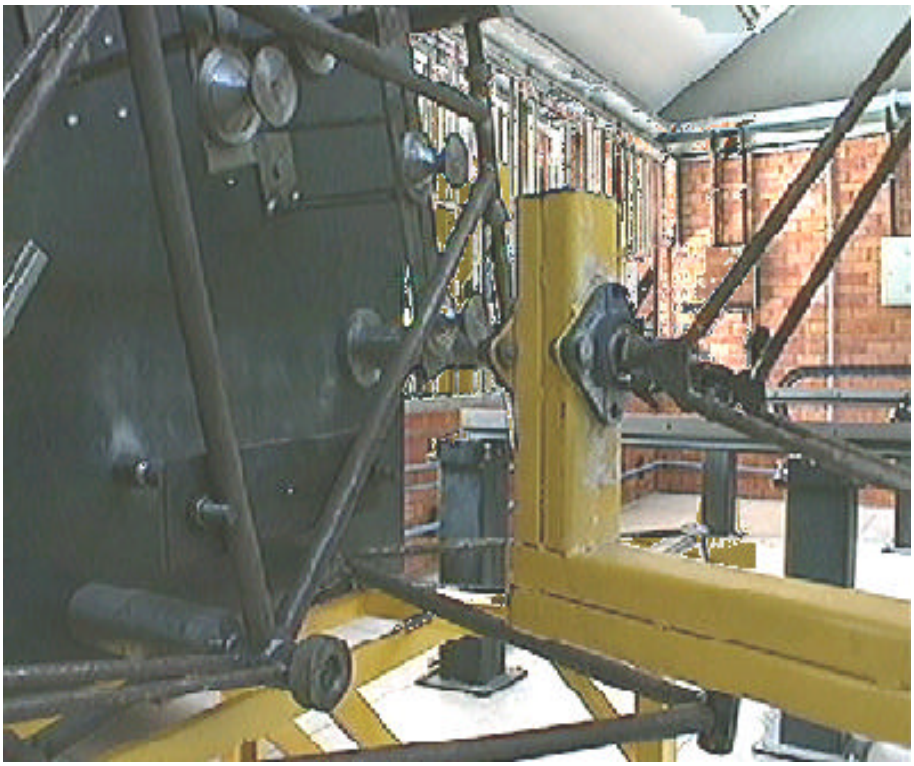

Figura 4.15: Imagem ampliada das peças soldadas.

mais adequada ao desenvolvimento de aplicações na Internet. O sistema apresenta ao usuário imagens obtidas do ambiente remoto assim como dados sensoriais do robô, e permite ao mesmo o controle de movimentos do robô e da câmera embarcada no sistema. Devido às restrições de velocidade de comunicação as imagens capturadas são comprimidas no formato JPEG, permitindo assim um menor consumo de banda de comunicação e melhorando a apresentação das imagens ao usuário. A necessidade de um sistema seguro de controle do robô e devido aos atrasos inerentes à comunicação via Internet levaram à utilização de um sistema de controle de velocidade do robô baseado na lógica *fuzzy*. Este controlador é responsável pela segurança do sistema, através do monitoramento dos dados sensoriais do ambiente e do controle da velocidade do robô. Foi veri cado, através de experimentos, que o controlador *fuzzy* é sensível aos dados obtidos através dos sensores ultrasônicos do robô, o que leva à necessidade de se desenvolver uma espécie de ltro de dados sensoriais para obter uma maior estabilidade e con abilidade no sistema de controle de velocidade. A utilização do sistema teleoperado em ambientes conhecidos também requer o posterior desenvolvimento de um sistema autônomo de navegação, o que simpli caria a teleoperação do robô móvel A R4000 em ambientes estruturados.

4.4.3 Propostas de Melhorias no Sistema

Com base nas análises apresentadas anteriormente, são propostas as seguintes evoluções ao projeto:

- Desenvolvimento de um mapa estático do ambiente, permitindo ao usuário o conhecimento de obstáculos fixos no ambiente de trabalho;
- Desenvolvimento de um sistema autônomo de navegação, de forma a auxiliar o usuário no controle do robô em um ambiente conhecido;
- Utilização de um sistema de filtro nos dados sensoriais (usando redes neurais ou lógica *fuzzy*), de forma a minimizar a variação de velocidade do controlador *fuzzy* decorrentes de dados sensoriais transientes;
- Permitir o envio de imagem sem compressão e maior qualidade para posterior processamento das mesma para fim de inspeção de juntas soldadas.

Capítulo 5

Conclusão

O desenvolvimento realizado no projeto e a sua análise permitiram as seguintes considerações finais sobre o mesmo:

- Restrições da Teleoperação via Internet

  A baixa velocidade da Internet na atualidade requer um sistema robusto e dotado de uma relativa inteligência para assegurar a segurança do sistema.

- Controle *Fuzzy*

  A restrição de velocidade de comunicação com o cliente necessitou de um sistema de inteligência artificial baseado em lógica *fuzzy* para garantir uma melhor interação robô-ambiente, garantindo-se assim a integridade física do sistema.

- Interface Java e Imagens do Ambiente

  A interface Java desenvolvida aliada às imagens capturadas e animadas mostraram-se adequados para a teleoperação do robô, mesmo considerando-se as restrições de comunicação.

- Navegação Autônoma

  A evolução do projeto baseia-se no desenvolvimento de um sistema autônomo de navegação, permitindo-se assim maior independência do sistema e reduzindo a ação do usuário a somente tarefas críticas.

Referências Bibliográficas

[1] PARMAR, R. S. *Welding Processes and Technology.* Khanna Publishers. Delhi. 1995.

[2] BARE, J. S. *An Intelligent Inspection and Survey Robot.* University of South Carolina. Columbia. USA. 1996.

[3] WEMZOW, D. BÄUMEIER, A. DÄRER, H. *Remote Control of Mobile Robot via Internet.* Manchester. 1996.

[4] ZAKI, S. MILGRAM, P. *A telerobotic virtual control system.* Proc. SPIE. Boston. 1991.

[5] ZAKI, S. MILGRAM, P. *Human Robot Synergism and Virtual Telerobotic Control.* Proc. IFAC. Canada. 1992.

[6] Nomadic Technologies. *XR4000 Mobile Robot.* Disponível na Internet via WWW. URL: http://www.robots.com/products.htm. 1999.

[7] Nomadic Technologies. *Nomad XR4000 Hardware Manual Release 1.0.* California. USA. 1999.

[8] MCKERROW, P. J. *Introduction to Robotics.* Addison-Wesley. 1991.

[9] AMAMI, M. CHOI, B. SCHULTZ, A. ADAMS, W. *Integrating Exploration and Localization for Mobile Robots.* Disponível na Internet via WWW. URL: http://www.aic.nrl.navy.mil/»yamauchi/index.html. 1998.

[10] SIEGWART, R. WANNAZ, C. GARCIA, P. *Guiding Mobile Robots through the Web.* IROS'98. Canadá. 1998.

[11] SAUCY, P. MONDADA, F. *KhepOnTheWeb: One Year of Access to a Mobile Robot on the Internet.* Microprocessor and Interface Lab. Switzerland. 1998.

[12] SCHULZ, D. BURGARD, W. CREMERS, A. *Predictive Simulation of Autonomous Robots for Tele-Operation Systems Using the World Wide Web.* University of Bonn. Germany. 1998.

[13] SIMMONS, R. *Xavier: An Autonomous Mobile Robot on the Web.* Carnegie Mellon University. USA. 1998.

[14] SOMMERVILLE, I. *Software Engineering.* Addison-Wesley. USA. 1996.

[15] BRETT, M.F., HALL, E., HARPER, E. *Inside TCP/IP.* New Riders Publishing. Indianapolis. USA. 1994.

[16] METCALFE, M. GIERTH, A. et al. *Programming UNIX Sockets in C - Frequently Asked Questions.* Disponível na Internet via WWW. URL: http://www.ibrado.com/sock-faq/ 1998.

[17] HALL, B. B. *Beej's guite to network programming using internet sockets.* Disponível na Internet via WWW. URL: http://www.eest.esuchico.edu/~beej/guide/net 1999.

[18] STEVENS, W. R. *Advanced Programming in the UNIX Environment.* Addison Wesley Longman. USA. 1993.

[19] Internet.com Corp. *Fuzzy Logic* Disponível na Internet via WWW. URL: http://webopedia.internet.com/TERM/f/fuzzy_logic.html. 2000.

[20] SILVA, I. S., SIMÕES, M.G. *Controle e Modelagem Fuzzy* Edgard Blücher. Sao Paulo. 1999.

[21] SAFFIOTTI, A. *Fuzzy Logic in Autonomous Robot Navigation, a case study* Disponível na Internet via WWW. URL: http://iridia.ulb.ac.be/FLAR/FAC/home.html 1997.

[22] PAHL, G. BEITZ, W. *Engineering Design: A Systematic Approach.* Springer Verlag. 1988.

[23] Nomadic Technologies. *Nomad XRDV Software Manual Release 1.0.* California. USA. 1999.

[24] BARRET, Martin L. *C and UNIX: Tools for Software Design.* J Wiley. New York. 1996.

[25] NCSA. *The Common Gateway Interface.* Disponível na Internet via e-mail: cgi@ncsa.uiuc.edu.

[26] Sun Microsystems. *The Java Tutorial.* Disponível na Internet via WWW. URL: http://java.sun.com/docs/books/tutorial/index.html. 1999.

[27] LIMA, J. ROMARIZ, L. S. *Desenvolvimento de um manipulador com dois graus de liberdade controlado remotamente via Internet.* CEM-IE98. Fortaleza. 1998.

[28] FUZZY *Xfuzzy* Disponível na Internet via WWW. URL: http://www.imse.cnm.es/fuzzy. 2000.

[29] Directed Perception Inc., *an Tilt Manufacturer HOME PAGE.* Disponível na Internet via WWW. URL: http://www.dperception.com/ 2000.

[30] SITTO, M. *Matrox Meteor FrameGrabber Driver for Linux v1.5.4.* Disponivel na Internet via FTP. URL: ftp://ftp.rwii.com/pub/linux/system/Meteor/meteorman.html 1999.

[31] The Independent JPEG Group. *JPEG Software* Disponível na Internet via FTP. URL: ftp://ftp.uu.net/graphics/jpeg/ 1998.

[32] Netscape. *An Exploration of Dynamic Documents* Disponível na Internet via WWW. URL: http://www1.netscape.com/assist/net_sites/pushpull.html 1999.

Apêndice A

Terminologia Básica de Robótica Móvel

ambiente remoto é o ambiente no qual está localizada a máquina teleoperada.

autonomia é a capacidade de um sistema funcionar adequadamente em seu ambiente sem a necessidade de intervenção humana.

compressão de imagens é o método utilizado para reduzir o tamanho físico de uma imagem, de forma a otimizar a sua utilização pelo usuário.

inteligência artificial são métodos computacionais que visam desenvolver um nível de raciocínio e inferência em máquinas.

lógica fuzzy é a lógica não aristotélica em que um fato pode ser verdadeiro, falso ou meio verdadeiro, utilizada para caracterizar computacionalmente a imprecisão típica dos seres humanos.

mapeamento é a organização dos dados sensoriais em uma forma apropriada para serem utilizados pelo robô móvel para a navegação autônoma no ambiente.

navegação é a movimentação do robô no ambiente, com base em dados sensoriais, odométricos e de mapas armazenados.

odometria é a estimativa com base em modelos cinemáticos da posição e orientação do robô em seu ambiente.

placa de captura de imagens é o equipamento que converte a imagem recebida por uma câmera de vídeo em informações adequadas ao uso em computadores.

planejamento de tarefa é o método pelo qual um robô realiza a escolha da tarefa a ser realizada com base nas ordens de um operador humano.

planejamento de trajetória é o método no qual um robô escolhe a trajetória ótima com base na tarefa a ser realizada, definida ou por um operador ou por um sistema de planejamento de tarefas.

processamento de imagens é a transformação e obtenção de informações de uma imagem para a utilização em processos de medição, posicionamento ou orientação de dispositivos.

processamento de sinais é a transformação de sinais elétricos ou de dados de forma a otimizar ou obter informações para posterior utilização por outros sistemas.

robô são sistemas capazes de realizar tarefas repetitivas de forma ótima, flexível e programável.

robô móvel são robôs que permitem movimentação própria sobre o solo, modificando sua posição em relação a um referencial fixo.

sensores são dispositivos que realizam a transformação de formas de energia com o objetivo de obter informações úteis a um sistema produtivo ou robótico.

sensores por infravermelho são sensores que utilizam luz infravermelha como sinal emitido e lido para medição de distâncias ou temperatura.

sensores táteis são sensores ativados através do toque ou colisão, sendo utilizados em robôs como sistema de segurança do sistema.

sensores ultrasônicos são sensores que utilizam pulsos de som de alta freqüência para medir, através do tempo de viagem da onda, a distância do sensor até um objeto próximo.

teleoperação é a metodologia utilizada para controlar dispositivos à distância, usualmente recebendo informações do ambiente remoto.

teleoperador é o equipamento controlado à distância através de um sistema de teleoperação.

telepresença é uma forma de teleoperação na qual o usuário, através de dispositivos especiais, tem a sensação de estar atuando diretamente sobre o ambiente remoto.

Apêndice B

Diagramas de Fluxo de Dados do Sistema

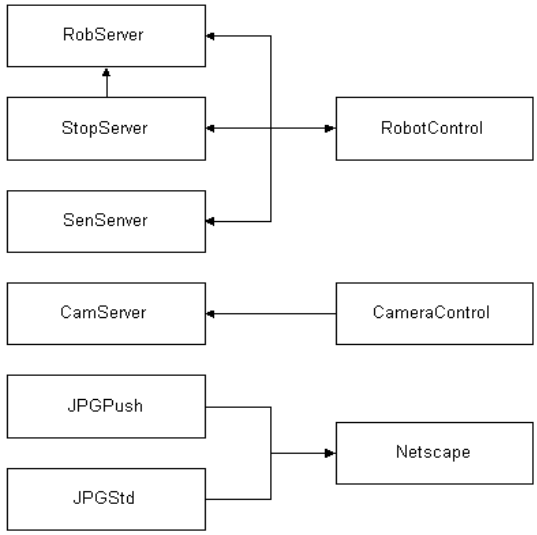

Figura B.1: Esquema geral do sistema.

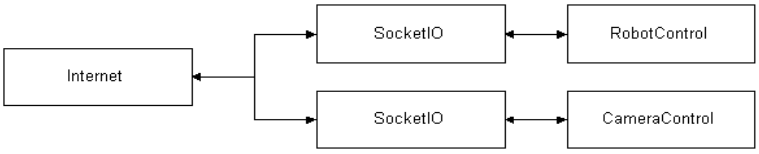

Figura B.2: Comunicação dos *applets*.

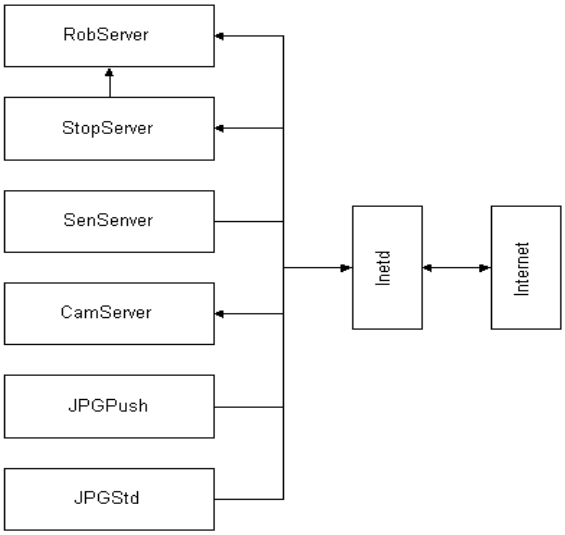

Figura B.3: Comunicação dos servidores.

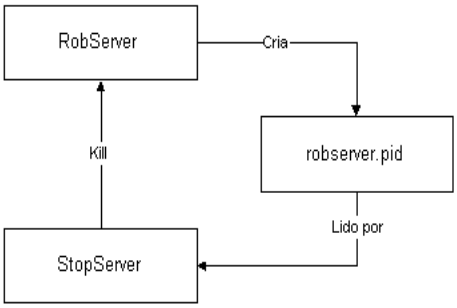

Figura B.4: Interação entre os servidores Robserver e StopServer.

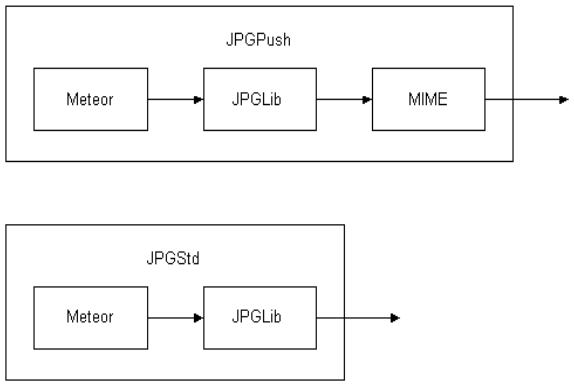

Figura B.5: Funcionamento dos servidores de vídeo.

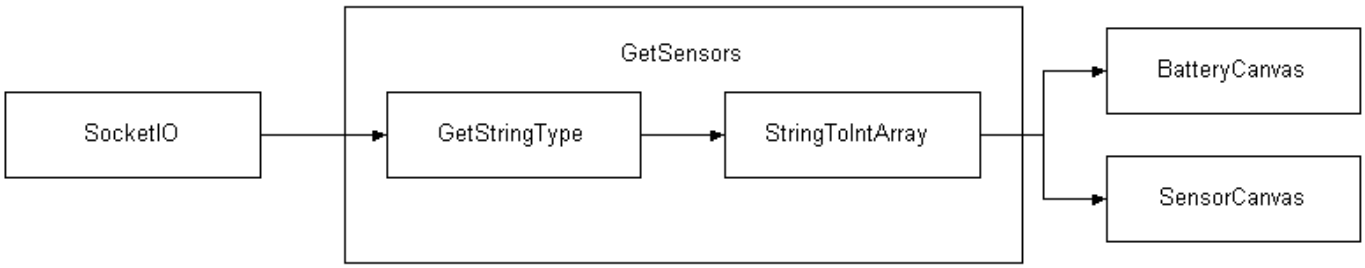

Figura B.6: Leitura dos dados sensoriais.

Apêndice C

Dados Utilizados no Projeto Fatorial

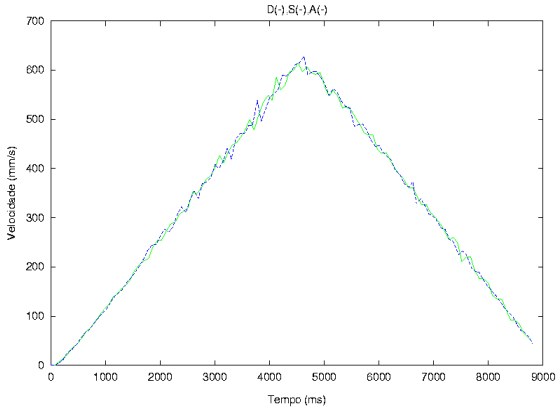

Figura C.1: Dados obtidos para o primeiro experimento.

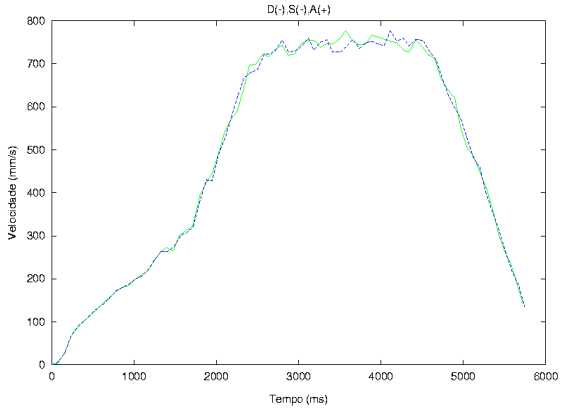

Figura C.2: Dados obtidos durante o segundo experimento.

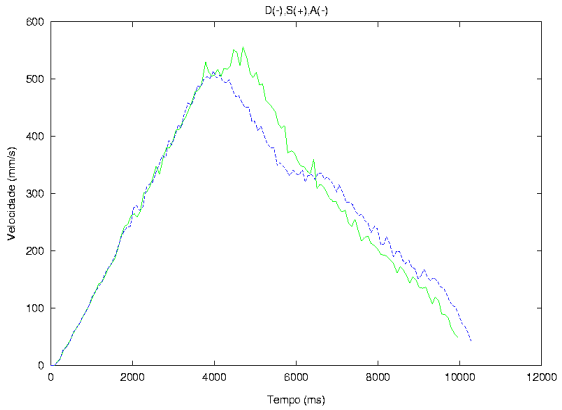

Figura C.3: Dados obtidos durante o terceiro experimento.

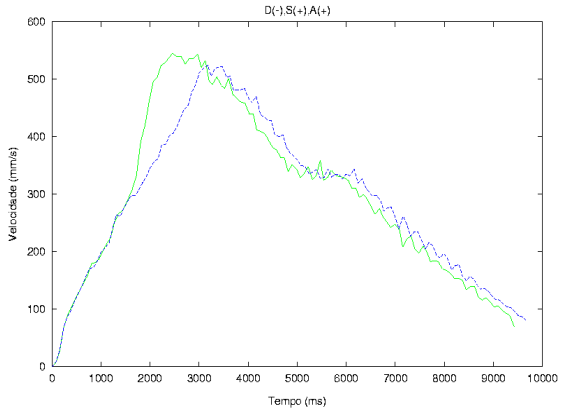

Figura C.4: Dados obtidos durante o quarto experimento.

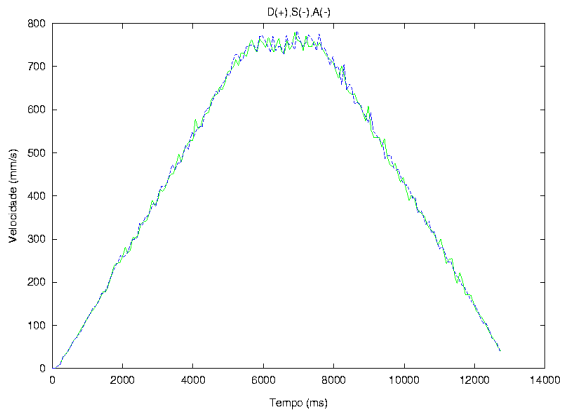

Figura C.5: Dados obtidos durante o quinto experimento.

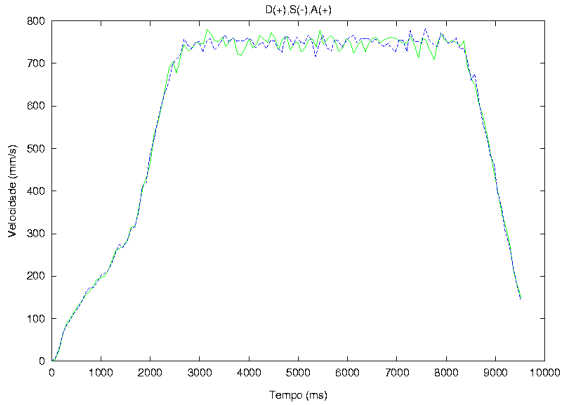

Figura C.6: Dados obtidos durante o sexto experimento.

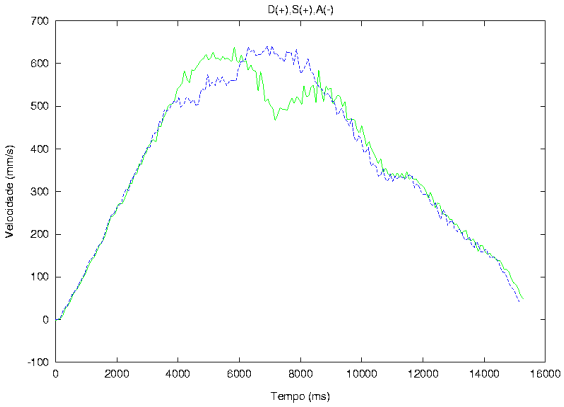

Figura C.7: Dados obtidos durante o sétimo experimento.

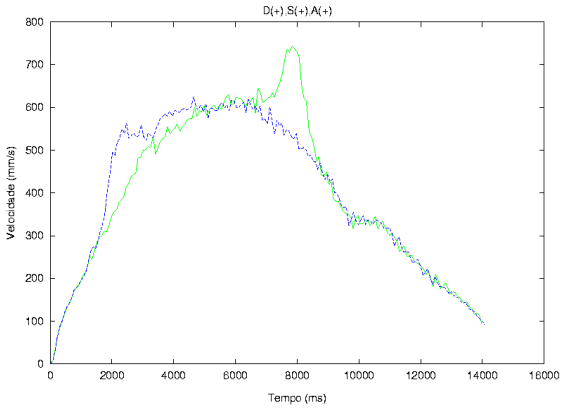

Figura C.8: Dados obtidos durante o oitavo experimento.

Apêndice D

Formulação Matemática dos Gráficos de Orientação

Os gráficos de orientação utilizados no *applet* de controle do sistema *pan-tilt* foram desenvolvidos através das transformações de coordenadas homogêneas, como mostra o equacionamento a seguir:

an

Figura D.1 mostra a geometria utilizada no desenvolvimento das matrizes de transformação:

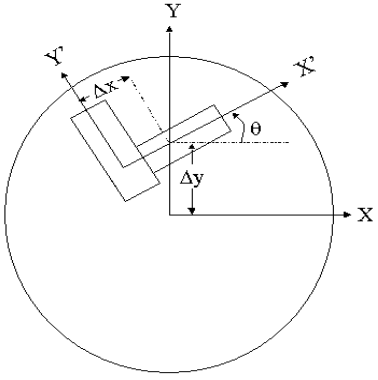

Figura D.1: Geometria do movimento *pan*.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\mu & -s\mu & 0 \\ s\mu & c\mu & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow P' = \begin{bmatrix} c\mu \cdot x - s\mu \cdot y - c\mu \cdot \Delta x \\ s\mu \cdot x + c\mu \cdot y - s\mu \cdot \Delta x + \Delta y \\ 1 \end{bmatrix}$$

Tilt

A figura 4.2 mostra a geometria utilizada no desenvolvimento das matrizes de transformação:

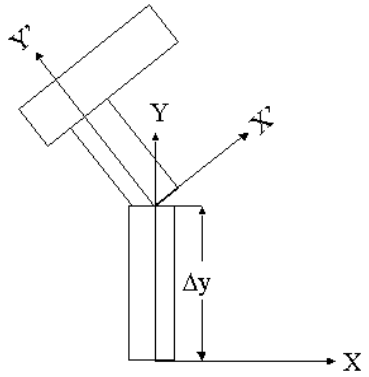

Figura 4.2: Geometria do movimento *tilt*.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\mu & -s\mu & 0 \\ s\mu & c\mu & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \; ! \; P' = \begin{bmatrix} c\mu \cdot x - s\mu \cdot y \\ s\mu \cdot x + c\mu \cdot y + \Delta y \\ 1 \end{bmatrix}$$

Apêndice

Listagem dos Códigos-Fonte dos Programas Desenvolvidos

.1 Servidores em Linguagem C

.1.1 RobServer

```
/*****************************************************************************
ROBSERVER.C - Robot's server through the INETD main server
 Sergio Roberto Gonsalves Tourino - tourino@graco.unb.br - 10/05/2000

Before using this program, edit the 'inetd.conf' and the 'services' files:

INETD.CONF:
 robserver  stream  tcp     nowait  root /home/tourino/robserver robserver

SERVICES:
 robserver  8086/tcp

After these modifications, restart the inetd server:
 kill -1 <PID do inetd - ver com ps>
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <signal.h>
#include "Nclient.h"

#define HOST "lower"
#define PORT 7073

#define NONE 0
#define FRONT 1
#define BACK 2
#define LEFT 3
```

```
#define RIGHT 4
#define SPINP 5
#define SPINN 6
#define ACELERATION 500
#define NOFF

void robot_panel (char *message);
void start_sensors (void);
void stop_sensors (void);
int disconnect (void);
void get_battery (void);
void get_IR (void);
void get_sonar (void);
void move_robot (int direction, int distance, int speed, int teta, int omega);

/*-----------------3/29/00 5:17PM------------------
 * This main routine parses commands from stdin, which
 * corresponds to a socket connection from a remote client
 * connected to this server. Its outputs are sent to the
 * stdout stream, back to the client machine. The local
 * tests are made with the stderr stream. This program is
 * to be used with the INETD main server.
 * --------------------------------------------------*/

int sensor_on = FALSE; /* global variable to verify sensor's state */
int stopnow = FALSE; /* global variable to stop the robot */

/*****************************************************************************/

void goodbye (void) {
int result;

        result = system ("rm /home/tourino/robserver.lock");
}

/*****************************************************************************/

void stoprobot () {

stopnow = TRUE;
printf ("Robot stopped!\n");
}

/*****************************************************************************/

int main (void) {

    const char delimiters[] = "=, ";    /* delimiters for the commands */
    char command[] = "direction=front, distance=500, nothing important, direction=left, goodbye";
    char *token, parameters[10], commandline[100];
    char direction[10];                 /* direction for the movement */

    char sensors[3]; /* state of sensors */
    char get[10];

    int open_connection = TRUE;
```

```
    int result,pid;
    int dir, distance = 500, speed=100, teta=524, omega=314; /* movement configuration */

    FILE *connection, *lock, *mypid;

    /* verify the lock file */

    lock = fopen ("/home/tourino/robserver.lock","r");

    if (lock == NULL) {                /* the lock does not exist */
        lock = fopen ("/home/tourino/robserver.lock","w+");
        fclose (lock);
    }
    else {                             /* the lock exists */
        fclose (lock);
        fprintf (stderr,"Lock file found. Program aborted.\n");
        exit (0);                      /* closes the program */
    }

    atexit (goodbye); /* defines the function 'goodbye' to be
called at program termination */

    /* create the PID file */

    pid = getpid ();        /* getting the pid of program */

    /* saving pid */
    mypid = fopen ("/home/tourino/robserver.pid","w+");
    fprintf (mypid,"%d",pid);
    fclose (mypid);

    /* defines the listener for the stop signal from the stop server */

    signal (SIGUSR1,stoprobot);

    connection = fopen ("/home/tourino/robserver.out","w+"); /* opening file */

    N_InitializeClient (HOST,PORT); /* opening robot's connection */
    result = N_ConnectRobot (1);

switch (result) {
case N_NO_ERROR:
fprintf (stderr,"Success!\n");
break;
case N_UNINITIALIZED:
fprintf (stderr,"Not initialized!\n");
exit(0);
break;
case N_ROBOT_NOT_FOUND:
fprintf (stderr,"Robot not found!\n");
exit(0);
break;
case N_CONNECTION_FAILED:
fprintf (stderr,"Connection failed!\n");
exit(0);
break;
```

```
case N_OUT_OF_MEMORY:
fprintf (stderr,"Out of memory!\n");
exit(0);
break;
}

    while (open_connection) {           /* loop until 'open_connection' is false */

        fscanf (stdin,"%s",command);    /* reads commands from stdin */
fprintf (connection,"%s\n",command); /* writes to file */
fflush (connection); /* it's better... */

        token = strtok(command,delimiters);  /* strtok parses the commands from 'command' variable */

        while ((token != NULL) && open_connection) {

            if (strcmp(token,"sensors") == 0) {
                strcpy (sensors,strtok(NULL,delimiters));
if (strcmp (sensors,"on") == 0) {
start_sensors ();
sensor_on = TRUE;
}
if (strcmp (sensors,"off") == 0) {
stop_sensors ();

/* since who is turning off the sensors will stop the session when I
remove the lock file sooner... */

        result = system ("rm /home/tourino/robserver.lock");
sensor_on = FALSE;
}
            }

    if (strcmp(token,"direction") == 0) {
strcpy (direction,strtok(NULL,delimiters));
    }

    if (strcmp(token,"distance") == 0) {
distance = atoi (strtok (NULL,delimiters));
    }

    if (strcmp(token,"speed") == 0) {
speed = atoi (strtok (NULL,delimiters));
    }

    if (strcmp(token,"teta") == 0) {
teta = atoi (strtok (NULL,delimiters));
    }

    if (strcmp(token,"omega") == 0) {
omega = atoi (strtok (NULL,delimiters));
    }

            if (strcmp(token,"start") == 0) {
                /* Here comes the commands for the control of the robot.
                 * This command MUST only work with all the variables (speed, acceleration, ...)
```

```
                * adjusted accordingly. */
if (strcmp(direction,"front") == 0)
dir = FRONT;
if (strcmp(direction,"back") == 0)
dir = BACK;
if (strcmp(direction,"left") == 0)
dir = LEFT;
if (strcmp(direction,"right") == 0)
dir = RIGHT;
if (strcmp(direction,"clockwise") == 0)
dir = SPINN;
if (strcmp(direction,"counterclockwise") == 0)
dir = SPINP;

if (sensor_on) {
move_robot (dir,distance,speed,teta,omega); /* moves the robot */
strcpy (direction,""); /* clears the variable */
dir = NONE;
}
else {
fprintf (stderr,"Error: sensors turned off.\n");
}
            }

            if (strcmp(token,"goodbye") == 0) {
                open_connection = FALSE;  /* terminates all of the loops */
            }

            token = strtok (NULL,delimiters);
        }
    }

    fprintf (stderr,"Goodbye. Connection closed.\n");

    fclose (connection);
    stop_sensors();

    result = disconnect ();

    return 0;
}

/*****************************************************************************/

struct MySonar {
struct N_Sonar sonar[N_MAX_SONAR_COUNT];
};

struct MyBumper {
struct N_Bumper bumper[N_MAX_BUMPER_COUNT];
};

struct Map {
int Reading[24];
};
```

```
void ReadSonarSet (int ss_number, struct MySonar *sonarset);
void ShowSonarSet (struct MySonar sonarset);
void MinSonarSet (struct MySonar sonar1, struct MySonar sonar2,
struct MySonar *minsonar);
void MapSonarSet (struct MySonar sonar0, struct MySonar sonar1,
struct MySonar sonar2, struct Map *map);
void ShowMap (struct Map map);
void MakeSonarMap (struct Map *map);
void ShowSonarMap (struct Map map);
int MapAngle (int angle);
int GetSMapReading (struct Map map, int angle, int n_sensors);
void ShowEvidence (struct Map map);

/*************************************************************************/

struct MyIR {
struct N_Infrared IR [N_MAX_SONAR_COUNT];
};

/*************************************************************************/

void ReadIRSet (int ss_number, struct MyIR *IRset);
void ShowIRSet (struct MyIR IRset);
void MaxIRSet (struct MyIR IR1, struct MyIR IR2,
struct MyIR *minIR);
void MapIRSet (struct MyIR IR0, struct MyIR IR1,
struct MyIR IR2, struct Map *map);
void ShowIRMap (struct Map map);
void MakeIRMap (struct Map *map);
void MakeBumperMap (struct Map *map);
int GetIMapReading (struct Map map, int angle, int n_sensors);

/*************************************************************************/

#define X N_XTRANSLATION
#define Y N_YTRANSLATION
#define T N_ROTATION

#define MIN_DIST 300 /* value in milimeters  */
#define MAX_INT 30 /* max value is 255 */

#define TO_RADIAN 3.14159265358979/180

/*************************************************************************/

/* Routine used to start the sonar sensors */

void start_sensors (void)
{
struct N_RobotState *state;
struct N_SonarSet *sonar_set;
int i,j;

state = N_GetRobotState(1);

for (i=0;i<6;i++) {
```

```
sonar_set = &(state->SonarController.SonarSet[i]);
N_GetSonarConfiguration(1);

sonar_set->DataActive = TRUE;
sonar_set->TimeStampActive = TRUE;

for (j=0;j<=7;j++)
sonar_set->FiringOrder[j] = j;
sonar_set->FiringOrder[8] = N_END_SONAR_FIRING_ORDER;

N_SetSonarConfiguration(1);

fprintf (stderr,"Sonar set %d set to\n",i);

fprintf (stderr,"Firing order: ");
j = 0;
while (sonar_set->FiringOrder[j] !=
N_END_SONAR_FIRING_ORDER) {
fprintf (stderr,"%d ",sonar_set->FiringOrder[j]);
j++;
}
}
}

/*****************************************************************************/

/* Routine used to turn off the sonar sensors */

void stop_sensors (void)
{
struct N_RobotState *state;
struct N_SonarSet *sonar_set;
int i;

state = N_GetRobotState(1);

for (i=0;i<6;i++) {
sonar_set = &(state->SonarController.SonarSet[i]);
N_GetSonarConfiguration(1);

sonar_set->DataActive = TRUE;
sonar_set->TimeStampActive = TRUE;
sonar_set->FiringOrder[0] = N_END_SONAR_FIRING_ORDER;

N_SetSonarConfiguration(1);
}
}

/*****************************************************************************/

/* Routine used to write to the robot's panel */

void robot_panel (char *message)
{
char  *to_panel;
```

```
int  result;

strcpy (to_panel,"/usr/local/xrdev/bin/cpnl_log '");
strcat (to_panel,message);
strcat (to_panel,"'");

result = system (to_panel);

if (result == -1 || result == 127)
fprintf (stderr,"Error while calling 'cpnl_log' program.\n");
}

/*************************************************************************/

/* Routine that closes the connection with the XRDev */

int disconnect (void)
{
int result;
result = N_DisconnectRobot (1);

switch (result) {
case N_NO_ERROR:
printf ("Success!\n");
break;
case N_CONNECTION_FAILED:
printf ("Connection lost!\n");
break;
case N_ROBOT_NOT_FOUND:
printf ("Robot not found!\n");
break;
}

return (result);
}

/*************************************************************************/

void moveaxis (int axis_to_move, int desired_motion, int desired_speed);

/*************************************************************************/

/* Moves the robot in the direction, distance and speed provided */

void move_robot (int direction, int distance, int speed, int teta, int omega) {

int ir, sonar, dist;
struct  Map mapS, mapI, mapB;

if (!sensor_on)
return; /* returns if the sensors are not on */

dist = distance;

MakeSonarMap (&mapS);
MakeIRMap (&mapI);
```

```
MakeBumperMap (&mapB);
switch (direction) {
case RIGHT:
if ((GetSMapReading (mapS,0,5) > MIN_DIST)
&& (GetIMapReading (mapI,0,5) < MAX_INT)
&& (GetBMapReading (mapB,0,5) == N_BUMPER_NONE))
moveaxis (X, dist, speed);
break;
case LEFT:
if ((GetSMapReading (mapS,180,5) > MIN_DIST)
&& (GetIMapReading (mapI,180,5) < MAX_INT)
&& (GetBMapReading (mapB,0,5) == N_BUMPER_NONE))
moveaxis (X, -dist, speed);
break;
case FRONT:
if ((GetSMapReading (mapS,90,5) > MIN_DIST)
&& (GetIMapReading (mapI,90,5) < MAX_INT)
&& (GetBMapReading (mapB,0,5) == N_BUMPER_NONE))
moveaxis (Y, dist, speed);
break;
case BACK:
if ((GetSMapReading (mapS,270,5) > MIN_DIST)
&& (GetIMapReading (mapI,270,5) < MAX_INT)
&& (GetBMapReading (mapB,0,5) == N_BUMPER_NONE))
moveaxis (Y, -dist, speed);
break;
case SPINP:
moveaxis (T, teta, omega);
break;
case SPINN:
moveaxis (T, -teta, omega);
break;
};

MakeSonarMap (&mapS);
/* ShowEvidence (mapS); */

}

/*****************************************************************************/


/* Moves the specified axis of the robot */

void moveaxis (int axis_to_move, int desired_motion, int desired_speed) {

struct N_RobotState *state;
struct N_Axis *axis;
struct N_AxisSet *axisSet;
struct Map mapS, mapI, mapB;
long inProgress, i, ir, spin,
check_sonar, min_dist, initial,
here, loops;
char c;
int  angle;
float sonar, distance, speed;
```

```
FILE *outfile;

        outfile = fopen ("/home/tourino/speed.out","a");

if (!sensor_on)
return; /* returns if the sensors are not on */

min_dist = MIN_DIST;

/* defines the min_dist as proportional to the speed */

switch (axis_to_move) {
case X:
angle = desired_motion > 0 ? 0 : 180;
spin = FALSE;
break;
case Y:
angle = desired_motion > 0 ? 90 : 270;
spin = FALSE;
break;
case T:
spin = TRUE;
break;
};

state = N_GetRobotState(1);
axisSet = &(state->AxisSet);
axis = &(axisSet->Axis[axis_to_move]);
axis->DataActive = TRUE;
N_GetAxes(1);
initial = axis->ActualPosition;

state = N_GetRobotState (1);
axisSet = &(state->AxisSet);
axis = &(axisSet->Axis[axis_to_move]);
axisSet->Global = FALSE;
axis->DataActive = TRUE;
axis->Mode=N_AXIS_POSITION_ABSOLUTE;
axis->DesiredSpeed = 100; /* start "stopped" */
/* axis->DesiredSpeed = desired_speed; */
axis->DesiredPosition = initial+desired_motion;
axis->Acceleration = ACELERATION;
axis->Update = TRUE;
N_SetAxes (1);

loops = 0;

do {
loops++;

/* Reading sensor's data */
MakeSonarMap (&mapS);
MakeIRMap (&mapI);

/* Reading data for the fuzzy controller */
```

```
sonar = GetSMapReading (mapS,angle,5);

state = N_GetRobotState (1);
axisSet = &(state->AxisSet);
axis = &(state->AxisSet.Axis[axis_to_move]);
axis->DataActive = TRUE;
N_GetAxes(1);

here = axis->ActualPosition;

distance = abs(desired_motion - here + initial);
printf ("distance=%d\n",distance);

/* call to the fuzzy controller */
#ifdef OFF
distance=8000;
#endif
fuzzycontrol (&sonar,&distance,&speed);

/* setting the new speed from fuzzy controller */

state = N_GetRobotState (1);
axisSet = &(state->AxisSet);
axis = &(state->AxisSet.Axis[axis_to_move]);
axis->DesiredSpeed = (long) speed;
axis->DesiredPosition = initial+desired_motion;
axis->Mode=N_AXIS_POSITION_ABSOLUTE;
axis->Update = TRUE;
axis->DataActive = TRUE;
axis->TimeStampActive = TRUE;
N_SetAxes (1);

state = N_GetRobotState (1);
axis->DataActive = TRUE;
N_GetAxes (1);
fprintf (outfile,"%ld %d\n",axis->TimeStamp,axis->ActualVelocity);
inProgress = axis->InProgress;

MakeSonarMap (&mapS);
MakeIRMap (&mapI);
MakeBumperMap (&mapB);

check_sonar = spin ? TRUE : ((GetSMapReading
(mapS,angle,9) > min_dist) && (GetIMapReading (mapI,angle,9) < MAX_INT));
/* && (GetBMapReading (mapB,0,5) != N_BUMPER_NONE));
*/

/* if spin is true, the routine ignores the sensor's data (the robot will
not collide turning by its axis) */

}
while (inProgress && check_sonar && (!stopnow) );
axis->DesiredSpeed = 0;
axis->DesiredPosition = here;
axis->Update = TRUE;
N_SetAxes (1);
```

```
printf ("loops=%d\n",loops);

stopnow = FALSE; /* the robot can move again */

fflush (outfile);
fclose (outfile);
}

/*****************************************************************************/

/* Gets the lower value of the sonar sets */

void MinSonarSet (struct MySonar sonar1, struct MySonar sonar2,
struct MySonar *minsonar) {

int i;

for (i=0;i<8;i++) {
minsonar->sonar[i].Reading = sonar1.sonar[i].Reading <
sonar2.sonar[i].Reading?sonar1.sonar[i].Reading:
sonar2.sonar[i].Reading;
}
}

/*****************************************************************************/

/* Reads sonar data and stores it in the MySonar structure */

void ReadSonarSet (int ss_number, struct MySonar *sonarset) {

struct N_RobotState *state;
struct N_Sonar *sonar;
int i;

state = N_GetRobotState (1);
sonar = state->SonarController.SonarSet[ss_number].Sonar;
N_GetSonar (1);

for (i=0;i<8;i++)
sonarset->sonar[i].Reading=sonar[i].Reading;
}

/*****************************************************************************/

/* Reads bumper data and stores it in the MyBumper structure */

void ReadBumperSet (int bs_number, struct MyBumper *bumperset) {

struct N_RobotState  *state;
struct N_Bumper *bumper;
int i;

state = N_GetRobotState (1);
bumper = state->BumperController.BumperSet[bs_number].Bumper;
N_GetBumper (1);
```

```
for (i=0;i<8;i++)
bumperset->bumper[i].Reading=bumper[i].Reading;
}

/*************************************************************************/

/* Prints out the sonar data to the stderr output */

void ShowSonarSet (struct MySonar sonarset) {

int i;

for (i=0;i<8;i++)
fprintf (stderr,"%d ",sonarset.sonar[i].Reading);
fprintf (stderr,"\n");
}

/*************************************************************************/

/* Translates the sonar data into MySonar structure to the Map structure */

void MapSonarSet (struct MySonar sonar0, struct MySonar sonar1,
struct MySonar sonar2, struct Map *map) {

int angle, i;

angle = 0;

for (i=6;i<8;i++) {
map->Reading[angle] = sonar2.sonar[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = sonar0.sonar[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = sonar1.sonar[i].Reading;
angle++;
}

for (i=0;i<6;i++) {
map->Reading[angle] = sonar2.sonar[i].Reading;
angle++;
}
}

/*************************************************************************/

/* Translates the bumper data into MyBumper structure to the Map structure
*/

void MapBumperSet (struct MyBumper bumper0, struct MyBumper bumper1,
```

```
struct MyBumper bumper2, struct Map *map) {

int angle, i;

angle = 0;

for (i=6;i<8;i++) {
map->Reading[angle] = bumper2.bumper[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = bumper0.bumper[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = bumper1.bumper[i].Reading;
angle++;
}

for (i=0;i<6;i++) {
map->Reading[angle] = bumper2.bumper[i].Reading;
angle++;
}
}

/*****************************************************************************/

/* Prints out the Map structure to the stderr output */

void ShowMap (struct Map map) {

int  i;

for (i=0;i<24;i++)
fprintf (stderr,"%.1f",map.Reading[i]>10000?10:map.Reading[i]/1000.0);

fprintf (stderr,"\n");
}

/*****************************************************************************/

/* Returns the greater value from the MyBumper structures */

void MaxBumperSet (struct MyBumper bumper1, struct MyBumper bumper2,
struct MyBumper *minbumper) {

int i;

for (i=0;i<8;i++) {
minbumper->bumper[i].Reading = bumper1.bumper[i].Reading >
bumper2.bumper[i].Reading?bumper1.bumper[i].Reading:bumper2.bumper[i].Reading;
}

}
```

```
/*************************************************************************/

/* Translate all the bumper sets into the Map structure */

void MakeBumperMap (struct Map *map) {

struct MyBumper bs,bi,bumper0,bumper1,bumper2;

ReadBumperSet (0,&bs);
ReadBumperSet (1,&bi);
MaxBumperSet (bs,bi,&bumper0);
ReadBumperSet (2,&bs);
ReadBumperSet (3,&bi);
MaxBumperSet (bs,bi,&bumper1);
ReadBumperSet (4,&bs);
ReadBumperSet (5,&bi);
MaxBumperSet (bs,bi,&bumper2);

MapBumperSet (bumper0,bumper1,bumper2,map);
}

/*************************************************************************/

/* Translate all the sonar sets into the Map structure */

void MakeSonarMap (struct Map *map) {

struct MySonar ss,si,sonar0,sonar1,sonar2;

ReadSonarSet (0,&ss);
ReadSonarSet (1,&si);
MinSonarSet (ss,si,&sonar0);
ReadSonarSet (2,&ss);
ReadSonarSet (3,&si);
MinSonarSet (ss,si,&sonar1);
ReadSonarSet (4,&ss);
ReadSonarSet (5,&si);
MinSonarSet (ss,si,&sonar2);

MapSonarSet (sonar0,sonar1,sonar2,map);
}

/*************************************************************************/

/* Displays a bar graph of the sonar data into the stderr output */

void ShowSonarMap (struct Map map) {

short i,j;
char s[80];

for (i=0;i<24;i++) {
strcpy (s,"");
       for (j=0;j < (map.Reading[i]>10000?10000:map.Reading[i])*0.006;j++)
         strcat (s,"-");
```

```
        fprintf (stderr,"%d\t%s\n",(map.Reading[i]>10000?10000:map.Reading[i]),s);
}

refresh();
usleep (100000);
clear ();
}

/*****************************************************************************/

/* Maps a provided angle to the Map structure's index */

int MapAngle (int angle) {

int a;
a = ceil (0.06969*angle-1.04545);
a = a < 0 ? 24+a : a;
return (a);
}

/*****************************************************************************/

/* Gets the value of sonar data in a specified direction */

int GetSMapReading (struct Map map, int angle, int n_sensors) {

int s, i, a, n, t;

n = n_sensors / 2;
a = MapAngle (angle);
s = 10000; /* max value to sonar data */
for (i=0;i<2*n+1;i++) {
t = a-n+i;
t = t < 0 ? 24-t : t;
t = t > 23 ?  t-24 : t;
s = map.Reading[t]<s ? map.Reading[t] : s;
}
return (s);
}

/*****************************************************************************/

/* Gets the value of IR data in a specified direction */

int GetIMapReading (struct Map map, int angle, int n_sensors) {

int s, i, a, n, t;

n = n_sensors / 2;
a = MapAngle (angle);
s = 0; /* min value for IR data */
for (i=0;i<2*n+1;i++) {
t = a-n+i;
t = t < 0 ? 24-t : t;
t = t > 23 ?  t-24 : t;
s = map.Reading[t]>s ? map.Reading[t] : s;
```

```
}
return (s);
}

/*************************************************************************/

/* Gets the value of Bumper data in a specified direction */

int GetBMapReading (struct Map map, int angle, int n_sensors) {

int s, i, a, n, t;

n = n_sensors / 2;
a = MapAngle (angle);
s = N_BUMPER_NONE; /* min value for bumper data */
for (i=0;i<2*n+1;i++) {
t = a-n+i;
t = t < 0 ? 24-t : t;
t = t > 23 ?  t-24 : t;
s = map.Reading[t]>s ? map.Reading[t] : s;
}
return (s);
}

/*************************************************************************/

/* Displays a gridmap containing the values of Map structure */

void ShowEvidence (struct Map map) {

int  i,j,l,xm,ym;
int  EGrid [20][20];
char  s[80];

for (i=0; i < 20; i++)
for (j=0; j < 20; j++)
EGrid [i][j] = 0;

EGrid [10][10] = 2;

for (i=0; i < 360; i+= 15) {
l = GetSMapReading (map, i, 1);
xm = (3 + (l+300) * cos (i*TO_RADIAN) / 1000) * 3;
ym = (3 + (l+300) * sin (i*TO_RADIAN) / 1000) * 3;
if ((xm < 20 && xm >= 0) && (ym >= 0 && ym < 20))
EGrid [xm][ym] = 1;
}

/* plotting from up to down... */

for (j=19; j >= 0; j--) {
strcpy (s,"");
for (i=0; i < 20; i++) {
if (EGrid [i][j] == 1)
strcat (s,"++");
else
```

```
if (EGrid [i][j] == 2)
strcat (s,"OO");
else
strcat (s,"..");
}
fprintf (stderr,"%s\n",s);
}
usleep (200000);
}

/*************************************************************************/

/* Returns the greater value from the MyIR structures */

void MaxIRSet (struct MyIR IR1, struct MyIR IR2,
struct MyIR *minIR) {

int i;

for (i=0;i<8;i++) {
minIR->IR[i].Reading = IR1.IR[i].Reading >
IR2.IR[i].Reading?IR1.IR[i].Reading:IR2.IR[i].Reading;
}
}

/*************************************************************************/

/* Translates the IR data into the MyIR structure */

void ReadIRSet (int ss_number, struct MyIR *IRset) {

struct N_RobotState *state;
struct N_Infrared *IR;
int i;

state = N_GetRobotState (1);
IR = state->InfraredController.InfraredSet[ss_number].Infrared;
N_GetInfrared (1);

for (i=0;i<8;i++)
IRset->IR[i].Reading=IR[i].Reading;
}

/*************************************************************************/

/* Prints out the IR data from a MyIR structure */

void ShowIRSet (struct MyIR IRset) {

int i;

for (i=0;i<8;i++)
fprintf (stderr,"%d ",IRset.IR[i].Reading);
fprintf (stderr,"\n");
}
```

```
/*************************************************************************/

/* Translates MyIR structures into the Map structure */

void MapIRSet (struct MyIR IR0, struct MyIR IR1,
struct MyIR IR2, struct Map *map) {

int angle, i;

angle = 0;

for (i=6;i<8;i++) {
map->Reading[angle] = IR2.IR[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = IR0.IR[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = IR1.IR[i].Reading;
angle++;
}

for (i=0;i<6;i++) {
map->Reading[angle] = IR2.IR[i].Reading;
angle++;
}
}

/*************************************************************************/

/* Prints out the Map structure for the IR data */

void ShowIRMap (struct Map map) {

int  i;

for (i=0;i<24;i++)
fprintf (stderr,"%d ",map.Reading[i]);

fprintf (stderr,"\n");
}

/*************************************************************************/

/* Translates all the IR data into the Map structure */

void MakeIRMap (struct Map *map) {

struct MyIR ss,si,IR0,IR1,IR2;

ReadIRSet (0,&ss);
ReadIRSet (1,&si);
```

```
MaxIRSet (ss,si,&IR0);
ReadIRSet (2,&ss);
ReadIRSet (3,&si);
MaxIRSet (ss,si,&IR1);
ReadIRSet (4,&ss);
ReadIRSet (5,&si);
MaxIRSet (ss,si,&IR2);

MapIRSet (IR0,IR1,IR2,map);
}
```

.1.2 SenServer

```
/*****************************************************************************
SENSERVER.C - Robot's sensor server through the INETD main server
 Sergio Roberto Gonsalves Tourino - tourino@graco.unb.br - 10/05/2000

Before using this program, edit the 'inetd.conf' and the 'services' files:

INETD.CONF:
 senserver  stream  tcp     nowait  root /home/tourino/senserver senserver

SERVICES:
 senserver  8085/tcp

After these modifications, restart the inetd server:
 kill -1 <PID of inetd - see with ps>
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "Nclient.h"

#define HOST "lower"
#define PORT 7073

/* #define DEBUG */

int disconnect (void);
void get_battery (void);
void get_IR (void);
void get_sonar (void);
void get_odometry (void);
void get_speed (void);

/*-----------------3/29/00 5:17PM------------------
 * This main routine parses commands from stdin, which
 * corresponds to a socket connection from a remote client
 * connected to this server. Its outputs are sent to the
 * stdout stream, back to the client machine. The local
 * tests are made with the stderr stream. This program is
 * to be used with the INETD main server.
 * -------------------------------------------------*/
```

```
int sensor_on = FALSE; /* global variable to verify sensor's state */

/*************************************************************************/

void goodbye (void) {
int result;

     result = system ("rm /home/tourino/senserver.lock");
}

/*************************************************************************/

int main (void) {

    const char delimiters[] = "=, ";    /* delimiters for the commands */
    char command[] = "direction=front, distance=500, nothing important, direction=left, goodbye";
    char *token, parameters[10], commandline[100];
    char direction[10];                 /* direction for the movement */

    char sensors[3]; /* state of sensors */
    char get[10];

    int open_connection = TRUE;
    int result;
    int dir, distance = 500, speed=100, teta=524, omega=314; /* movement configuration */

    FILE *lock;

    /* verify the lock file */

    lock = fopen ("/home/tourino/senserver.lock","r");

    if (lock == NULL) {                 /* the lock does not exist */
        lock = fopen ("/home/tourino/senserver.lock","w+");
        fclose (lock);
    }
    else {                              /* the lock exists */
        fclose (lock);
        fprintf (stderr,"Lock file found. Program aborted.\n");
        exit (0);                       /* closes the program */
    }

    atexit (goodbye); /* defines the 'goodbye' function to be
called at program termination */

    N_InitializeClient (HOST,PORT); /* opening robot's connection */
    result = N_ConnectRobot (1);

switch (result) {
case N_NO_ERROR:
fprintf (stderr,"Success!\n");
break;
case N_UNINITIALIZED:
fprintf (stderr,"Not initialized!\n");
exit(0);
break;
```

```
case N_ROBOT_NOT_FOUND:
fprintf (stderr,"Robot not found!\n");
exit(0);
break;
case N_CONNECTION_FAILED:
fprintf (stderr,"Connection failed!\n");
exit(0);
break;
case N_OUT_OF_MEMORY:
fprintf (stderr,"Out of memory!\n");
exit(0);
break;
}

    while (open_connection) {           /* loop until 'open_connection' is false */

        fscanf (stdin,"%s",command);    /* reads commands from stdin */

        token = strtok(command,delimiters);  /* strtok parses the commands from 'command' variable */

        while ((token != NULL) && open_connection) {

    if (strcmp(token,"get") == 0) {
strcpy (get,strtok(NULL,delimiters));
if (strcmp (get,"battery") == 0) {
get_battery ();
}
if (strcmp (get,"IR") == 0) {
get_IR ();
}
if (strcmp (get,"sonar") == 0) {
get_sonar ();
}
if (strcmp (get,"odometry") == 0) {
get_odometry ();
}
if (strcmp (get,"speed") == 0) {
get_speed ();
}
    }

            if (strcmp(token,"goodbye") == 0) {
                open_connection = FALSE;  /* terminates all of the loops */
            }

            token = strtok (NULL,delimiters);
        }
    }

    fprintf (stderr,"Goodbye. Connection closed.\n");

    result = disconnect ();

    return 0;
}
```

```
/*************************************************************************/

struct MySonar {
struct N_Sonar sonar[N_MAX_SONAR_COUNT];
};

struct MyBumper {
struct N_Bumper bumper[N_MAX_BUMPER_COUNT];
};

struct Map {
int Reading[24];
};

void ReadSonarSet (int ss_number, struct MySonar *sonarset);
void ShowSonarSet (struct MySonar sonarset);
void MinSonarSet (struct MySonar sonar1, struct MySonar sonar2,
struct MySonar *minsonar);
void MapSonarSet (struct MySonar sonar0, struct MySonar sonar1,
struct MySonar sonar2, struct Map *map);
void ShowMap (struct Map map);
void MakeSonarMap (struct Map *map);
void ShowSonarMap (struct Map map);
int MapAngle (int angle);
int GetSMapReading (struct Map map, int angle, int n_sensors);
void ShowEvidence (struct Map map);

/*************************************************************************/

struct MyIR {
struct N_Infrared IR [N_MAX_SONAR_COUNT];
};

/*************************************************************************/

void ReadIRSet (int ss_number, struct MyIR *IRset);
void ShowIRSet (struct MyIR IRset);
void MaxIRSet (struct MyIR IR1, struct MyIR IR2,
struct MyIR *minIR);
void MapIRSet (struct MyIR IR0, struct MyIR IR1,
struct MyIR IR2, struct Map *map);
void ShowIRMap (struct Map map);
void MakeIRMap (struct Map *map);
void MakeBumperMap (struct Map *map);
int GetIMapReading (struct Map map, int angle, int n_sensors);

/*************************************************************************/

#define TO_RADIAN 3.14159265358979/180

/*************************************************************************/

/* Routine that closes the connection with the XRDev */

int disconnect (void)
{
```

```
int result;
result = N_DisconnectRobot (1);

switch (result) {
case N_NO_ERROR:
printf ("Success!\n");
break;
case N_CONNECTION_FAILED:
printf ("Connection lost!\n");
break;
case N_ROBOT_NOT_FOUND:
printf ("Robot not found!\n");
break;
}

return (result);
}

/*****************************************************************************/

/* Routine used to get the batteries' voltages */

void get_battery (void) {

struct N_RobotState *state;
struct N_Battery *battery;
int i;

state = N_GetRobotState (1);

fprintf (stdout," B");
for (i=0;i<N_MAX_BATTERY_COUNT;i++) {
battery = &(state->BatterySet.Battery[i]);
N_GetBattery (1);
fprintf (stdout," %d",battery->Voltage/100);
}
fprintf (stdout," B ");

fflush (stdout);
}

/*****************************************************************************/

void get_IR (void) {

struct Map mapI;
int i;

MakeIRMap (&mapI);

fprintf (stdout," I");
for (i=0;i<24;i++)
fprintf (stdout," %d",mapI.Reading[i]/10);

fprintf (stdout," I ");
fflush (stdout);
```

```
}

/*****************************************************************************/

void get_sonar (void) {

struct Map mapS;
int i;

#ifdef DEBUG
FILE *outfile;

outfile = fopen ("/home/tourino/sensors.out","a");
#endif

MakeSonarMap (&mapS);

fprintf (stdout," S");
for (i=0;i<24;i++) {
fprintf (stdout," %d",mapS.Reading[i]/100);
#ifdef DEBUG
fprintf (outfile," %d",mapS.Reading[i]);
#endif
}

fprintf (stdout," S ");
fflush (stdout);

#ifdef DEBUG
fprintf (outfile,"\n");
fflush (outfile);
fclose (outfile);
#endif

}

/*****************************************************************************/

void get_odometry (void) {

struct N_RobotState *state;
struct N_Integrator *integrated;

state = N_GetRobotState (1);
integrated = &(state->Integrator);

N_GetIntegratedConfiguration (1);
fprintf (stdout," O %d %d %d %d O ",
integrated->x, integrated->y,
integrated->Steering, integrated->Rotation);
fflush (stdout);
}


/*****************************************************************************/
```

```
void get_speed (void) {

struct N_RobotState *state;
struct N_Axis *axis;

state = N_GetRobotState(1);

axis = &(state->AxisSet.Axis[X]);
N_GetAxes(1);
fprintf (stdout," V");
fprintf (stdout," %d ",axis->ActualVelocity);

axis = &(state->AxisSet.Axis[Y]);
N_GetAxes(1);
fprintf (stdout,"%d ",axis->ActualVelocity);

axis = &(state->AxisSet.Axis[T]);
N_GetAxes(1);
fprintf (stdout,"%d V ",axis->ActualVelocity);

fflush(stdout);
}

/*************************************************************************/

/* Gets the lower value of the sonar sets */

void MinSonarSet (struct MySonar sonar1, struct MySonar sonar2,
struct MySonar *minsonar) {

int i;

for (i=0;i<8;i++) {
minsonar->sonar[i].Reading = sonar1.sonar[i].Reading <
sonar2.sonar[i].Reading?sonar1.sonar[i].Reading:
sonar2.sonar[i].Reading;
}
}

/*************************************************************************/

/* Reads sonar data and stores it in the MySonar structure */

void ReadSonarSet (int ss_number, struct MySonar *sonarset) {

struct N_RobotState *state;
struct N_Sonar *sonar;
int i;

state = N_GetRobotState (1);
sonar = state->SonarController.SonarSet[ss_number].Sonar;
N_GetSonar (1);

for (i=0;i<8;i++)
sonarset->sonar[i].Reading=sonar[i].Reading;
}
```

```
/*************************************************************************/

/* Reads bumper data and stores it in the MyBumper structure */

void ReadBumperSet (int bs_number, struct MyBumper *bumperset) {

struct N_RobotState  *state;
struct N_Bumper *bumper;
int i;

state = N_GetRobotState (1);
bumper = state->BumperController.BumperSet[bs_number].Bumper;
N_GetBumper (1);

for (i=0;i<8;i++)
bumperset->bumper[i].Reading=bumper[i].Reading;
}

/*************************************************************************/

/* Translates the sonar data into MySonar structure to the Map structure */

void MapSonarSet (struct MySonar sonar0, struct MySonar sonar1,
struct MySonar sonar2, struct Map *map) {

int angle, i;

angle = 0;

for (i=6;i<8;i++) {
map->Reading[angle] = sonar2.sonar[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = sonar0.sonar[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = sonar1.sonar[i].Reading;
angle++;
}

for (i=0;i<6;i++) {
map->Reading[angle] = sonar2.sonar[i].Reading;
angle++;
}
}

/*************************************************************************/

/* Translates the bumper data into MyBumper structure to the Map structure
*/
```

```
void MapBumperSet (struct MyBumper bumper0, struct MyBumper bumper1,
struct MyBumper bumper2, struct Map *map) {

int angle, i;

angle = 0;

for (i=6;i<8;i++) {
map->Reading[angle] = bumper2.bumper[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = bumper0.bumper[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = bumper1.bumper[i].Reading;
angle++;
}

for (i=0;i<6;i++) {
map->Reading[angle] = bumper2.bumper[i].Reading;
angle++;
}
}

/*************************************************************************/

/* Returns the greater value from the MyBumper structures */

void MaxBumperSet (struct MyBumper bumper1, struct MyBumper bumper2,
struct MyBumper *minbumper) {

int i;

for (i=0;i<8;i++) {
minbumper->bumper[i].Reading = bumper1.bumper[i].Reading >
bumper2.bumper[i].Reading?bumper1.bumper[i].Reading:bumper2.bumper[i].Reading;
}
}

/*************************************************************************/

/* Translate all the bumper sets into the Map structure */

void MakeBumperMap (struct Map *map) {

struct MyBumper bs,bi,bumper0,bumper1,bumper2;

ReadBumperSet (0,&bs);
ReadBumperSet (1,&bi);
MaxBumperSet (bs,bi,&bumper0);
ReadBumperSet (2,&bs);
ReadBumperSet (3,&bi);
```

```
MaxBumperSet (bs,bi,&bumper1);
ReadBumperSet (4,&bs);
ReadBumperSet (5,&bi);
MaxBumperSet (bs,bi,&bumper2);

MapBumperSet (bumper0,bumper1,bumper2,map);
}

/*****************************************************************************/

/* Translate all the sonar sets into the Map structure */

void MakeSonarMap (struct Map *map) {

struct MySonar ss,si,sonar0,sonar1,sonar2;

ReadSonarSet (0,&ss);
ReadSonarSet (1,&si);
MinSonarSet (ss,si,&sonar0);
ReadSonarSet (2,&ss);
ReadSonarSet (3,&si);
MinSonarSet (ss,si,&sonar1);
ReadSonarSet (4,&ss);
ReadSonarSet (5,&si);
MinSonarSet (ss,si,&sonar2);

MapSonarSet (sonar0,sonar1,sonar2,map);
}

/*****************************************************************************/

/* Maps a provided angle to the Map structure's index */

int MapAngle (int angle) {

int a;
a = ceil (0.06969*angle-1.04545);
a = a < 0 ? 24+a : a;
return (a);
}

/*****************************************************************************/

/* Gets the value of sonar data in a specified direction */

int GetSMapReading (struct Map map, int angle, int n_sensors) {

int s, i, a, n, t;

n = n_sensors / 2;
a = MapAngle (angle);
s = 10000; /* max value to sonar data */
for (i=0;i<2*n+1;i++) {
t = a-n+i;
t = t < 0 ? 24-t : t;
t = t > 23 ?  t-24 : t;
```

```
s = map.Reading[t]<s ? map.Reading[t] : s;
}
return (s);
}

/*****************************************************************************/

/* Gets the value of IR data in a specified direction */

int GetIMapReading (struct Map map, int angle, int n_sensors) {

int s, i, a, n, t;

n = n_sensors / 2;
a = MapAngle (angle);
s = 0; /* min value for IR data */
for (i=0;i<2*n+1;i++) {
t = a-n+i;
t = t < 0 ? 24-t : t;
t = t > 23 ?  t-24 : t;
s = map.Reading[t]>s ? map.Reading[t] : s;
}
return (s);
}

/*****************************************************************************/

/* Gets the value of Bumper data in a specified direction */

int GetBMapReading (struct Map map, int angle, int n_sensors) {

int s, i, a, n, t;

n = n_sensors / 2;
a = MapAngle (angle);
s = N_BUMPER_NONE; /* min value for bumper data */
for (i=0;i<2*n+1;i++) {
t = a-n+i;
t = t < 0 ? 24-t : t;
t = t > 23 ?  t-24 : t;
s = map.Reading[t]>s ? map.Reading[t] : s;
}
return (s);
}

/*****************************************************************************/

/* Returns the greater value from the MyIR structures */

void MaxIRSet (struct MyIR IR1, struct MyIR IR2,
struct MyIR *minIR) {

int i;

for (i=0;i<8;i++) {
minIR->IR[i].Reading = IR1.IR[i].Reading >
```

```
IR2.IR[i].Reading?IR1.IR[i].Reading:IR2.IR[i].Reading;
}
}

/*************************************************************************/

/* Translates the IR data into the MyIR structure */

void ReadIRSet (int ss_number, struct MyIR *IRset) {

struct N_RobotState *state;
struct N_Infrared *IR;
int i;

state = N_GetRobotState (1);
IR = state->InfraredController.InfraredSet[ss_number].Infrared;
N_GetInfrared (1);

for (i=0;i<8;i++)
IRset->IR[i].Reading=IR[i].Reading;
}

/*************************************************************************/

/* Translates MyIR structures into the Map structure */

void MapIRSet (struct MyIR IR0, struct MyIR IR1,
struct MyIR IR2, struct Map *map) {

int angle, i;

angle = 0;

for (i=6;i<8;i++) {
map->Reading[angle] = IR2.IR[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = IR0.IR[i].Reading;
angle++;
}

for (i=0;i<8;i++) {
map->Reading[angle] = IR1.IR[i].Reading;
angle++;
}

for (i=0;i<6;i++) {
map->Reading[angle] = IR2.IR[i].Reading;
angle++;
}
}

/*************************************************************************/
```

```
/* Translates all the IR data into the Map structure */

void MakeIRMap (struct Map *map) {

struct MyIR ss,si,IR0,IR1,IR2;

ReadIRSet (0,&ss);
ReadIRSet (1,&si);
MaxIRSet (ss,si,&IR0);
ReadIRSet (2,&ss);
ReadIRSet (3,&si);
MaxIRSet (ss,si,&IR1);
ReadIRSet (4,&ss);
ReadIRSet (5,&si);
MaxIRSet (ss,si,&IR2);

MapIRSet (IR0,IR1,IR2,map);

}
```

.1.3 StopServer

```
/*****************************************************************************
STOPSERVER.C - Robot's stop server through the INETD main server
 Sergio Roberto Gonsalves Tourino - tourino@graco.unb.br - 10/05/2000

Before using this program, edit the 'inetd.conf' and the 'services' files:

INETD.CONF:
 stopserver  stream  tcp     nowait  root /home/tourino/stopserver stopserver

SERVICES:
 stopserver  8084/tcp

After these modifications, restart the inetd server:
 kill -1 <PID of inetd - see with ps>
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>

#define TRUE (1==1)
#define FALSE   (1==0)

/*****************************************************************************/

void goodbye (void) {
int result;

     result = system ("rm /home/tourino/stopserver.lock");
}

/*****************************************************************************/
```

```
int main (void) {

    const char delimiters[] = "=, ";    /* delimiters for the commands */
    char command[] = "direction=front, distance=500, nothing important, direction=left, goodbye";
    char *token, parameters[10], commandline[100];
    char direction[10];                 /* direction for the movement */

    char sensors[3]; /* state of sensors */
    char get[10];

    int open_connection = TRUE;
    int result, pid;
    char spid[5];
    int dir, distance = 500, speed=100, teta=524, omega=314; /* movement configuration */

    FILE *lock, *robotserver;

    /* verify the lock file */

    lock = fopen ("/home/tourino/stopserver.lock","r");

    if (lock == NULL) {                 /* the lock does not exist */
        lock = fopen ("/home/tourino/stopserver.lock","w+");
        fclose (lock);
    }
    else {                              /* the lock exists */
        fclose (lock);
        fprintf (stderr,"Lock file found. Program aborted.\n");
        exit (0);                       /* closes the program */
    }

    atexit (goodbye); /* defines the 'goodbye' function to be
called at program termination */

    while (open_connection) {           /* loop until 'open_connection' is false */

        fscanf (stdin,"%s",command);    /* reads commands from stdin */

        token = strtok(command,delimiters);  /* strtok parses the commands from 'command' variable */

        while ((token != NULL) && open_connection) {

    if (strcmp(token,"stop") == 0) {
strcpy (get,strtok(NULL,delimiters));
if (strcmp (get,"on") == 0) {
printf ("stop=on!\n");
robotserver = fopen ("/home/tourino/robserver.pid","r");
   fscanf (robotserver,"%s",spid);
   fclose (robotserver);
   pid = atoi (spid);
printf ("Sending signal to the %d process.\n",pid);
   kill (pid,SIGUSR1);   /* send the signal to the server */
}
    }

            if (strcmp(token,"goodbye") == 0) {
```

```
                open_connection = FALSE;  /* terminates all of the loops */
            }

            token = strtok (NULL,delimiters);
        }
    }

    fprintf (stderr,"Goodbye. Connection closed.\n");

    return 0;
}
```

.1.4 CamServer

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define FALSE   (0==1)
#define TRUE    (0==0)
#define VERT_CHANGE 400
#define HORI_CHANGE 1000
#define MAX_X 11000
#define MIN_X -11000
#define MAX_Y 2400
#define MIN_Y -3600

/*****************************************************************************
CAMSERVER.C - Robot's camera server through the INETD main server
 Sergio Roberto Gonsalves Tourino - tourino@graco.unb.br - 11/02/2000

Before using this program, edit the 'inetd.conf' and the 'services' files:

INETD.CONF:
 camserver  stream  tcp     nowait  root /home/tourino/projeto/socket/servnet servnet

SERVICES:
 camserver  8086/tcp

After these modifications, restart the inetd server:
 kill -1 <PID do inetd - ver com ps>
*****************************************************************************/

int main (void) {

    const char delimiters[] = "= ,";    /* delimiters for the commands */
    char command[] = "direction=front, distance=500, nothing important, direction=left, goodbye";
    char *token, parameters[10], commandline[100];
    char direction[10];                 /* direction for the movement */
    char xabsolute[5], yabsolute[5]; /* absolute values for movement */
    char xchange[4], ychange[4]; /* change values for movement */
    int open_connection = TRUE;
    int x = 0,y = 0;
    int result, msg_id;
    int vert_change = VERT_CHANGE, hori_change = HORI_CHANGE;
```

```
    FILE *connection, *lock;

    /* verify the lock file */

    lock = fopen ("/home/tourino/camserver.lock","r");

    if (lock == NULL) {  /* the lock does not exist */
lock = fopen ("/home/tourino/camserver.lock","w+");
fclose (lock);
    }
    else { /* the lock exists */
fclose (lock);
fprintf (stderr,"Lock file found. Program aborted.\n");
exit (0); /* closes the program */
    }

    connection = fopen ("/home/tourino/camserver.out","w+"); /* opening file */
    msg_id = init_message(); /* starting IPC */

    while (open_connection) {           /* loop until 'open_connection' is false */

        fscanf (stdin,"%s",command);    /* reads commands from stdin */
fprintf (connection,"%s\n",command); /* writes to file */
fflush (connection); /* it's better... */

        token = strtok(command,delimiters);  /* strtok parses the commands from 'command' variable */

        while ((token != NULL) && open_connection) {

/*     fprintf (stdout,"%s\n",token); /* echoing back */
/*     fflush (stdout); /* it's better... */

    if (strcmp(token,"framerate") == 0) {
send_message (msg_id,1,strtok(NULL,delimiters));
    }

           if (strcmp(token,"direction") == 0) {
               strcpy (direction,strtok(NULL,delimiters));
/*             fprintf (stderr,"I will walk to %s.\n",direction); */
           }

    if (strcmp(token,"xabsolute") == 0) {
strcpy (xabsolute,strtok(NULL,delimiters));
/* fprintf (stderr,"Going to xabsolute=%s\n",xabsolute); */
    }

    if (strcmp(token,"yabsolute") == 0) {
strcpy (yabsolute,strtok(NULL,delimiters));
/* fprintf (stderr,"Going to yabsolute=%s\n",yabsolute); */
    }

    if (strcmp(token,"xchange") == 0) {
strcpy (xchange,strtok(NULL,delimiters));
/* fprintf (stderr,"Setup: xchange=%s\n",xchange); */
if (strlen(xchange) > 0)
hori_change = atoi (xchange);
```

```
    }

    if (strcmp(token,"ychange") == 0) {
strcpy (ychange,strtok(NULL,delimiters));
/* fprintf (stderr,"Setup ychange=%s\n",ychange); */
if (strlen(ychange) > 0)
vert_change = atoi (ychange);
    }

            if (strcmp(token,"start") == 0) {
                /* Here comes the commands for the control of the robot.
                 * This command MUST only work with all the variables (speed, acceleration, ...)
                 * adjusted accordingly. */

if (strcmp (direction,"zero") == 0) {
x = 0;
y = 0;
}

if (strcmp (direction,"center") == 0) {
x = 0;
y = 0;
}

if (strcmp (direction,"up") == 0)
if (y+vert_change <= MAX_Y)
y += vert_change;
if (strcmp (direction,"down") == 0)
if (y-vert_change >= MIN_Y)
y -= vert_change;

if (strcmp (direction,"left") == 0)
if (x+hori_change >= MIN_X)
x += hori_change;
if (strcmp (direction,"right") == 0)
if (x-hori_change <= MAX_X)
x -= HORI_CHANGE;

if (strcmp (direction,"upleft") == 0) {
if (x+hori_change >= MIN_X)
x += hori_change;
if (y+vert_change <= MAX_Y)
y += vert_change;
}

if (strcmp (direction,"downleft") == 0) {
if (x+hori_change >= MIN_X)
x += hori_change;
if (y-vert_change >= MIN_Y)
y -= vert_change;
}

if (strcmp (direction,"upright") == 0) {
if (x-hori_change <= MAX_X)
x -= hori_change;
if (y+vert_change <= MAX_Y)
```

```
y += vert_change;
}

if (strcmp (direction,"downright") == 0) {
if (x-hori_change <= MAX_X)
x -= hori_change;
if (y-vert_change >= MIN_Y)
y -= vert_change;
}

if ((strlen (xabsolute) > 0) && (strlen (yabsolute) > 0)) {
if ((atoi(xabsolute) > MIN_X) && (atoi(xabsolute)
< MAX_X) && (atoi(yabsolute) > MIN_Y) &&
(atoi(yabsolute) < MAX_Y)) {
sprintf (parameters,"%s %s",xabsolute,yabsolute);
x = atoi (xabsolute);
y = atoi (yabsolute);
}
else {
sprintf (xabsolute,"%d",x);
sprintf (yabsolute,"%d",y);
strcpy (parameters,xabsolute);
strcat (parameters," ");
strcat (parameters,yabsolute);
}
}
else
sprintf (parameters,"%d %d",x,y);
strcpy (commandline,"/usr/local/robot-devices/dp_pantilt/pt ");
strcat (commandline,parameters);
/* fprintf (stderr,"%s\n",commandline); */
result = system (commandline);
strcpy (xabsolute,""); /* erasing the old values... */
strcpy (yabsolute,"");
strcpy (direction,"");
           }

           if (strcmp(token,"goodbye") == 0) {
               open_connection = FALSE;  /* terminates all of the loops */
           }

           token = strtok (NULL,delimiters);
       }
   }

/*    fprintf (stderr,"Goodbye. Connection closed.\n"); */

   fclose (connection);
   remove_queue (msg_id);
   result = system ("rm /home/tourino/camserver.lock"); /* erases lock file */

   return 0;
}
```

.1.5 JPGStd

```
#include <stdio.h>/* printf */
#include <errno.h>/* errno */
#include <stdlib.h>/* malloc */
#include <unistd.h>/* sleep */
#include <sys/mman.h>/* MAP_FILE */
#include <sys/fcntl.h>/* open */
#include <jpeglib.h>            /* IJG routines */

#include "ioctl_meteor.h"

#define VIDEO_DEV       "/dev/meteor0"

#define COLS 320 /* # of pixels in a row in output image */
#define ROWS 240 /* # of lines in output image */
#define SIZE (ROWS * COLS * 4) /* 4 bytes/pixel for METEOR_GEO_RGB24 */

static JSAMPLE * image_buffer;  /* Points to large array of R,G,B-order data */
static int image_height;        /* Number of rows in image */
static int image_width;         /* Number of columns in image */

void write_JPEG_file (int quality) {
  struct jpeg_compress_struct cinfo;
  struct jpeg_error_mgr jerr;
  JSAMPROW row_pointer[1];      /* pointer to JSAMPLE row[s] */
  int row_stride;               /* physical row width in image buffer */

  cinfo.err = jpeg_std_error(&jerr);
  jpeg_create_compress(&cinfo);

  jpeg_stdio_dest(&cinfo, stdout);   /* writes compressed image to stdout */

  cinfo.image_width = image_width;      /* image width and height, in pixels */
  cinfo.image_height = image_height;
  cinfo.input_components = 3;           /* # of color components per pixel */
  cinfo.in_color_space = JCS_RGB;       /* colorspace of input image */

  jpeg_set_defaults(&cinfo);

  jpeg_set_quality(&cinfo, quality, TRUE /* limit to baseline-JPEG values */);

  jpeg_start_compress(&cinfo, TRUE);

  row_stride = image_width * 3; /* JSAMPLEs per row in image_buffer */

  while (cinfo.next_scanline < cinfo.image_height) {
    row_pointer[0] = &image_buffer[cinfo.next_scanline * row_stride];
    (void) jpeg_write_scanlines(&cinfo, row_pointer, 1);
  }

  jpeg_finish_compress(&cinfo);

  jpeg_destroy_compress(&cinfo);
}

int main (int argc, char **argv)
```

```
{
  struct meteor_counts cnt;
  struct meteor_geomet geo;
  char *mmbuf;   /* memory map buffer */
  char *imdata, *buf;
  int ifd, c, x, y;

  if ((ifd = open(VIDEO_DEV, O_RDONLY)) < 0) {
    perror (VIDEO_DEV);
    exit (1);
  }

  geo.rows = ROWS; /* # of lines in output image */
  geo.columns = COLS; /* # of pixels in a row in output image */
  geo.frames = 1; /* # of frames in a buffer */
  geo.oformat = METEOR_GEO_RGB24 ; /* RGB 24 in 4 bytes: B,G,R,NULL */
  if (ioctl (ifd, METEORSETGEO, &geo) < 0) {
    perror ("ioctl SetGeometry failed");
    exit (1);
  }

  c = METEOR_FMT_NTSC;
  if (ioctl (ifd, METEORSFMT, &c) < 0) {
    perror ("ioctl SetFormat failed");
    exit (1);
  }

  c = METEOR_INPUT_DEV0; /* composite video RCA jack */
  if (ioctl (ifd, METEORSINPUT, &c) < 0) {
    perror ("ioctl Setinput failed");
    exit (1);
  }

  sleep (1);

  if (0 > (mmbuf = (char *) mmap((caddr_t) 0, SIZE, PROT_READ,
 MAP_FILE | MAP_PRIVATE, ifd, (off_t) 0))) {
    perror ("mmap failed");
    exit (1);
  }

  c = METEOR_CAP_SINGLE ;
  if (ioctl (ifd, METEORCAPTUR, &c)) {
    perror ("ioctl SingleCapture failed");
    exit (1);
  }

  if (ioctl (ifd, METEORGCOUNT, &cnt)) {
      perror ("ioctl GetCount failed");
      exit (1);
  }

  close (ifd);

  image_width = COLS;
  image_height = ROWS;
```

```
  if (NULL == (image_buffer = (JSAMPLE *)malloc(image_width*image_height*3))) {
    perror ("malloc");
    exit (1);
  }

  buf = mmbuf; /* start of frame grabber data (B,G,R,NULL) */
  imdata = (char *) image_buffer; /* start of JPEG buffer (R,G,B) */
  for (y=0; y<ROWS; y++) {
    for (x=0; x<COLS; x++) {
      *(imdata++) = *(buf+2); /* R */
      *(imdata++) = *(buf+1); /* G */
      *(imdata++) = *(buf); /* B */
      buf += 4; /* skip NULL */
    }
  }

  printf ("HTTP/1.0 200 Okay\n");
  printf ("Content-type: image/jpeg\n\n");
  write_JPEG_file (30);

  return (0);
}
```

.1.6 JPGPush

```
#include <stdio.h>/* printf */
#include <errno.h>/* errno */
#include <stdlib.h>/* malloc */
#include <unistd.h>/* sleep */
#include <sys/mman.h>/* MAP_FILE */
#include <sys/fcntl.h>/* open */
#include <jpeglib.h>            /* IJG routines */
#include <sys/resource.h>/* setpriority */
#include <signal.h>/* getpid */

#include <sys/signal.h>
#include <sys/time.h>

#include "ioctl_meteor.h"

#define VIDEO_DEV       "/dev/meteor0"

#define COLS 240 /* # of pixels in a row in output image */
#define ROWS 180 /* # of lines in output image */
#define SIZE (ROWS * COLS * 4) /* 4 bytes/pixel for METEOR_GEO_RGB24 */

#define PROG_PRIO -10 /* priority for the server */

static JSAMPLE * image_buffer;  /* Points to large array of R,G,B-order data */
static int image_height;        /* Number of rows in image */
static int image_width;         /* Number of columns in image */

char *mmbuf;    /* memory map buffer */
char *data;     /* image data buffer */
int fc=0;       /* frame counter */
int size = SIZE;      /* image size */
```

```
int im_ready;  /* image ready in data buffer */
int framerate=10;

/* function prototypes */
static void gotframe(int x);

/* signal structure */
static struct sigaction sigact = { gotframe,
                                   0,
                                   SA_RESTART,
                                   NULL };

static struct sigaction oldact;

void gotframe(int sig)
{
  fc++;

  if (fc%framerate==0) {
    memcpy(data,mmbuf,size);
    im_ready=1;
  }

  if ((oldact.sa_handler)!=NULL) {
    (oldact.sa_handler)(sig);
  }
}

void write_JPEG_file (int quality)
{
  struct jpeg_compress_struct cinfo;
  struct jpeg_error_mgr jerr;
  JSAMPROW row_pointer[1];      /* pointer to JSAMPLE row[s] */
  int row_stride;               /* physical row width in image buffer */

  cinfo.err = jpeg_std_error(&jerr);
  jpeg_create_compress(&cinfo);

  jpeg_stdio_dest(&cinfo, stdout);   /* writes compressed image to stdout */

  cinfo.image_width = image_width;      /* image width and height, in pixels */
  cinfo.image_height = image_height;
  cinfo.input_components = 3;           /* # of color components per pixel */
  cinfo.in_color_space = JCS_RGB;       /* colorspace of input image */

  jpeg_set_defaults(&cinfo);

  jpeg_set_quality(&cinfo, quality, TRUE /* limit to baseline-JPEG values */);

  jpeg_start_compress(&cinfo, TRUE);

  row_stride = image_width * 3; /* JSAMPLEs per row in image_buffer */

  while (cinfo.next_scanline < cinfo.image_height) {
    row_pointer[0] = &image_buffer[cinfo.next_scanline * row_stride];
    (void) jpeg_write_scanlines(&cinfo, row_pointer, 1);
```

```
  }

  jpeg_finish_compress(&cinfo);

  jpeg_destroy_compress(&cinfo);
}

void write_JPEG_Image (void)
{
  struct meteor_counts cnt;
  struct meteor_geomet geo;
  char *imdata, *buf;
  int ifd, c, x, y, sigmode = 0;

  struct timeval t;
  long t0,t1;
  int ic=0;  /* image file counter */
  char nf[20];

  FILE *time_cap;

  time_cap = fopen ("/home/tourino/timecap.out","w");

  if ((ifd = open(VIDEO_DEV, O_RDONLY)) < 0) {
    perror (VIDEO_DEV);
    exit (1);
  }

  geo.rows = ROWS; /* # of lines in output image */
  geo.columns = COLS; /* # of pixels in a row in output image */
  geo.frames = 1; /* # of frames in a buffer */
  geo.oformat = METEOR_GEO_RGB24 ; /* RGB 24 in 4 bytes: B,G,R,NULL */

  sigmode = METEOR_SIG_FRAME;

  if (ioctl (ifd, METEORSETGEO, &geo) < 0) {
    perror ("ioctl SetGeometry failed");
    exit (1);
  }

  c = METEOR_FMT_NTSC;
  if (ioctl (ifd, METEORSFMT, &c) < 0) {
    perror ("ioctl SetFormat failed");
    exit (1);
  }

  c = METEOR_INPUT_DEV0; /* composite video RCA jack */
  if (ioctl (ifd, METEORSINPUT, &c) < 0) {
    perror ("ioctl Setinput failed");
    exit (1);
  }

  sleep (1);

  if ( sigaction(SIGUSR2, &sigact, &oldact) ) {
    perror("sigaction failed");
```

```
    exit(1);
  }

  c = SIGUSR2 | sigmode;
  if (ioctl(ifd, METEORSSIGNAL, &c) < 0) {
    perror("ioctl SetSignal failed");
    exit(1);
  }

  data=(char *)malloc(size);

  mmbuf=(char *)mmap((caddr_t)0, size, PROT_READ,
                     MAP_FILE|MAP_PRIVATE, ifd, (off_t)0);
  if ( mmbuf == (char *)-1 ) {
    perror("mmap failed");
    exit(1);
  }

  /* capture continuously */

  c = METEOR_CAP_CONTINOUS;
  if (ioctl(ifd, METEORCAPTUR, &c)) {
    perror("ioctl SingleCapture failed");
    exit(1);
  }

  /* M A I N   L O O P */

  gettimeofday(&t, NULL);
  t0 = t.tv_sec*1000 + t.tv_usec/1000;
  t1 = t0;

  msg_id = init_message(); /* starting IPC */

  while (t1-t0<600000)    /* Timeout = 10 min. */
    {
      msg = read_message (msg_id,1);

      if (msg != NULL)
framerate = atol (msg);       /* frame rate setup */

      // usleep(100);
      gettimeofday(&t, NULL);
      t1 = t.tv_sec*1000 + t.tv_usec/1000;

      if (im_ready) {

fprintf (time_cap,"%ld\n",t1);
fflush (time_cap);

printf ("\n--JPEGImage\nContent-type: image/jpeg\n\n");

image_width = COLS;
image_height = ROWS;
   if (NULL == (image_buffer = (JSAMPLE *)malloc(image_width*image_height*3))) {
     perror ("malloc");
```

```
    exit (1);
  }

  buf = mmbuf; /* start of frame grabber data (B,G,R,NULL) */
  imdata = (char *) image_buffer; /* start of JPEG buffer (R,G,B) */
  for (y=0; y<ROWS; y++) {
    for (x=0; x<COLS; x++) {
      *(imdata++) = *(buf+2); /* R */
      *(imdata++) = *(buf+1); /* G */
      *(imdata++) = *(buf); /* B */
      buf += 4; /* skip NULL */
    }
  }

  write_JPEG_file (30);
fflush (stdout);

       im_ready=0;
     }

     gettimeofday(&t, NULL);
     t1 = t.tv_sec*1000 + t.tv_usec/1000;
 }

 c = METEOR_CAP_STOP_CONT;
 if (ioctl(ifd, METEORCAPTUR, &c)) {
   perror("ioctl SingleCapture2 failed");
   exit(1);
 }

 close(ifd);
 fclose (time_cap);
}

int main (int argc, char **argv)
{
int pid;
FILE *mypid;

if (setpriority (PRIO_PROCESS,0,PROG_PRIO) != 0)
fprintf (stderr,"Error in setpriority!\n");

pid = getpid (); /* getting the pid of program */

/* saving pid */
mypid = fopen ("/home/tourino/push.pid","w+");
fprintf (mypid,"%d",pid);
fclose (mypid);

printf ("HTTP/1.0 200 Okay\n");
printf ("Content-type: multipart/x-mixed-replace;boundary=JPEGImage\n\n");
write_JPEG_Image ();
}
```

.2 Clientes em Linguagem Java

.2.1 ImageControl

```
// <APPLET CODE=ImageControl.class width=400 height=200></APPLET>

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.net.*;
import java.io.*;
import java.lang.*;

// Main class for control of the camera of the XR4000 robot

public class ImageControl extends Applet {

static String[]
  stringButtons = {"UL","Up","UR","Left","Center","Right","DL","Down","DR","Move","Change"},
  initSetup     = {"10","5","0","0"};

static int[]
  maxLimits = {140, 38, 140, 30},
  minLimits = {2, 2, -140, -46};

Panel
  controlPanel,
  canvasPanel,
  centerPanel;

Button[]
  controlButtons = new Button[11];

Label
  controlLabel;

URL
  server;

SocketIO
  connection;

TextField[]
  setupText = new TextField[4];

PanCanvas
  panCanvas;

TiltCanvas
  tiltCanvas;

int
  pan = 0,
  tilt = 0,// absolute location of the camera
  panchange = 10,
  tiltchange = 5; // change values for the pantilt
```

```
boolean
  inUse = false;

// Init method for the applet. Initializes the applet and creates the panel and the
// buttons for its use.

public void init() {

  controlLabel = new Label ("Connecting to server...");

  try {
   connection = new SocketIO (getCodeBase().getHost(),8086);
// opens the connection with the camera's remote server
controlLabel.setText ("Server connected.");
  }
  catch (UnknownHostException e) {
System.err.println ("Error in URL of camera server.");
controlLabel.setText ("Error in URL of camera server.");
inUse = true;
  }
  catch (IOException e) {
System.err.println ("Error connecting camera server.");
controlLabel.setText ("Error connecting camera server.");
inUse = true;
  }

  controlPanel = new Panel (new GridLayout (7,3));

  panCanvas = new PanCanvas (pan,10,15,5,10,25,5,40,30);
  tiltCanvas = new TiltCanvas (tilt,15,10,5,5,15,30);

  Dimension sizeCanvas = new Dimension (100,100);
  panCanvas.setSize (sizeCanvas);
  tiltCanvas.setSize (sizeCanvas);
  panCanvas.setScale ((float) 0.5);
  tiltCanvas.setScale ((float) 0.5);

  int i;

  for (i=0;i<4;i++) { // creates all textfields
setupText[i] = new TextField (initSetup[i]);
setupText[i].addTextListener (new EventText (i));
  }

  for (i=0;i<9;i++) {                  // creates all buttons
    controlButtons[i]=new Button (stringButtons[i]);
    controlButtons[i].addActionListener (new EventButton (i));
    controlPanel.add (controlButtons[i]);
  }

  controlButtons[9]= new Button (stringButtons[9]);
  controlButtons[9].addActionListener (new EventButton (9));
  controlButtons[10]= new Button (stringButtons[10]);
  controlButtons[10].addActionListener (new EventButton (10));
```

```
  controlPanel.add (new Label (""));
  controlPanel.add (new Label ("Absolute"));
  controlPanel.add (new Label ("Change"));
  controlPanel.add (new Label ("Pan"));
  controlPanel.add (setupText[2]);
  controlPanel.add (setupText[0]);
  controlPanel.add (new Label ("Tilt"));
  controlPanel.add (setupText[3]);
  controlPanel.add (setupText[1]);
  controlPanel.add (new Label (""));
  controlPanel.add (controlButtons[9]);
  controlPanel.add (controlButtons[10]);

  canvasPanel = new Panel (new GridLayout (2,1));
  canvasPanel.add (panCanvas);
  canvasPanel.add (tiltCanvas);

  centerPanel = new Panel (new GridLayout (1,2));
  centerPanel.add (controlPanel);
  centerPanel.add (canvasPanel);

  setLayout (new BorderLayout ());
  add (centerPanel,BorderLayout.CENTER);
  add (controlLabel,BorderLayout.SOUTH);

}

// Destroy method for the applet. Stops the socket connection with the servers.

public void destroy () {
  try {
connection.send("direction=center, start, goodbye"); // closing the remote server
   connection.close();
  } catch (IOException e) {
System.err.println ("Error closing connection with camera server.");
controlLabel.setText ("Error closing connection.");
  }
}

// Class used to catch the events created by the buttons.

class EventButton implements ActionListener {

int index;

  EventButton (int index) {
this.index = index;
  }

  public void actionPerformed (ActionEvent e) {
String command, pans, tilts;
int i;

switch (index) {
case 0:
case 3:
```

```
case 6:
pan += panchange;
break;
case 2:
case 5:
case 8:
pan -= panchange;
break;
}
switch (index) {
case 0:
case 1:
case 2:
tilt += tiltchange;
break;
case 6:
case 7:
case 8:
tilt -= tiltchange;
break;
case 4:
pan = 0;
tilt= 0;
break;
case 9:
pan = Integer.parseInt (setupText[2].getText());
tilt = Integer.parseInt (setupText[3].getText());
break;
case 10:
panchange = Integer.parseInt (setupText[0].getText());
tiltchange = Integer.parseInt (setupText[1].getText());
break;
}

if (index < 9) {
setupText[2].setText(pan+"");
setupText[3].setText(tilt+"");
}

if (pan < minLimits[2])
pan = minLimits[2];
if (pan > maxLimits[2])
pan = maxLimits[2];
if (tilt < minLimits[3])
tilt = minLimits[3];
if (tilt > maxLimits[3])
tilt = maxLimits[3];

panCanvas.setAngle (pan);
tiltCanvas.setAngle (tilt);

pans = Math.round (pan/0.012857) + "";
tilts = Math.round (tilt/0.012857) + "";

command = "xabsolute="+pans+",yabsolute="+tilts+", start";
```

```
       try {
         connection.send (command);
       }
       catch(Exception exception) {
controlLabel.setText ("Error: Server in use.");
inUse = true;
exception.printStackTrace();
      }

if (inUse) {
for (i=0;i<11;i++)
controlButtons[i].setEnabled (false);
}
  }
}

class EventText implements TextListener {

int index;

EventText (int index) {
this.index = index;
}

public void textValueChanged (TextEvent e) {

int textvalue = Integer.parseInt (setupText[index].getText());

if (textvalue > maxLimits[index])
setupText[index].setText(maxLimits[index]+"");
if (textvalue < minLimits[index])
setupText[index].setText(minLimits[index]+"");
}
}

// Canvas class

public class PanCanvas extends Canvas {

int teta, dx, dy, bc, hc, bs, hs, r; // robot's dimensions
float scale,phi;
Dimension size;

public PanCanvas (int teta, int dx, int dy, int bc, int hc, int bs, int hs, int r, int  phi) {

this.teta = teta;
this.dx = dx;
this.dy = dy;
this.bc = bc;
this.hc = hc;
this.bs = bs;
this.hs = hs;
this.r = r;
this.phi = (float) MyMath.toRadians((double) phi);
scale = (float) 1.0;
size = new Dimension (200,200);
```

```
repaint ();
}

public void setAngle (int teta) {

this.teta = teta;
repaint();
}

public void setScale (float s) {
this.scale = s;
repaint();
}

public void setSize (Dimension d) {
size = d;
}

Point fromCamera (Point P) {

Point n = new Point ();
double trad;

trad = MyMath.toRadians ((double) teta);

n.x = (int) (Math.cos(trad)*P.x - Math.sin(trad)*P.y - Math.cos(trad)*dx);
n.y = (int) (Math.sin(trad)*P.x + Math.cos(trad)*P.y - Math.sin(trad)*dx + dy);

n.x = (int) (n.x * scale);
n.y = (int) (n.y * scale);

return n;
}

Point fromPanTilt (Point P) {

Point n = new Point ();
double trad;

trad = MyMath.toRadians ((double) teta);

n.x = (int) (Math.cos(trad)*P.x - Math.sin(trad)*P.y);
n.y = (int) (Math.sin(trad)*P.x + Math.cos(trad)*P.y + dy);

n.x = (int) (n.x * scale);
n.y = (int) (n.y * scale);

return n;
}


public Dimension getPreferredSize() {

        return getMinimumSize();
    }
```

```
public Dimension getMinimumSize() {

        return size;
    }


public void paint (Graphics g) {

Dimension s;
Point A,B,C,D,E,F,G,H,M,N,O,P;
Polygon camera, pantilt;

camera = new Polygon();
pantilt = new Polygon();

s = getSize();

g.clearRect (0,0,s.width,s.height);
g.setColor (Color.black);
g.drawOval ((int)(s.width/2 - r*scale),(int)(s.height/2 - r*scale),
(int)(2*r*scale),(int)(2*r*scale)); // robot

A = new Point (fromCamera (new Point (-bc/2, hc/2)));
B = new Point (fromCamera (new Point (bc/2, hc/2)));
C = new Point (fromCamera (new Point (bc/2, -hc/2)));
D = new Point (fromCamera (new Point (-bc/2, -hc/2)));

E = new Point (fromPanTilt (new Point (-bs/2, hs/2)));
F = new Point (fromPanTilt (new Point (bs/2, hs/2)));
G = new Point (fromPanTilt (new Point (bs/2, -hs/2)));
H = new Point (fromPanTilt (new Point (-bs/2, -hs/2)));

M = new Point (fromCamera (new Point (0, hc/2)));
N = new Point (fromCamera (new Point ((int)(-10*hc*Math.sin(phi)),  (int)(hc/2+10*hc*Math.cos(phi)) )));
O = new Point (fromCamera (new Point (0,21*hc/2)));
P = new Point (fromCamera (new Point
((int)(10*hc*Math.sin(phi)),(int)(hc/2+10*hc*Math.cos(phi)) )));

camera.addPoint (s.width/2 + A.x, -A.y + s.height/2);
camera.addPoint (s.width/2 + B.x, -B.y + s.height/2);
camera.addPoint (s.width/2 + C.x, -C.y + s.height/2);
camera.addPoint (s.width/2 + D.x, -D.y + s.height/2);

pantilt.addPoint (s.width/2 + E.x, -E.y + s.height/2);
pantilt.addPoint (s.width/2 + F.x, -F.y + s.height/2);
pantilt.addPoint (s.width/2 + G.x, -G.y + s.height/2);
pantilt.addPoint (s.width/2 + H.x, -H.y + s.height/2);

g.setColor (Color.blue);
g.fillPolygon (pantilt);
g.setColor (Color.green);
g.fillPolygon (camera);

g.setColor (Color.red);
g.drawLine (s.width/2 + M.x,s.height/2 - M.y,s.width/2 + N.x,s.height/2 - N.y);
g.drawLine (s.width/2 + M.x,s.height/2 - M.y,s.width/2 + P.x,s.height/2 - P.y);
```

```
g.setColor (Color.black);
g.drawLine (s.width/2 + M.x,s.height/2 - M.y,s.width/2 + O.x,s.height/2 - O.y);

g.drawRect (0,0,s.width-1,s.height-1);
}
} // end of PanCanvas

public class TiltCanvas extends Canvas {

int teta, dx, dy, bc, hc, bs, hs, r; // robot's dimensions
float scale, phi;
Dimension size;

public TiltCanvas (int teta, int dy, int bc, int hc, int bs, int hs, int phi) {

this.teta = teta;
this.dy = dy;
this.bc = bc;
this.hc = hc;
this.bs = bs;
this.hs = hs;
this.phi = (float) MyMath.toRadians((double) phi);
scale = (float) 1.0;
size = new Dimension (200,200);
repaint ();
}

public void setAngle (int teta) {

this.teta = teta;
repaint();
}

public void setScale (float s) {
this.scale = s;
repaint();
}

public void setSize (Dimension d) {
size = d;
}

Point fromPanTilt (Point P) {

Point n = new Point ();
double trad;

trad = MyMath.toRadians ((double) teta);

n.x = (int) (Math.cos(trad)*P.x - Math.sin(trad)*P.y);
n.y = (int) (Math.sin(trad)*P.x + Math.cos(trad)*P.y + dy);

n.x = (int) (n.x * scale);
n.y = (int) (n.y * scale);

return n;
```

```
}


public Dimension getPreferredSize() {

        return getMinimumSize();
    }

public Dimension getMinimumSize() {

         return size;
    }


public void paint (Graphics g) {

Dimension s;
Point A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P;
Polygon camera, pantilt, base;

base = new Polygon();
camera = new Polygon();
pantilt = new Polygon();

s = getSize();

g.clearRect (0,0,s.width,s.height);
g.setColor (Color.black);

A = new Point ((int)(-bs/2*scale), 0);
B = new Point ((int)(bs/2*scale), 0);
C = new Point ((int)(bs/2*scale),(int) (dy*scale));
D = new Point ((int)(-bs/2*scale),(int) (dy*scale));

E = new Point (fromPanTilt (new Point (-bs/2, 0)));
F = new Point (fromPanTilt (new Point (bs/2, 0)));
G = new Point (fromPanTilt (new Point (bs/2, hs)));
H = new Point (fromPanTilt (new Point (-bs/2, hs)));

I = new Point (fromPanTilt (new Point (-bc/2, hs)));
J = new Point (fromPanTilt (new Point (bc/2, hs)));
K = new Point (fromPanTilt (new Point (bc/2, hs+hc)));
L = new Point (fromPanTilt (new Point (-bc/2, hs+hc)));


M = new Point (fromPanTilt (new Point (bc/2, hs+hc/2)));
N = new Point (fromPanTilt (new Point ((int)(10*bc*Math.cos(phi)), (int)(hs+hc/2+10*bc*Math.sin(phi)) )));
O = new Point (fromPanTilt (new Point (21*bc/2,hs+hc/2)));
P = new Point (fromPanTilt (new Point ((int)(10*bc*Math.cos(phi)),
(int)(hs+hc/2-10*bc*Math.sin(phi)) )));

base.addPoint (s.width/2 + A.x, -A.y + s.height/2);
base.addPoint (s.width/2 + B.x, -B.y + s.height/2);
base.addPoint (s.width/2 + C.x, -C.y + s.height/2);
base.addPoint (s.width/2 + D.x, -D.y + s.height/2);
```

```
pantilt.addPoint (s.width/2 + E.x, -E.y + s.height/2);
pantilt.addPoint (s.width/2 + F.x, -F.y + s.height/2);
pantilt.addPoint (s.width/2 + G.x, -G.y + s.height/2);
pantilt.addPoint (s.width/2 + H.x, -H.y + s.height/2);

camera.addPoint (s.width/2 + I.x, -I.y + s.height/2);
camera.addPoint (s.width/2 + J.x, -J.y + s.height/2);
camera.addPoint (s.width/2 + K.x, -K.y + s.height/2);
camera.addPoint (s.width/2 + L.x, -L.y + s.height/2);

g.setColor (Color.black);
g.fillPolygon (base);
g.setColor (Color.blue);
g.fillPolygon (pantilt);
g.setColor (Color.green);
g.fillPolygon (camera);

g.setColor (Color.red);
g.drawLine (s.width/2 + M.x,s.height/2 - M.y,s.width/2 + N.x,s.height/2 - N.y);
g.drawLine (s.width/2 + M.x,s.height/2 - M.y,s.width/2 + P.x,s.height/2 - P.y);
g.setColor (Color.black);
g.drawLine (s.width/2 + M.x,s.height/2 - M.y,s.width/2 + O.x,s.height/2 - O.y);

g.drawRect (0,0,s.width-1,s.height-1);
}
} // end of TiltCanvas

} // end of applet
```

.2.2 RobotControl

```
// <APPLET CODE=RobotControl.class width=400 height=500></APPLET>

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.net.*;
import java.io.*;

// Main class for control of the XR4000 robot

public class RobotControl extends Applet {

static String[]
  stringButtons =
{"CCW","Front","CW","Left","Stop","Right","","Back","","Change"},
  actionButtons =
{"counterclockwise","front","clockwise","left","stop","right","","back","",""},
  initSetup     = {"100","15","500","30"};

Panel
  controlPanel, sensorPanel, sonarPanel, batteryPanel, mapPanel;

Button[]
  controlButtons = new Button[11];
```

```
Choice
  sensorChoice = new Choice();

Label
  controlLabel, xLabel, yLabel, tLabel;

URL
  server;

SocketIO
  connection, sensorconnection, stopconnection;

GetSensors
  sensors;

SensorCanvas
  sensorCanvas;

BatteryCanvas
  batteryCanvas;

//MapCanvas
//  mapCanvas;

TextField[]
  setupText = new TextField[4];

int[] sonar = new int[24],
      ir = new int[24];

int battery = 13000;

// Init method for the applet. Initializes the applet and creates the panel and the
// buttons for its use.

public void init() {

  controlLabel = new Label ("");
  xLabel = new Label ("X=0");
  yLabel = new Label ("Y=0");
  tLabel = new Label ("Teta=0");

  try {
   connection = new SocketIO (getCodeBase().getHost(),8086); // opens the connection with the camera's remote server
connection.send ("sensors=on, distance=500");
sensorconnection = new SocketIO (getCodeBase().getHost(),8085);
  stopconnection = new SocketIO (getCodeBase().getHost(),8084);
  }
  catch (UnknownHostException e) {
System.err.println ("Error in URL of robot server.");
  }
  catch (IOException e) {
System.err.println ("Error connecting robot server.");
controlLabel.setText ("Error connecting server.");
  }
```

```
  controlPanel = new Panel (new GridLayout (9,3));

  int i;

  for (i=0;i<4;i++) { // creates all textfields
setupText[i] = new TextField (initSetup[i]);
setupText[i].addTextListener (new EventText (i));
  }

  for (i=0;i<9;i++) {                  // creates all buttons
    controlButtons[i]=new Button (stringButtons[i]);
    controlButtons[i].addActionListener (new EventButton (i));
    if ((i != 6) && (i != 8))
controlPanel.add (controlButtons[i]);
    else
controlPanel.add (new Label (""));
  }

  controlButtons[9]= new Button (stringButtons[9]);
  controlButtons[9].addActionListener (new EventButton (9));

  controlPanel.add (new Label (""));
  controlPanel.add (new Label ("Distance"));
  controlPanel.add (new Label ("Speed"));
  controlPanel.add (new Label ("Linear (mm)"));
  controlPanel.add (setupText[2]);
  controlPanel.add (new Label (""));
  controlPanel.add (new Label ("Angular (deg)"));
  controlPanel.add (setupText[3]);
  controlPanel.add (setupText[1]);
  controlPanel.add (new Label (""));
  controlPanel.add (new Label (""));
  controlPanel.add (controlButtons[9]);
  controlPanel.add (new Label ("Odometry"));
  controlPanel.add (new Label (""));
  controlPanel.add (new Label (""));
  controlPanel.add (xLabel);
  controlPanel.add (yLabel);
  controlPanel.add (tLabel);

  setLayout (new BorderLayout (2,2));
  add (controlPanel,BorderLayout.CENTER);
  add (controlLabel,BorderLayout.SOUTH);

  sensors = new GetSensors();

  sensorCanvas = new SensorCanvas (sonar);
  sensorCanvas.setScale ((float) 1.5);
  sensorCanvas.repaint();

//  mapPanel = new Panel (new BorderLayout());
//  mapCanvas = new MapCanvas (sonar,7000,3000,(int) (Math.PI/2*1000));
//  mapPanel.add (new Label (""),BorderLayout.NORTH);
//  mapPanel.add (mapCanvas);

  batteryCanvas = new BatteryCanvas (battery);
```

```
  batteryCanvas.setScale ((float) 1.0);
  batteryCanvas.repaint();

  sensorPanel = new Panel (new GridLayout (3,1));
  sonarPanel = new Panel (new BorderLayout ());
  batteryPanel = new Panel (new BorderLayout ());

  sonarPanel.add (sensorCanvas,BorderLayout.SOUTH);
  sonarPanel.add (sensorChoice,BorderLayout.NORTH);
  batteryPanel.add (batteryCanvas,BorderLayout.SOUTH);

  sensorPanel.add (sonarPanel);
  sensorPanel.add (batteryPanel);
//  sensorPanel.add (mapPanel);

  sensorChoice.addItem ("Sonar");
  sensorChoice.addItem ("Infrared");

  add (sensorPanel,BorderLayout.EAST);
}

// Destroy method for the applet. Stops the socket connection with the servers.

public void destroy () {

  sensors.cancel ();

  try {
connection.send("sensors=off, goodbye"); // closing the remote server
   connection.close();
sensorconnection.send ("goodbye");
sensorconnection.close();
stopconnection.send ("goodbye");
stopconnection.close();
  } catch (IOException e) {
System.err.println ("Error closing connection with robot server.");
  }
}

// Class used to catch the events created by the buttons.

class EventButton implements ActionListener {

int index;

  EventButton (int index) {
this.index = index;
  }

  public void actionPerformed (ActionEvent e) {
String command,teta, omega;

teta = Math.round (Integer.parseInt (setupText[3].getText()) *
Math.PI / 180 * 1000) + "";
omega = Math.round (Integer.parseInt (setupText[1].getText()) *
Math.PI / 180 * 1000) + "";
```

```
if (index < 9)
command = "direction=" + actionButtons[index] + ", start";
else
command =
"distance="+setupText[2].getText()+",teta="+teta+",speed="+setupText[0].getText()+",omega="+omega+",start";

try {
if (index == 4) // stop button
stopconnection.send ("stop=on");
else
        connection.send (command);
}
catch(Exception exception) {
exception.printStackTrace();
controlLabel.setText ("Error sending command.");
}
  }
}

class EventText implements TextListener {

int index;

EventText (int index) {
this.index = index;
}

public void textValueChanged (TextEvent e) {

int textvalue = Integer.parseInt (setupText[index].getText());

}
}

// Threaded class that gets sensor data from server

class GetSensors extends Thread {

String rsensor, rbattery, rodometry;
boolean running = true;
StringToIntArray csensor, cbattery, codometry;
GetStringType tsensor, tbattery, todometry;
int[] batterydata = new int[24];
        int[] odometrydata = new int[24];
int i, lessbattery;
int sleeptime = 500;

GetSensors () {
start();
}

void cancel () {
running = false;
}
```

```
public void run () {

int[] data = new int[24];

while (running) {
try {

if (sensorChoice.getSelectedIndex() == 0) // sonar
sensorconnection.send ("get=sonar");
else
sensorconnection.send ("get=IR");
sleep (sleeptime);
rsensor = sensorconnection.get ();
}
catch (InterruptedException e) { }
catch (IOException e) {
controlLabel.setText ("Server out.");
running = false;
}

tsensor = new GetStringType (rsensor);

if (sensorChoice.getSelectedIndex() == 0) {
csensor = new StringToIntArray
(tsensor.getString ("S"));
sensorCanvas.setLimits (500,2000,5000);
}
else {
csensor = new StringToIntArray
(tsensor.getString ("I"));
sensorCanvas.setLimits (100,180,220);
}

data = csensor.getArray();
for (i = 0;i<24;i++)
if (sensorChoice.getSelectedIndex() == 0)
data[i] *= 100;
else
data[i] = 250 - data[i]*10;

sensorCanvas.update (data);


try {
sensorconnection.send ("get=battery");
sleep (sleeptime);
rbattery = sensorconnection.get ();
}
catch (InterruptedException e) { }
catch (IOException e) {
controlLabel.setText ("Server out.");
running = false;
}
tbattery = new GetStringType (rbattery);
cbattery = new StringToIntArray
(tbattery.getString ("B"));
```

```
batterydata = cbattery.getArray();
lessbattery = 15000;

for (i=0;i<4;i++) {
lessbattery = batterydata[i]*100 <
lessbattery ? batterydata[i]*100 :
lessbattery;
}

batteryCanvas.update (lessbattery);

try {
sensorconnection.send ("get=odometry");
sleep (sleeptime);
rodometry = sensorconnection.get();
}
catch (InterruptedException e) { }
catch (IOException e) {
controlLabel.setText ("Server out.");
running = false;
}
todometry = new GetStringType (rodometry);
codometry = new StringToIntArray
(todometry.getString ("O"));
odometrydata = codometry.getArray();

xLabel.setText ("X="+odometrydata[0]);
yLabel.setText ("Y="+odometrydata[1]);
tLabel.setText ("Teta="+Math.round
(odometrydata[3] * 180 / Math.PI / 1000));

// mapCanvas.update
//(csensor.getArray(),odometrydata[0],
//odometrydata[1], odometrydata[2]);
}
}
}

// Class that transforms string from server to array of integers

public class StringToIntArray {

String s;
int p,q,i;
int[] array;

public StringToIntArray (String S) {
s = S;
array = new int[24];
}

public void putString (String S) {
s = S;
}
```

```
public String getString () {
return s;
}

public int[] getArray () {

p = 0;
q = 0;
i = 0;

while ((q != -1) && (i < 24)) {
p = s.indexOf (' ',p);
q = s.indexOf (' ',p+1);
try {
if (q != -1)
array[i++] = Integer.parseInt (s.substring (p++,q).trim());
}
catch (NumberFormatException e) {
System.err.println (s);
}
}
return array;
}
}

// class that gets from a big string a smaller string separated by a type id

public class GetStringType {

        String s;
        int p,q;

        public GetStringType (String S) {

                s = S;
        }

        public void putString (String S) {

                s = S;
        }

        public String getString (String type) {

String r;

                p = s.indexOf (type,0) + 1;
                q = s.indexOf (type,p);

if ((p > 0) && (q > p))
                 r = " "+s.substring (p,q).trim()+" ";
else
r = new String (" ");
return r;
        }
} // end of GetStringType
```

```
// Sensor canvas class

public class SensorCanvas extends Canvas {

int close, medium, far, r;
float scale;
Dimension size;
int[] sensordata;

public SensorCanvas (int[] sensordata) {
this.sensordata = new int[24];

this.sensordata = sensordata;

close = 500;
medium = 2000;
far = 5000;

r = 25;
scale = (float) 1.0;
size = new Dimension (100,100);

repaint();
}

public void update (int[] sensordata) {
this.sensordata = sensordata;
repaint ();
}

public void setLimits (int close, int medium, int far) {
this.close = close;
this.medium = medium;
this.far = far;
}

public void setScale (float scale) {
this.scale = scale;
}

public void setSize (Dimension size) {
this.size = size;
}

public Dimension getPreferredSize () {
return getMinimumSize();
}

public Dimension getMinimumSize () {
return size;
}

public void paint (Graphics g) {

Dimension s;
```

```
Color color;
int angle,i;

s = getSize();

g.clearRect (0,0,s.width,s.height);
g.drawRect (0,0,s.width-1,s.height-1);
g.fillOval ((int)(s.width/2 - r*scale-2),(int)(s.height/2 - r*scale-2),
(int)(2*r*scale+4),(int)(2*r*scale+4)); // robot

for (angle = 0,i = 0;angle < 360; angle += 15,i++) {

if (sensordata[i] < close) {
color = Color.red;
if (sensordata[i] < (int)(close/3))
color = color.brighter();
if (sensordata[i] > (int)(2*close/3))
color = color.darker();
}

else
if (sensordata[i] < medium) {
color = Color.yellow;
if (sensordata[i] < (int)(close + (medium-close)/3))
color = color.brighter();
if (sensordata[i] > (int)(close + (medium-close)*2/3))
color = color.darker();
}
else {
color = Color.green;
if (sensordata[i] < (int)(medium+(far-medium)/3))
color = color.brighter();
if (sensordata[i] > (int)(medium+(far-medium)*2/3))
color = color.darker();
}

g.setColor (color);
g.fillArc ((int)(s.width/2 - r*scale),(int)(s.height/2 - r*scale),
(int)(2*r*scale),(int)(2*r*scale),angle+2,11);
}

g.setColor (Color.black);
g.fillOval ((int)(s.width/2 - 2*r/3*scale-2),(int)(s.height/2 - 2*r/3*scale-2),
(int)(4*r/3*scale+4),(int)(4*r/3*scale+4)); // robot
}
} // end of SensorCanvas

// Battery canvas class

public class BatteryCanvas extends Canvas {

int min, red, yellow, green, max, voltage;
float scale;
Dimension size;

public BatteryCanvas (int voltage) {
```

```
this.voltage = voltage;
min = 10000;
red = 11000;
yellow = 12000;
green = 13000;
max = 15000;
scale = (float) 1.0;
size = new Dimension (100,50);
repaint();
}

public void setLimits (int min,int red, int yellow, int green, int max) {
this.min = min;
this.red = red;
this.green = green;
this.yellow = yellow;
this.max = max;
repaint();
}

public void update (int voltage) {
this.voltage = voltage;
repaint();
}

public void setScale (float scale) {
this.scale = scale;
}

public void setSize (Dimension size) {
this.size = size;
}

public Dimension getPreferredSize () {
return getMinimumSize();
}

public Dimension getMinimumSize () {
return size;
}

public void paint (Graphics g) {

Dimension s;
Color color;
double angle;

s = getSize();
g.clearRect (0,0,s.width,s.height);
g.setColor (Color.green);
g.fillArc (0,0,s.width,s.height*2,180,-180);
g.setColor (Color.yellow);
g.fillArc (0,0,s.width,s.height*2,180,(int)
(-((double)(yellow-min)/(max-min))*180));
g.setColor (Color.red);
g.fillArc (0,0,s.width,s.height*2,180,(int)
```

```
(-((double)(red-min)/(max-min))*180));
g.setColor (Color.white);
g.fillArc ((int)(10*scale),(int)(10*scale),(int) ((s.width-20)*scale),
(int) ((s.height*2-20)*scale),0,180);
g.setColor (Color.black);
angle = Math.PI*(1-(double)(voltage-min)/(max-min));
g.drawLine ((int) (s.width/2),s.height-1,
(int) (s.width/2*(1+Math.cos(angle))-1),
(int) (s.height-s.width/2*Math.sin(angle)-1));
g.drawString (""+min/1000,(int) (12*scale),s.height);
g.drawString (""+max/1000,(int)
(s.width-23*scale),s.height);
g.drawString ("V",(int) (s.width/2-2*scale),(int)
(s.height/2));
}
} // end of BatteryCanvas

public class MapCanvas extends Canvas {

float scale,teta;
Dimension size;
int x,y;
int[] sensordata;
int[][] map;

int Grid = 15;

public MapCanvas (int[] sensordata, int x, int y, int teta) {

int i,j;

this.sensordata = new int[24];
this.sensordata = sensordata;
this.x = x/1000;
this.y = y/1000;
this.teta = (float) (teta/1000);
scale = (float) 1.0;
size = new Dimension (150,150);

map = new int[Grid][Grid];

for (i=0;i<Grid;i++)
for (j=0;j<Grid;j++)
map[i][j]=0;
map[this.x][Grid-this.y-1]=100; /* robot point */
repaint();
}

public void update (int[] sensordata, int x, int y, int teta) {

int i,j;
int X,Y;
float phi,r;

this.sensordata = sensordata;
this.x = x/1000;
```

```
this.y = y/1000;
this.teta = (float) (teta/1000);

for (i=0;i<Grid;i++)
for (j=0;j<Grid;j++)
map[i][j] -= 1;  /* forgetting obstacles */

for (i=0;i<24;i++) {
r = sensordata[i]/1000;
phi = (float)((7.5 + i*15) * Math.PI / 180);
X = (int)(x*(Math.cos(phi)*Math.cos(teta)-
Math.sin(phi)*Math.sin(teta)) - y*(Math.cos(phi)*Math.sin(teta)+
Math.sin(phi)*Math.cos(teta)) + r);
Y = (int)(x*(Math.sin(phi)*Math.cos(teta)+
Math.cos(phi)*Math.sin(teta)) + y*(Math.cos(phi)*Math.cos(teta)-
Math.sin(phi)*Math.sin(teta)));
if ((X < Grid) && (X > 0) && (Y < Grid) && (Y > 0))
map[X][Grid-Y-1]=60;
/* one minute of life */
}

map[x][Grid-y-1]=100; /* robot point */

repaint();
}

public void setScale (float scale) {
this.scale = scale;
}

public void setSize (Dimension size) {
this.size = size;
}

public Dimension getPreferredSize () {
return getMinimumSize();
}

public Dimension getMinimumSize () {
return size;
}

public void paint (Graphics g) {

int i,j,dx,dy;
Dimension s;

s = getSize();

dx = s.width / Grid;
dy = s.height / Grid;

for (i=0;i<Grid;i++)
for (j=0;j<Grid;j++) {
if (map[i][j] > 0)
  g.setColor(new
```

```
Color((float) (map[i][j]/60),(float) 0.0,(float) 0.0));
else
g.setColor(Color.white);
if (map[i][j] > 60)
g.setColor(Color.black);
g.fillRect (i*dx,j*dy,
(i+1)*dx,(j+1)*dy);
}
g.setColor (Color.black);
for (i=0;i<Grid;i++) {
g.drawLine (i*dx,0,i*dx,s.height);
g.drawLine (0,i*dy,s.width,i*dy);
}
g.drawLine (s.height-1,0,s.height-1,s.width);
g.drawLine (0,s.width-1,s.height,s.width-1);
}
} // end of MapCanvas

} // end of applet
```

.2.3 SocketIO

```
/* -----------------2/24/00 10:52AM------------------
 * SocketIO class
 *
 * Programmer: Sérgio Roberto Gonsalves Tourino
 * Contact: tourino@graco.unb.br
 *
 * This class is used to define input and output streams
 * through a socket connection, and use them to transfer
 * string-like data. The conversion made between byte arrays
 * and string is for C language compatibility (ASCII characters).
 * ------------------------------------------------*/

import java.io.*;
import java.net.*;

public class SocketIO {

  BufferedInputStream in;              // input stream
  BufferedOutputStream out;            // output stream
  Socket socket;                       // socket connection
  String str;
  boolean connected = true;            // connection flag
  boolean flushIO;                     // flush flag
  byte[] inbyte = new byte[100];
  byte[] outbyte = new byte[100];

  /* -----------------2/24/00 10:43AM------------------
   * Default constructor defines flushed IO
   * ------------------------------------------------*/

  public SocketIO (String host, int port) throws UnknownHostException, IOException {

    this (host,port,true);             // Default is flushed IO
  }
```

```
  /* -----------------2/24/00 10:43AM-----------------
   * Constructor to define the flushing behavior
   * ------------------------------------------------*/

  public SocketIO (String host, int port, boolean flushIO) throws
UnknownHostException, IOException {

    this.flushIO = flushIO;             // The user defines flushing
    socket = new Socket(host,port);
    out = new BufferedOutputStream(socket.getOutputStream());
    in = new BufferedInputStream(socket.getInputStream());
  }

   /* -----------------2/24/00 10:44AM-----------------
    * Method for output strings as byte arrays
    * ------------------------------------------------*/

  public synchronized void send (String output) throws IOException {

    output = output + "\n";             // for C compatibility
    outbyte = output.getBytes();        // string->byte array
    out.write (outbyte,0,output.length());
    if (flushIO)
      out.flush();                    // flushes the stream
}

  /* -----------------2/24/00 10:45AM-----------------
   * Method for input byte arrays as string
   * ------------------------------------------------*/

  public synchronized String get() throws IOException {

    int size = in.read(inbyte);
    str = new String (inbyte);          // byte array->string
    str = str.substring (0,size-1);
return str;
}

  /* -----------------2/24/00 10:45AM-----------------
   * Gets the handler for the input stream
   * ------------------------------------------------*/

public InputStream getInputStream () {
return in;
}

  /* -----------------2/24/00 10:46AM-----------------
   * Gets the handler for the output stream
   * ------------------------------------------------*/

public OutputStream getOutputStream () {
return out;
}

  /* -----------------2/24/00 10:46AM-----------------
```

```
   * Gets the handler for the socket connection
   * ------------------------------------------------*/

public Socket getSocket () {
return socket;
}

  /* -----------------2/24/00 10:46AM-----------------
   * Closes all the streams and sockets used
   * ------------------------------------------------*/

  public synchronized void close () throws IOException {

in.close();
out.close();
socket.close();
}

  /* -----------------2/24/00 10:47AM-----------------
   * Default toString method for debugging
   * ------------------------------------------------*/

public String toString () {
return "Socket: "+socket.toString()+"\n"+"Input: "+in.toString()+"\n"+"Output: "+out.toString();
}
}
```

.3 Arquivo de Configuração do Controlador *Fuzzy*

```
// Saved from AST by root@radio, Fri May  5 11:07:25 2000

#and min
#or max
#not not
#implication min
#also max
#composition max
#defuzzification CenterOfArea

type sonar : real [8001] ( 0.000000 < 8000.000000 ) {
   verynear triangle (-1000.000000, 0.000000, 911.032028)
   near triangle (0.000000, 1338.078292, 3103.202847)
   medium triangle (1081.850534, 2960.854093, 4982.206406)
   far triangle (3814.946619, 5124.555160, 6604.982206)
   veryfar triangle (5067.615658, 8000.000000, 8100.000000)
}

type speed : real [1501] ( 0.000000 < 1500.000000 ) {
   veryslow trapezoid (-1000.000000, 0.000000, 53.380783, 154.804270)
   slow trapezoid (218.861210, 261.565836, 362.989324, 469.750890)
   medium trapezoid (507.117438, 565.836299, 693.950178, 816.725979)
   fast trapezoid (870.106762, 955.516014, 1056.939502, 1147.686833)
   veryfast trapezoid (1190.391459, 1270.462633, 1473.309609, 1500.000000)
}
```

```
type distance : real [5001] ( 0.000000 < 5000.000000 ) {
   verynear triangle (-1000.000000, 0.000000, 551.601423)
   near triangle (0.000000, 747.330961, 1868.327402)
   medium triangle (818.505338, 1868.327402, 3220.640569)
   far triangle (2384.341637, 3238.434164, 4181.494662)
   veryfar triangle (3185.053381, 5000.000000, 5100.000000)
}

system (sonar ? sonar, distance ? distance, speed ! speed)
 rulebase {
   if sonar is verynear & distance is verynear   ->  speed is veryslow
   if sonar is verynear & distance is near   ->  speed is veryslow
   if sonar is verynear & distance is medium   ->  speed is veryslow
   if sonar is verynear & distance is far   ->  speed is veryslow
   if sonar is verynear & distance is veryfar   ->  speed is veryslow
   if sonar is near & distance is verynear   ->  speed is veryslow
   if sonar is near & distance is near   ->  speed is slow
   if sonar is near & distance is medium   ->  speed is slow
   if sonar is near & distance is far   ->  speed is slow
   if sonar is near & distance is veryfar   ->  speed is slow
   if sonar is medium & distance is verynear   ->  speed is veryslow
   if sonar is medium & distance is near   ->  speed is slow
   if sonar is medium & distance is medium   ->  speed is medium
   if sonar is medium & distance is far   ->  speed is medium
   if sonar is medium & distance is veryfar   ->  speed is medium
   if sonar is far & distance is verynear   ->  speed is veryslow
   if sonar is far & distance is near   ->  speed is slow
   if sonar is far & distance is medium   ->  speed is medium
   if sonar is far & distance is far   ->  speed is fast
   if sonar is far & distance is veryfar   ->  speed is fast
   if sonar is veryfar & distance is verynear   ->  speed is veryslow
   if sonar is veryfar & distance is near   ->  speed is slow
   if sonar is veryfar & distance is medium   ->  speed is medium
   if sonar is veryfar & distance is far   ->  speed is fast
   if sonar is veryfar & distance is veryfar   ->  speed is veryfast
 }
```