6 (seis)

HENRIQUE GOBBI DE OLIVEIRA

CONTROLE REMOTO PARA ROBÔ MÓVEL

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obter Graduação em Engenharia Mecatrônica.

São Paulo

2001

HENRIQUE GOBBI DE OLIVEIRA

CONTROLE REMOTO PARA ROBÔ MÓVEL

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obter Graduação em Engenharia Mecatrônica.

Orientador:
Prof. Jun Okamoto Júnior
Depto. de Engenharia Mecatrônica

São Paulo
2001

Agradecimento

Agradeço aos meus pais por tudo o que fizeram para que a minha graduação neste curso fosse possível. Obrigado pela paciência e por entenderem a minha ausência nestes últimos meses em que tive que dedicar-me integralmente à conclusão desta faculdade.

Sou grato ao professor Jun Okamoto Junior, responsável por me orientar neste Trabalho Final. Muito obrigado pela sua paciência, pela total disposição em esclarecer as minhas dúvidas e por sempre me indicar o melhor caminho a seguir.

Não posso me esquecer do companheirismo dos meus colegas de classe, que estiveram comigo durante esses anos todos de universidade, dividindo não só as aulas, como também as notas em trabalhos, as cervejadas do Centro Acadêmico e as dúvidas sobre o nosso futuro.

Gostaria de agradecer os meus irmãos, que sempre me atenderam quando pedia que imprimissem algum trabalho, plotassem algum desenho ou retirassem um livro na biblioteca.

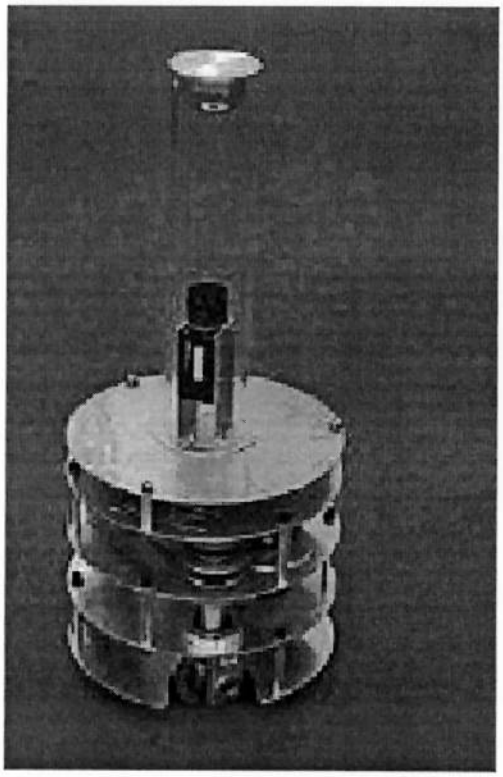
Além desses, tem também a minha gerente Wanda Rosalino, que sempre me dispensou do estágio, quando precisei me ausentar para dar continuidade, e finalmente terminar, o meu Trabalho Final.

"Dedico este trabalho à minha família que sempre me apoiou e a meus pais que além do apoio emocional deram-me também o apoio financeiro para que eu pudesse terminar mais esta etapa de minha vida."

ÌNDICE

1 - Introdução

Hoje em dia o mundo todo está conectado à Internet. Devido à tremenda facilidade de comunicação oferecida por ela, não há limites espaciais para a troca de informações ao redor do mundo.



Com esta facilidade passou a ser comum o controle de equipamentos remotamente valendo-se da facilidade do transporte de informações propiciado pela rede. Hoje em dia é muito comum um Administrador de Sistemas verificar o estado de suas instalações de sua casa através de um computador com um modem. E as coisas vão além disto, existem pessoas que monitoram a própria casa através da Internet, verificando de tempos em tempos as imagens geradas por câmeras instaladas em seus computadores domiciliares.

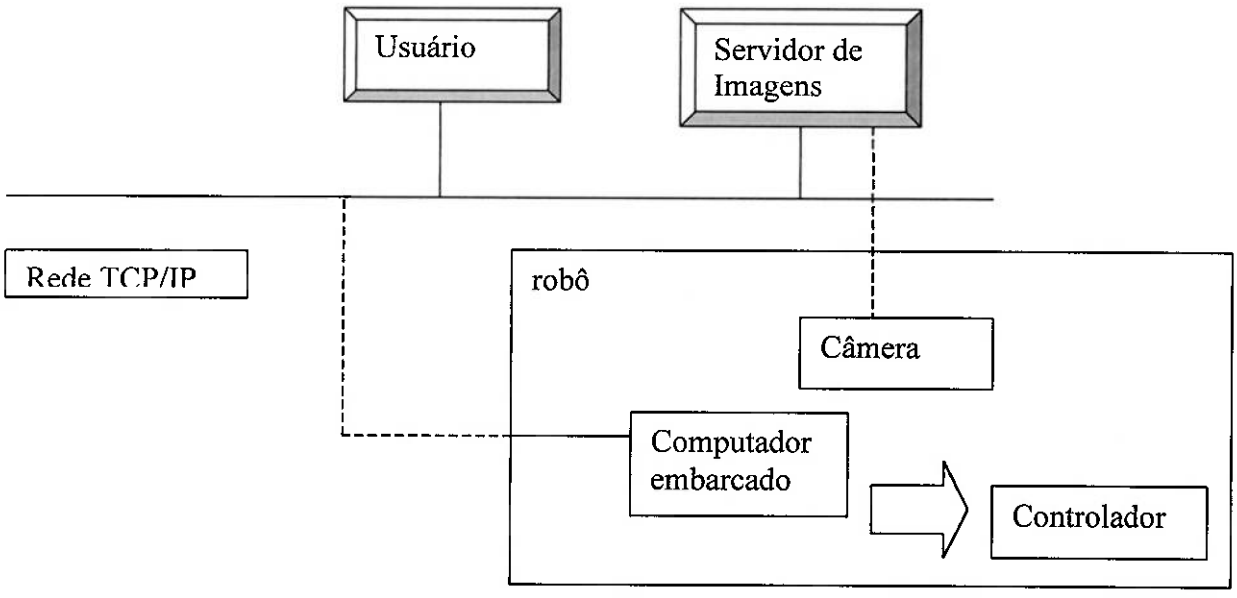
Neste projeto estaremos implementando o controle de um robô pela Internet. Poderemos enxergar ao redor do robô através das imagens de sua câmera e

comandar o seu movimento através de uma página acessivel por qualquer browser.

2 - Ambiente de Desenvolvimento do Projeto

Este projeto tem com ambiente o Laboratório de Telecomando e Monitoração Remota da nossa escola. Lá temos uma rede local (LAN) onde, entre outros computadores, temos uma máquina Linux/Red Hat 7.1 que servirá como nosso servidor de imagens. Além disso, temos um robô dotado de um computador PC-104, uma câmera e um controlador responsável pela sua movimentação.

O ambiente de desenvolvimento deste projeto está representado graficamente abaixo:



Ambiente de desenvolvimento do projeto

Segue abaixo uma descrição detalhada de cada um dos componentes citados:

- **Controlador**: Provê os dois graus de liberdade (movimento no plano) que este robô possui executando o controle dos dois motores (um motor de passo e um motor DC) que movem o robô. Possui uma interface serial por onde recebe os comandos e envia respostas. Apesar do grande número de comandos aceitos por este controlador, neste projeto serão enviados apenas dois comandos, "move" e "rotate" que farão com que o robô se movimente no plano. Os comandos restantes são para verificação do estado dos registradores e dos motores e não serão abordados neste projeto;
- **Computador embarcado**: Computador responsável pela interface entre o robô e a rede local. É um PC padrão (Pentium 233 MHz, 64 Mbytes de RAM), porém com uma *mother board* em tamanho reduzido. Possui duas interfaces seriais onde será conectado o controlador, saída para teclado e vídeo, que não serão utilizadas, e uma interface Ethernet conectada a um adaptador *wireless*. O sistema operacional utilizado neste computador é o Linux – wduarf (white duarf).
- **Câmera**: Apontada para um espelho côncavo, fornece uma visão omnidirecional das proximidades do robô. Envia as imagens via rádio para um receptor que as repassa à placa de captura de vídeo do servidor de imagens;

- **Servidor de Imagens**: Computador (Pentium 4 1.4 GHz 256 Mbytes RAM) com o sistema operacional Linux Red Hat 7.1 que capta as imagens da placa de captura e as converte para um formato comum à Internet. Neste computador há um servidor de páginas eletrônicas (Apache) que fornecerá uma página com as imagens captadas pela câmera;
- **Usuário**: Computador do qual o robô será controlado.

3 - Descrição das necessidades

O objetivo deste projeto é controlar remotamente a movimentação do robô, tanto do motor de passo quanto do motor DC, através de um computador conectado à Internet. Mais especificamente, fornecer um ambiente de transmissão e de dados através de uma rede TCP/IP, que ao mesmo tempo traga as imagens captadas pelo robô até o usuário e transmita os comando do usuário para o robô.

Para alcançar este objetivo o projeto deve:

I – Projetar e implementar um software residente no Servidor de Imagens que envie as imagens captadas pelo robô para quem as solicite de uma maneira eficiente;

II – Projetar e implementar um software residente no Usuário que:

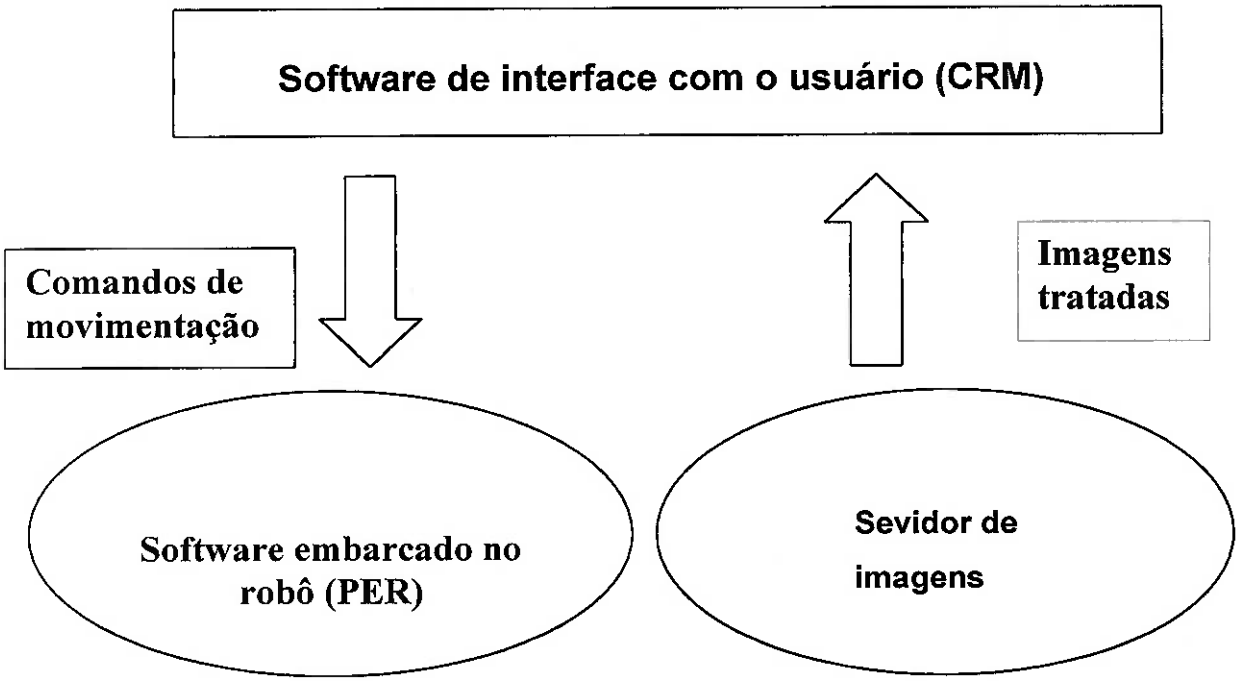
- Solicite as imagens ao Servidor de Imagens e as exiba ao usuário;
- Receba do usuário os comandos de movimentação;
- Se conecte com o robô e lhe passe os comandos recebidos do usuário.

III - Projetar e implementar um software residente no Computador embarcado que receba os comandos de movimentação do usuário, os traduza e os envie para o controlador do robô.

4 - Estudo de possibilidades

Devido à relativa complexidade deste projeto, para podermos estudar as várias maneiras diferentes de alcançarmos o nosso objetivo, precisamos estudar cada funcionalidade separadamente. Isto pode ser feito sem o comprometimento da solução resultante, pois os módulos deste projeto têm o comportamento de "caixas pretas", isto é, o funcionamento interno de uma caixa não influencia no funcionamento da outra. A única coisa que realmente importa, numa análise global, são as entradas e as saídas dos módulos.

De acordo com o que vimos no item anterior, podemos modularizar o nosso projeto da seguinte forma:



Divisão do projeto e relação entre os módulos criados

Teremos então três softwares: o Servidor de Imagens", o PER (Programa Embarcado no Robô) que receberá os comandos do CRM (Comandante do robô móvel) e os passará ao controlador e o CRM que será a interface por trás de todo o ambiente, responsável por captar os comandos do usuário e passá-los ao PER.

Com o problema assim dividido, nós agora podemos estudá-lo como se fossem três problemas distintos, e podemos estudar a melhor solução para cada parte do problema, não nos esquecendo estudar e padronizar devidamente a interface entre eles.

4.1 - O software de interface com o usuário

Este é, sem dúvida, a parte do projeto onde podemos ter muitas soluções possíveis, e a escolhida terá grande influência sobre o aspecto final do projeto, pois este software fará a interface com o usuário.

4.1.1 - Software Fixo ou Página Eletrônica

Um dos primeiros pontos a serem abordados é como este software será instalado. Temos duas possiblidades: executar o controle do robô através de páginas

eletrônicas; ou controlá-lo através de um software fixo em um computador da Internet.

No caso do controle através de páginas da WEB o usuário acessaria um endereço eletrônico que forneceria um software com todas as ferramentas necessárias para a execução da tarefa proposta. Esta solução é atraente pois, permitirá que o robô seja controlado de qualquer lugar em que se tenha um computador conectado à Internet. Além disso, esta solução é totalmente independente da plataforma utilizada, uma vez que, praticamente todas as plataformas existentes contam com um navegador para a Internet.

Em contrapartida, deve-se atentar à questão da segurança. Visto que, a Internet está cheia de pessoas mal-intencionadas, a disponibilidade de uma página para o controle de um robô pode atrair ataques. Se este for o caminho escolhido, deve-se tomar cuidado com o uso deste software. Caso ele seja usado apenas em meio acadêmico, pode-se simplesmente ignorar a questão da segurança, pois o robô será ligado apenas esporadicamente e sempre com a supervisão de uma pessoa responsável. Caso contrário, deve-se implementar uma solução mais segura como por exemplo, o SSL (conexões seguras) e logins/senhas para que o robô tenha pelo menos alguma proteção contra ataques.

4.1.2 - CGI-bin ou Applet

No caso da escolha pelo controle através da WEB há outra decisão a ser tomada: o modo como os comandos sairão do CRM e chegarão ao PER. Para isso, há duas opções, o uso de programas *cgi-bin* (Common Gateway Interface) e o uso de programas embarcados em páginas eletrônicas (*applets*).

No caso do uso de programas cgi, quando o usuário quiser enviar o comando, o navegador passará este comando ao WEB server, que por sua vez, passará este comando para um programa cgi, que enviará este comando ao PER. Como se pode ver é um longo caminho a ser percorrido pelos dados, e este caminho envolve o entrelaçamento de duas máquinas-de-estado distintas (uma entre o navegador e o WEB server e outra entre o WEB server e o PER) o que torna a implementação complicada e a detecção e solução de problemas muito complexas.

No caso do uso de um *applet* a solução ficaria bem mais enxuta, pois com um *applet* é possível abrir *sockets* (conexões através de uma rede TCP/IP) muito facilmente, assim, o *applet* se comunicaria diretamente com o PER não necessitando da intervenção do WEB server.

Esta implementação tem apenas um contratempo: por especificação (que na verdade é uma medida de segurança contra a proliferação de vírus pela Internet) um *applet* só pode abrir conexões com o computador no qual está o WEB server

que lhe forneceu ao navegador, teria-se, portanto, que colocar um WEB server no robô. Isto, na verdade, não é nenhum problema pois temos no robô um sistema operacional Linux como qualquer outro, e colocar um WEB server num Linux é uma tarefa relativamente simples (a maioria das distribuições Linux já vem com um WEB server e no caso do Linux do computador embarcado no robô isto também é verdade).

Vale ressaltar ainda, que no caso do uso de um *applet,* o trabalho do WEB server fica muito reduzido, pois ele somente tem que servir uma página e nada mais. Após isto o *applet* irá se conectar ao PER e fará todo o resto do serviço. Este é mais um argumento que torna viável a colocação de um WEB server no robô visto que, este WEB server terá muito pouco trabalho e não atrapalhará qualquer outra tarefa que este computador venha a exercer no futuro.

Após a apresentação dos fatores a favor da utilização de um controle através da WEB fica difícil falar sobre a solução do software fixo, pois esta não apresenta as vantagens apresentados pela outra.

O primeiro e mais imediato fator seria a necessidade de o usuário instalar o software em cada computador no qual fosse realizar o controle. Além disso, este software iria consumir espaço no disco rígido. Pode se alegar, que hoje em dia, espaço em disco é barato, mas com a solução pela WEB o computador nem ao menos necessita de um disco rígido. E, ainda por cima, se recairia em um dos dois problemas referentes à plataforma: caso se opte por uma independência de plataforma será necessário o uso de uma linguagem que tenha este suporte, e a

maioria das linguagens com esta característica são muito lentas e consomem muitos recursos devido ao porte do interpretador; caso não se escolha a alternativa anterior, se recairá num programa dependente de plataforma, o que não é desejável, visto a grande variedade de plataformas presentes no mercado.

4.1.3 - Forma de obtenção de Comandos

Outro ponto de dúvida é como o usuário irá passar os comando de movimentação para o computador. Temos três possibilidades: teclado, mouse (clicando em botões na tela) e joystick.

A última solução é sem dúvida a mais atraente, porém, ela implica em grandes complicações. Uma delas, é praticamente inviabilizar o controle via WEB, pois pesquisas sobre Java indicaram que isto nunca foi feito e os programadores mais experientes consultados (tanto pessoalmente quanto pela internet) puseram vários empecilhos a esta implementação.

As outras duas soluções não apresentam qualquer tipo de desvantagens ou vantagens uma em relação à outra, sendo muitas vezes questão de escolha pessoal o uso de uma ou de outra. Além disso, não há nenhum problema de implementação para ambos os casos. Visto que, os dois meios de comunicação são muito difundidos e há várias referências onde se apoiar para o desenvolvimento deste tipo de interface.

4.1.4 - Solução escolhida

Com o apresentado acima podemos escolher a realização do controle através da WEB como solução para o nosso problema, pois ela apresenta várias vantagens e nenhuma desvantagem frente à outra solução. Quanto à questão do uso de cgi ou *applets,* será escolhida a segunda devido à maior simplicidade e elegância.

No caso da forma de obtenção de comandos, a solução ideal, no caso da exclusão da possibilidade do uso de joystick receberia tanto comandos passados pelo mouse, através de botões na tela, quanto pelo teclado. Assim, o usuário poderá escolher qual forma de controle usar e a implementação deste software deverá ser feita com vistas a esta especificação.

4.2 - Servidor de imagens

Este software deve fornecer uma página eletrônica que quando acessada forneça as imagens captadas pela câmera do robô. As imagens da câmera estão na memória do computador num formato *raw* e devem ser convertidas para um formato de mais alto nível antes de serem publicadas.

4.2.1 - Streaming ou Snapshots

Para a exibição das imagens nós temos duas opções: a exibição de um vídeo "contínuo" ou a exibição de uma figura que se atualize de tempos em tempos.

A primeira solução é sem dúvida a mais atraente mas também a de mais difícil implementação e a que consumirá mais recursos de banda durante a transmissão de dados. Existem várias soluções deste tipo relatadas na internet, a maioria delas referentes ao uso de webcam (que é uma aplicação bem semelhante a esta), porém elas são sempre acompanhadas de restrições que tornam a implementação difícil ou inviável. Por exemplo, a referência eletrônica (1) faz o seguinte comentário:

> "...Streaming means you see the images of the WebCam in a continuous motion (like in a TV) but the reality is that the image quality tends to be poor and the motion is not as smooth as the TV. This method consumes a lot of resources and it's not as easy to set up as the snapshot WebCams..."

Portanto vemos que o uso de exibição de imagens contínuas faz com que a qualidade da imagem seja prejudicada devido, em grande parte, ao estado tecnológico em que a transmissão de dados se encontra, onde é difícil se transmitir dados a uma velocidade suficiente para a exibição de um vídeo.

Em outras soluções encontradas a exibição de imagens em forma de vídeo esbarra em outro tipo de problema além do citado acima. Aparentemente a grande maioria das aplicações de vídeo desenvolvidas pela comunidade Linux são feitas utilizando-se o driver de captura do grupo "Video for Linux" (referências eletrônicas 2-4), que não é o driver usado atualmente pelo laboratório, portanto a implememementação destas soluções fica dependente da utilização de um sistema operacional com outro kernel, que esteja apto a lidar com este driver, e este tipo de tarefa foge ao escopo deste trabalho.

Além disso, as soluções apresentadas só funcionam com o navegador Netscape, com outros navegadores só é possível a exibição de *snapshots* pois os desenvolvedores alegam que somente o Netscape tem o *plug-in* adequado para a exibição de vídeo em tempo real. Portanto, todo o atrativo da independência e generalidade da solução de um software via WEB seria perdido, pois o usuário precisaria necessariamente estar usando o Netscape para controlar o robô.

A segunda solução é relativamente mais simples que a primeira, porém, apresenta algumas vantagens em relação à ela. As imagens tendem a ter melhor qualidade, são gastos menos recursos de banda e sua manutenção é muito mais simples. O único argumento que se pode pode levantar contra esta solução é que as imagens serão atualizadas de tempos em tempos, ou seja, a imagem andará em "soquinhos", mas isto não prejudica a visualização da imagem tendo em vista a velocidade de movimentação do robô.

Com os fatos apresentados acima, concluí-se que a solução a ser escolhida é a segunda, não por vantagens inerentes à ela, mas sim, pelas desvantagens apresentadas pela primeira. Porém, caso venha-se a trabalhar neste projeto no futuro, deve-se atentar para a primeira solução pois a tecnologia de *streaming* de vídeo é muito nova e evolui muito rapidamente e provavelmente será viável num futuro próximo.

4.3 – O Software embarcado no robô

Para este software não há muitas soluções a serem analisadas. Este software precisa se comunicar com o software de interface com o usuário, receber os comando deste programa através da rede local e passar estes comandos para o controlador através da interface serial do Computador embarcado. Não há muito o que escolher pois a funcionalidade já está determinada.

As mesmas considerações sobre segurança feitas no item 4.1 devem ser levadas em contas aqui. Uma vez que iremos abrir um port do computador, temos que tomar cuidado para que este port não seja uma porta pela qual invasores possam entrar e causar estragos indesejáveis.

4.4– As Interfaces

Após definir o funcionamento dos módulos precisamos definir como será feita a comunicação entre eles, ou seja, como o software de interface com o usuário se comunicará com o PER e com o Servidor de Imagens.

Como foi dito anteriormente, neste projeto haverá comunicação de dados através de uma rede TCP/IP. Para realizar a comunicação entre aplicativos numa rede TCP/IP nós precisamos conhecer os protocolos da camada Transporte do Modelo ISO/OSI (veja o Apêndice A), ou seja o protocolo TCP e o protocolo UDP.

Os protocolos da camada Transporte utilizam-se de um identificador conhecido como *port*. Este identificador é um número de 16 bits, sendo que o conjunto formado pelo endereço IP de um equipamento, por um dos protocolos da camada Transporte (TCP ou UDP), e por um dado *port* do mesmo alocado por uma aplicação, constitui o que se chama comumente com "end-point", ou no jarguão dos programadores, *socket*.

Dito isto, precisamos definir qual será o protocolo e o port usado para a realização da comunicação.

O envio de imagens já está definido, visto que, o servidor de imagens terá necessariamente que fornecer uma página eletrônica com a imagem da câmera,

ou seja, será usado o protocolo de transporte TCP no port 80 que é o padrão para a transmissão do protocolo http (camada aplicação).

Para o envio de comandos nós temos que definir como será esta interface, pois a comunicação realizar-se-á entre duas aplicações que não se encaixam nos padrões existentes na Internet (http, telnet, ftp, etc). Teremos então que escolher um port, um protocolo (TCP ou UDP) e um padrão para as mensagens trocadas entre estes dois softwares.

O protocolo UDP transmite os dados sem, na verdade, se conectar com o outro *end-point* e sem verificar se a mensagem realmente chegou a seu destino, ele simplesmente envia o pacote ao endereço destino. O protocolo TCP, ao contrário, se conecta ao outro *end-point* e faz um controle sobre os dados transportados, fazendo com que não haja perdas de dados durante a comunicação.

Assim, fica claro que escolheremos o protocolo TCP para a nossa interface devido à sua eficiência e confiabilidade. O port pode ser escolhido arbitrariamente, só tomando cuidado de não se escolher um port reservado, ou seja, de zero à 1024, já que estes ports são reservados para as aplicações padrões da Internet (http, ftp, ssh, etc). Para esta aplicação será escolhido o port 2000.

Agora que o protocolo de comunicação está definido, falta escolher o que será transmitido entre o CRM e o PER, ou seja, o padrão de comunicação, e para isto

opta-se pela simplicidade. Os comandos serão enviados em forma de strings contendo os comandos de movimentação. Por exemplo, se o CRM quiser que o robô rotacione 90º, ele envia um pacote contendo a string *"rotate 90"* e então espera-se que o PER saiba o que fazer com esta informação.

5 – Especificação

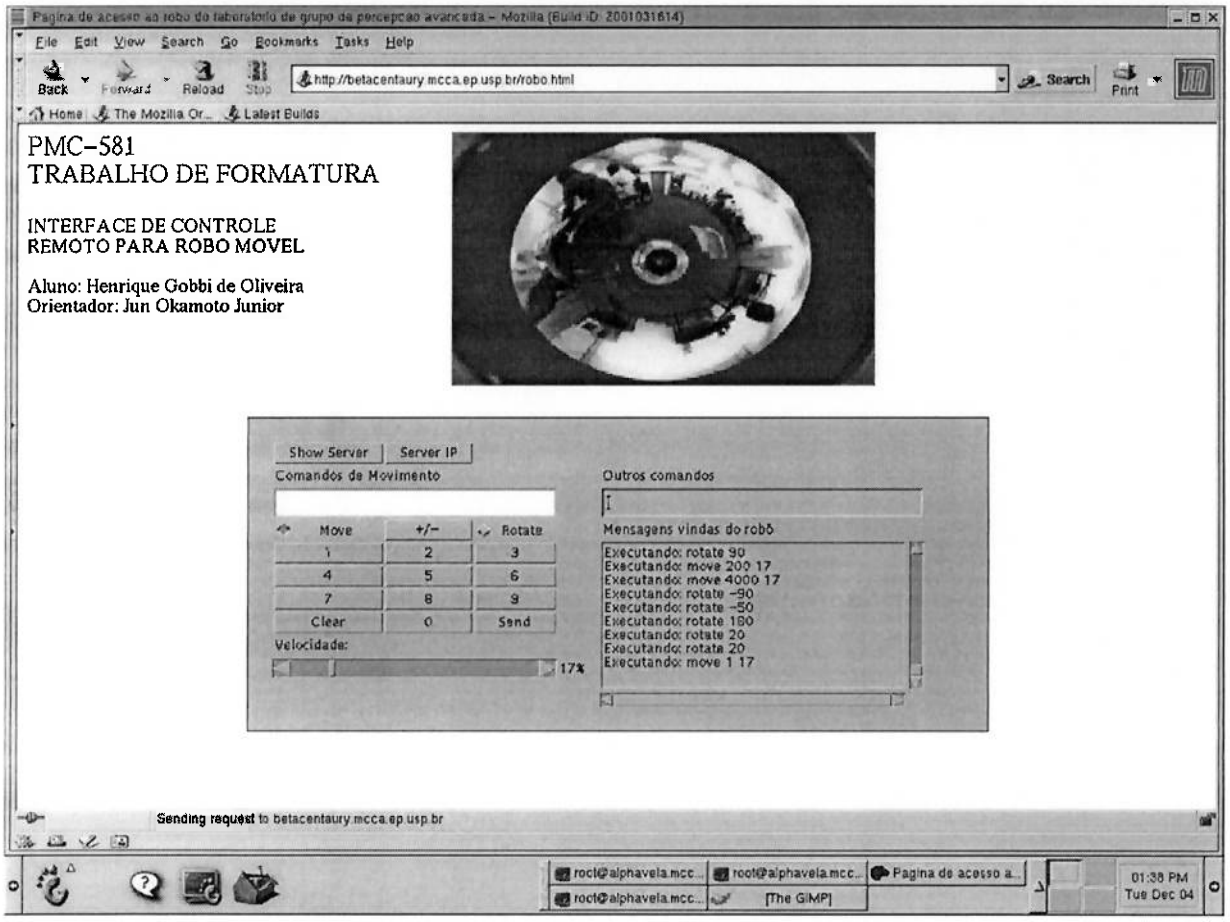
Como visto no capítulo anterior, podemos escrever a seguinte especificação para cada um dos componentes deste projeto:

- CRM: Página eletrônica através da qual o usuário terá acesso às imagens captadas pela câmera do robô e poderá controlar a movimentação do robô através de um applet embarcado nesta página, que se conectará ao PER como cliente numa conexão TCP no port 2000 e lhe mandará os comandos, em forma de strings compreensíveis pelo controlador, ditados pelo usuário tanto via mouse quanto via teclado;
- PER: Programa que aguardará conexões como server no port 2000 e passará os comandos recebidos através deste port para o controlador do robô;
- Servidor de Imagens: Deve fornecer uma página eletrônica contendo as imagens captadas pela câmera do robô, imagens que deverão ser atualizadas em uma taxa de tempo previamente determinada

6 – Detalhamento da solução escolhida

Após decidir como será cada um dos módulos mostrar-se-á agora com detalhes como foi feita a implementação de cada software.

A página final ficou com o seguinte aspecto:



Nela pode-se ver uma imagem da sala e logo abaixo a caixa com o applet onde se pode enviar comandos ao robô.

Segue agora o detalhamento de cada um dos softwares implementados.

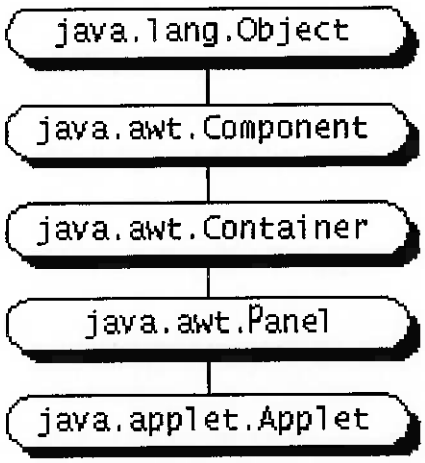
6.1 – O Software de Interface com o usuário

Este programa deve ser um applet (programa embarcada em páginas eletrônicas), que se conectará ao PER como cliente numa conexão TCP no port 2000 e lhe enviará os comandos que o usuário desejar. Este programa foi inteiramente escrito em Java e espera-se que o leitor tenha algum conhecimento em orientação de objeto para que possa compreender este relatório na sua totalidade.

6.1.1 – Applets

Um applet é um programa escrito em Java que pode ser incluído numa página HTML da mesma forma que uma figura. Quando usamos um navegador que possui um interpretador Java, o applet é carregado e executado pelo JVM (Java Virtual Machine) contido no navegador assim que abrirmos a página.

Todo applet é criado através de uma subclasse da classe Applet (java.applet.Applet). A figura abaixo mostra a herança hierárquica da classe Applet:



Por herança todo applet tem quatro funções que respondem pelas quatro operações mais básicas na vida de um applet:

```
public class Simple extends Applet {
   . . .
   public void init() { . . . }
   public void start() { . . . }
   public void stop() { . . . }
   public void destroy() { . . . }
   . . .
}
```

e estas funções correspondem aos seguintes eventos:

- Init: inicializa o applet a cada vez que ele é carregado (ou recarregado). Este método é usado para inicializações que só necessitam

ser feitas uma vez e que não são muito demoradas (por exemplo: se seu applet for exibir figuras, este é um bom lugar para carregá-las);

- Start: todo applet que faz alguma coisa depois da inicialização (exceto respostas a eventos) deve redeclarar este método. Este método deve iniciar os processos que por acaso este applet execute. No nosso caso, este método irá abrir uma conexão com o PER;
- Stop: esta rotina é correspondente ao start(). Todos os processos que a rotina start iniciou devem ser finalizadas por esta rotina, por exemplo, se um applet exibe uma animação, ele deve parar de tentar desenhar na tela quando o usuário está olhando outra janela.;
- Destroy: O uso desta rotina é raro, uma vez que todas as finalizações costumam ser feitas na rotina stop. Ela faz todas as coisas que precisam ser feitas para que o applet possa ser finalizado.

Porém, nem todo applet precisa redeclarar estas quatro funções, applets muitos simples podem até não redeclarar nenhuma delas, mas o mais comum é que o applet precise de redeclarações.

6.1.2 – A classe comandante

De acordo com os conceitos de orientação de objeto, funcionalidades distintas devem ser modularizadas e colocadas em classes distintas. Nada mais natural

para o nosso programa, portanto, do que criar uma classe que se conecte ao PER e forneça rotinas para que facilmente se troque mensagens com ele.

Para isto foi desenvolvida a classe comandante (listagem no apêndice C). Esta classe, ao ser instanciada, imediatamente se conecta ao socket passado como parâmetro (endereço IP + port) através de uma instanciação da classe socket.

A classe socket é um cliente de uma conexão TCP. Ela fornece conexão com outro end-point através de dois streaming, um de input e outro de output. A classe comandante cria, a partir destes dois streamings duas rotinas para que se possa enviar dados ao robô e capturar os dados enviados no sentido inverso. Estas duas rotinas são send e get.

A rotina send recebe como parâmetro um string, que será passado ao robô sem nenhuma alteração. A rotina get não recebe nenhum parâmetro e, quando chamada, lê todos os dados enviados pelo robô e retorna-os em formato de um string.

Além disso, a classe comandante também fornece a rotina finalize que fecha os dois streamings de comunicação e também a conexão com o outro end-point.

6.1.3 – O Abstract Windowing Toolkit (AWT)

O AWT é um meio simples e flexível para a construção de aplicações que necessitem de uma interface gráfica para sua operação. O modelo AWT é dividido em vários componentes com suas próprias funcionalidades:

- Frames;
- Components;
- Panels;
- Layout Managers;
- Events.

Um frame nada mais é do que uma janela da interface gráfica do seu computador. Programas gráficos rodam dentro de um ou mais frames.

Os components são um conjunto de objetos customizados para vários tipos de interação com o usuário. Entre estes objetos encontramos botões, textos, listas para seleção de opções, barras de rolagem entre outras coisas.

Panels são usados para agrupar um conjunto de componentes que exerçam uma determinada funcionalidade. Visto que panels são também componentes, um panel pode conter outros panels e assim por diante.

Layout Managers são ferramentas que permitem o controle sobre a disposição dos componentes e dos panels dentro de um frame. Existem várias maneiras distintas de se realizar este controle e estas várias maneiras são representadas por diferentes managers.

Por fim nós temos os eventos. A maioria das interfaces gráficas é baseada em eventos, o programa espera que o usuário faça algo para que ele responda adequadamente. Eventos neste caso podem ser tantos click em botões, alguma informação digitada no teclado e assim por diante.

Neste trabalho utilizar-se-ão todos os componentes do modelo AWT menos o frame e do panel, pois o browser já fornece um frame e um panel. Na verdade a classe applet é derivada da classe panel (java.awt.Panel).

6.1.4 – O CRM

Toda a discussão feita até agora neste capítulo culmina com este applet. Este programa deverá se conectar ao PER para lhes passar os comandos de movimentação e portanto terá que se valer da classe comandante para auxiliá-lo nesta tarefa. Além disso, ele deve interagir graficamente com o usuário para obter os comandos de movimentação e, para isso, serão usadas classe do AWT.

A listagem deste applet está no apêndice D onde pode-se visualizar tudo o que estará sendo comentado. Este programa, após algumas declarações, chega a função start onde são colocados todos os componentes usados pelo usuário para controlar a movimentação do robô.

São usados os seguintes componentes:

- Label: todos os títulos da interface são labels, além disso, este componente é muitas vezes usado como espaçador com strings cheios de espaços, para separar componentes;
- Button: Este componente é utilizado para diversas tarefas. No topo da janela há dois botões, o Server e o Server IP que mostram respectivamente o nome do server e o endereço IP do server, estes botões servem como ferramentas de debug. Há também os botões contendo os algarismos de 0 a 9 para que o usuário possa passar valores numéricos, o botão +/- para que o usuário possa trocar o sinal do número digitado, o botão Clear que limpa o número atual e o botão Send que envia o comando digitado ao robô;
- Checkbox: Há apenas um componente deste tipo neste applet. Ele serve para o usuário escolher se quer enviar um comando de rotação ou de movimentação;
- TextField: Há dois TextFields neste programa, um para comandos de movimentação e outro para outros comandos. O primeiro é onde aparecem os números pressionados nos botões, o qual, o usuário não pode editar via

teclado. O outro, que só se pode editar com o teclado, serve para que usuário passe outros comandos que não sejam de movimentação ao robô;

- Scrollbar: Este componente é usado para que o usuário escolha a velocidade com a qual o robô irá responder aos comandos move. Os valores deste scroll vão de 1 à 100 % e representam a porcentagem da máxima velocidade alcançavél pelo robô;
- TextArea: É usada para imprimir as mensagens vindas do PER. Este componente é uma caixa de texto que automaticamente cria um scroll quando o texto começa a se tornar extenso.

Após a colocação dos componetes o applet tenta instanciar a classe comandante. Caso ele não não consiga, o programa é abortado e uma mensagem de texto é impressa (caso o browser tenha uma janela de diálogo para aplicações Java como o Konqueror). Caso ele consiga, o applet está pronto e está somente aguardando eventos para trabalhar.

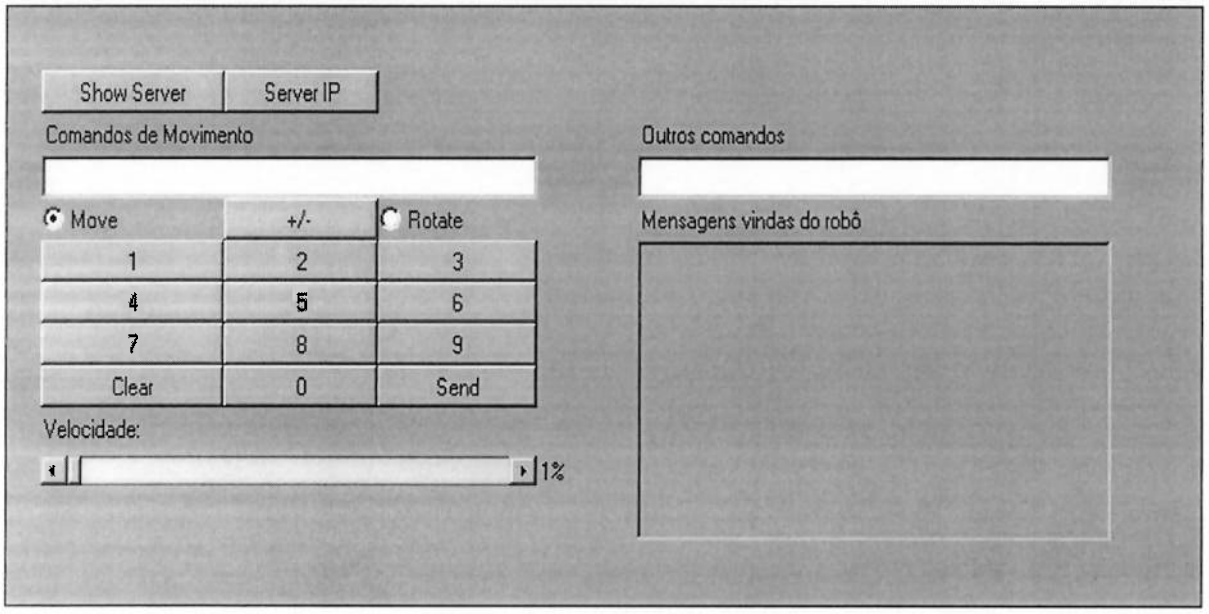
Como se pode ver no código fonte, para alguns componentes é executado a rotina addActionListener(this), esta rotina informa ao sistema que caso seja detectado algum evento relacionado a este objeto, a rotina actionPerformed contida nesta classe deve ser chamada tendo como parâmetro o próprio objeto.

A rotina ActionPerformed deste programa é onde se encontra toda a sua funcionalidade. Ele verifica qual objeto gerou o evento e realiza a ação

correspondente a este objeto. No caso dos botões numéricos, do botão +/- e do botão de Clear, ela atualiza o TextField correspondente aos comandos de movimentação. Caso seja pressionado Server ou Server IP, esta rotina imprime a informação correspondente na TextArea. Caso seja pressionado o botão Send, a rotina envia a informação contida na TextField Comandos de Movimentação, na Checkbox e, eventualmente, na Scrollbar para o robô, através da rotina send da classe comandante. O último evento é relacionado à TextField "outros comandos". Quando um evento relacionado a este objeto é detectado, a ActionPerformed envia o comando contido em "outros comandos" através da classe comandante.

Ainda em relação ao tratamento dos eventos tem-se que falar sobre a Scrollbar que tem um tratamento diferenciado. Ao invés de utilizar a ActionPerformed ela utiliza a adjustamentValueChanged através do comando addAdjustamentListener(this). A rotina adjustamentValueChanged apenas atualiza o label referente à velocidade para o valor correspondente ao Scrollbar toda vez que este valor é modificado.

O resultado final pode ser visualizado abaixo:



6.2 - Servidor de imagens

Como decidido anteriormente este programa deve fornecer uma página eletrônica contendo a imagem captada pela câmera do robô, imagem esta que deve ser atualizada de tempos em tempos.

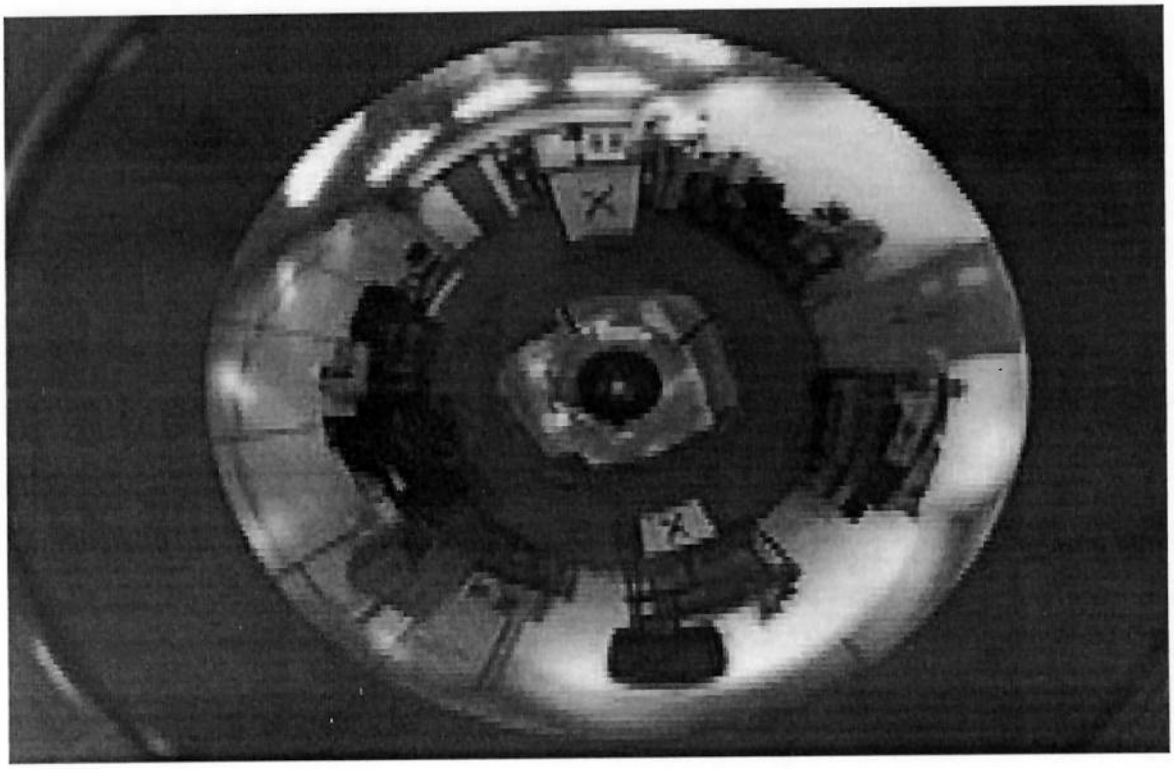
Para fornecer uma página eletrônica há de se ter um WEB server instalado na máquina. O sistema operacional usado (Linux/Red Hat 7.1) já vem com o Apache, portanto, tem-se somente que gerar a página com a figura.

A figura é a parte mais complicada deste problema. As imagens estão em formato *raw* na RAM do computador. Para capturá-las utilizamos um programa chamado snap que lê a memória e transforma os dados obtidos numa figura em formato PGM. Porém, os navegadores não reconhecem este formato, então, precisamos utilizar outro aplicativo fornecido pela RedHat, o ppmtojpeg que transforma a nossa figura PGM para o formato JPEG que é compreendido por qualquer navegador.

Porém, esta tarefa deve ser feita a todo momento para que a imagem esteja sempre atualizada, então foi feito um programa em shell script que faz o processo descrito acima a todo momento. A listagem deste programa está no apêndice E.

Com a figura pronta nós precisamos agora escrever uma página html que mostre esta figura. Esta tarefa é relativamente simples e não merece maiores comentários. A listagem da página está no apêndice F. Note que há uma *tag* com comando *Refresh*, este comando diz para o navegador atualizar a página a cada n segundos, o número n é escolhido no campo *content* da mesma *tag*. De acordo com o que está representado no apêndice, a página será atualizada a cada dois segundos, que foi o valor escolhido após vários testes correspondendo satisfatoriamente às necessidades. No entanto, ele pode ser alterado para qualquer outro valor desejado.

Abaixo temos um snapshot capturado



6.3 – O PER

Este programa receberá os comandos enviados pelo CRM através de uma conexão TCP no port 2000 e passará este comandos para o controlador através da porta serial do computador. Para receber os comandos pela rede ele primeiro deve receber uma conexão, para isto este programa deve ficar aguardando conexões no port 2000 do computador e responder às requisições que chegem por esta porta.

O código fonte deste programa esta no apêndice G. Nele pode-se visualizar tudo o que foi dito acima. Após algumas declarações a rotina principal chama a rotina socket, avisando-a de que se deseja um socket de rede (AF_INET) e que o protocolo a ser usado é o TCP (SOCK_STREAM). Mais informações sobre a rotina socket se encontram no apêndice B.

Após isso é preciso avisar o sistema operacional do computador que este programa passará a responder por todas as requisições ao port 2000. Para isto utiliza-se a rotina bind. Esta rotina recebe três parâmetros: o socket criado anteriormente, uma estrutura do tipo sockaddr_in e o tamanho desta estrutura. Nesta estrutura, o campo denominado sin_port representa o port que será usado na conexão.

Para encerrar-se o set up do server nos precisamos criar uma lista para as conexões que chegam. Fazemos isto com o função listen. No nosso caso nós passamos como parâmetros o nosso socket e o número dez que indica que a fila conterá no máximo 10 conexões.

Vale ressaltar que este programa não aceita mais de uma conexão ao mesmo tempo. O que na verdade faz total sentido pois duas pessoas não podem controlar o robô ao mesmo tempo. Se chegar um pedido de conexão enquanto alguém já no controle, este pedido ficará na fila até que a primeira conexão seja finalizada.

Agora que a parte de rede esta certa precisa acertar a parte da comunicação com o controlador do robô, porém esta é bem mais simples que a primeira. Como no sistema operacional Linux devices correspondem à arquivos nós podemos tratar nossa porta serial como sendo um arquivo, assim as tarefas de leitura e escritas podem ser feitas da maneira mais simples possível (fprintf e gets). Portanto para se conectar ao controlador o único comando necessário é o fopen.

Após isto o programa entra num laço infinito onde ele aguarda a chegada de dados pela rede (read) , envia os dados recebidos para o controlador (fprintf) e envia uma confirmação de recebimento de comando ao cliente (write). Os printf´s desta rotina são apenas para debug e, caso haja um terminal aberto, imprimem algumas informações na tela.

6 - Conclusão

Após verificar a diversidade de temas abordados neste trabalho é possível concluir que ele foi muito proveitoso. Nele abordou-se quatro linguagens de programação: C, Java, HTML e Shell Script. E além disto teve-se contato com uma questão muito interessante: a transmissão de imagens pela Internet.

Porém há muita coisa que pode ser melhorada neste trabalho ainda. Pode-se implementar a exibição das respostas fornecidas pelo controlador para os comandos relativos à estado de registradores e motores. Pode-se também transformar a exibição de snapshots em streamings de video. Fica a sugestão caso alguém se interesse em dar continuidade a este projeto.

O resultado final foi bom. Agora é possível o controle do robô de qualquer lugar com Internet, mas existe um porém, o link deve ter uma velocidade boa pois senão a visualização das imagens fica prejudicada. Sobre as imagens vale também salientar que é preciso um certo treino para as vizualizações numa imagem omnidirecional. No começo as coisas parecem estranhas e é difícil a identificação de objetos e até mesmo de pessoas.

7 – Referências:

7.1 – Bibliográficas:

1 – "Object-oriented programming and Java"

Danny C.C. Poo

Derek B.K. Kiong

2 – "Guia Internet de Conectividade"

Cyclades Brasil

3 – "Curso TCP/IP Básico"

Cyclades Brasil

4 – "Beginning Linux Programming"

Richard Stones

Neil Matthew

7.2 – Eletrônicas:

1 – webcamworld.com: WebCam Lab : Streaming or Snapshot WebCam

http://developers.webcamworld.com/methods.html

2 – Video for Linux resources

http://www.exploits.org/v4l

3 – Building Number Three

http://roadrunner.swansea.uk.linux.org/v4l.shtml

4 – Official Camserv home page

http://cserv.sourceforge.net/

5 – The Source for Java Technology

http://java.sun.com/

6 – Free Java Applets, Games, Programming Tutorials, and Downloads

http://javaboutique.internet.com/

Apêndice A: Modelo ISO/OSI

Origem

O modelo OSI (Open System Interconnect) foi criado em 1977 pela ISO (International Organization for Standardization) com o objetivo de criar padrões de conectividade para a interligação de sistemas de computadores.

Descrição

Os aspectos gerais desta conectividade foram divididos em 7 camadas funcionais, facilitando assim a compreensão das questões fundamentais de um processo de comunicação entre programas de uma rede de computadores, apesar de tal modelo não ter sido adotado para fins comerciais. A tabela abaixo mostra o modelo ISO/OSI e a função dos protocolos de comunicação em cada uma das camadas desse modelo. O modelo ISO/OSI faz assim uma divisão muito clara das funcionalidades das camadas de um sistema de comunicação. Ele é de grande auxílio para o entendimento das diversas arquiteturas de comunicação existentes no mercado.

Camada Física	A camada física compreende as especificações do hardware utilizado na

	rede, compreendidas entre aspectos mecânicos, elétricos e físicos.
Camada de Enlace	A "visão" da camada de enlace se restringe a dois nós da rede somente: os protocolos desta camada têm como funcionalidade principal a de fazer com que os dados transmitidos de um computador cheguem ao outro diretamente ligado a ele com integridade.
Camada de Rede	Na camada de rede o conhecimento da rede passa a existir (topologia, como os nós estão conectados entre si). Protocolos desta camada tratam de encaminhar as mensagens na rede segundo algoritimos de roteamento, disciplinas de controle de fluxo e endereçamento.
Camada de Transporte	Os protocolos de transporte possuem a "visão fim-a-fim" de um processo de comunicação: eles devem garantir que os dados transmitidos por um programa

	de um computador cheguem ao seu destino (outro programa) com integridade, usando para isso mecanismos tais como controle de fluxo, correção de erros, entre outros.
Camada de Sessão	Esta camada trata do "diálogo" entre os programas rodando em computadores de uma rede. Detalhes com autenticação, tipo de comunicação, e estabelecimento de pontos de sincronismo na comunicação são tratados nesta camada.
Camada de Apresentação	Trata da sintaxe e semântica dos dados transmitidos entre os programas. Criptografia, conversão entre caracteres ASCII e EBCDIC, compressão e descompressão de dados,etc.
Camada de Aplicação	Trata da definição dos protocolos de aplicação propriamente ditos. É importante observar que esta camada não define como a aplicação final deve ser, mas sim o protocolo de aplicação

	correspondente.

Nesse modelo pode-se notar também que as funcionalidades de um sistema de comunicação foram divididas em dois domínios, o da rede, referente à conectividade entre os computadores, descrito pelas camadas 1 a 3 (Física, Enlace e Rede), e o da aplicação, referente à comunicação entre os programas que fazem uso da rede, descrito pelas camadas 5 a 7 (Sessão, Apresentação e Aplicação). A camada 4 (Transporte) é a camada que faz assim a ligação entre os programas de aplicação e os recursos das redes de computadores.

Apêndice B: Socket Manual

SOCKET(2) Linux Programmer's Manual SOCKET(2)

NAME

socket - create an endpoint for communication

SYNOPSIS

#include <sys/types.h>

#include <sys/socket.h>

int socket(int domain, int type, int protocol);

DESCRIPTION

Socket creates an endpoint for communication and returns a descriptor.

The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. The currently understood formats include:

Name Purpose Man page

PF_UNIX,PF_LOCAL	Local communication	unix(7)
PF_INET	IPv4 Internet protocols	ip(7)
PF_INET6	IPv6 Internet protocols	
PF_IPX	IPX - Novell protocols	
PF_NETLINK	Kernel user interface device	netlink(7)
PF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
PF_AX25	Amateur radio AX.25 protocol	
PF_ATMPVC	Access to raw ATM PVCs	
PF_APPLETALK	Appletalk	ddp(7)
PF_PACKET	Low level packet interface	packet(7)

The socket has the indicated type, which specifies the communication semantics. Currently defined types are:

SOCK_STREAM

Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK_DGRAM

Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET

Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each read system call.

SOCK_RAW

Provides raw network protocol access.

SOCK_RDM

Provides a reliable datagram layer that does not guarantee ordering.

SOCK_PACKET

Obsolete and should not be used in new programs; see packet(7).

Some socket types may not be implemented by all protocol families; for example, SOCK_SEQPACKET is not implemented for AF_INET.

The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists

to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is specific to the "communication domain" in which communication is to take place; see protocols(5). See getprotoent(3) on how to map protocol name strings to protocol numbers.

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. They do not preserve record boundaries. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect(2) call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2) and recv(2) calls. When a session has been completed a close(2) may be performed. Out-of-band data may also be transmitted as described in send(2) and received as described in recv(2).

The communications protocols which implement a SOCK_STREAM ensure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot

be successfully transmitted within a reasonable length of time, then the connection is considered to be dead. When SO_KEEPALIVE is enabled on the socket the protocol checks in a protocol-specific manner if the other end is still alive. A SIGPIPE signal is raised if a process sends or receives on a broken stream; this causes naive processes, which do not handle the signal, to exit. SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded. Also all message boundaries in incoming datagrams are preserved.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in send(2) calls. Datagrams are generally received with recvfrom(2), which returns the next datagram with its return address.

SOCK_PACKET is an obsolete socket type to receive raw packets directly from the device driver. Use packet(7) instead.

An fcntl(2) call with the the F_SETOWN argument can be

used to specify a process group to receive a SIGURG signal when the out-of-band data arrives or SIGPIPE signal when a SOCK_STREAM connection breaks unexpectedly. It may also be used to set the process or process group that receives the I/O and asynchronous notification of I/O events via SIGIO. Using F_SETOWN is equivalent to an ioctl(2) call with the SIOSETOWN argument.

When the network signals an error condition to the protocol module (e.g. using a ICMP message for IP) the pending error flag is set for the socket. The next operation on this socket will return the error code of the pending error. For some protocols it is possible to enable a per-socket error queue to retrieve detailed information about the error; see IP_RECVERR in ip(7).

The operation of sockets is controlled by socket level options. These options are defined in <sys/socket.h>. Setsockopt(2) and getsockopt(2) are used to set and get options, respectively.

RETURN VALUES

-1 is returned if an error occurs; otherwise the return

value is a descriptor referencing the socket.

ERRORS

EPROTONOSUPPORT

The protocol type or the specified protocol is not supported within this domain.

ENFILE Not enough kernel memory to allocate a new socket structure.

EMFILE Process file table overflow.

EACCES Permission to create a socket of the specified type and/or protocol is denied.

ENOBUFS or ENOMEM

Insufficient memory is available. The socket cannot be created until sufficient resources are freed.

EINVAL Unknown protocol, or protocol family not available.

Other errors may be generated by the underlying protocol

modules.

CONFORMING TO

4.4BSD (the socket function call appeared in 4.2BSD). Generally portable to/from non-BSD systems supporting clones of the BSD socket layer (including System V variants).

NOTE

The manifest constants used under BSD 4.* for protocol families are PF_UNIX, PF_INET, etc., while AF_UNIX etc. are used for address families. However, already the BSD man page promises: "The protocol family generally is the same as the address family", and subsequent standards use AF_* everywhere.

BUGS

SOCK_UUCP is not implemented yet.

SEE ALSO

accept(2), bind(2), connect(2), getprotoent(3), getsockname(2), getsockopt(2), ioctl(2), listen(2), read(2), recv(2), select(2), send(2), shutdown(2), socketpair(2), write(2)

"An Introductory 4.3 BSD Interprocess Communication Tutorial" is reprinted in UNIX Programmer's Supplementary Documents Volume 1.

"BSD Interprocess Communication Tutorial" is reprinted in UNIX Programmer's Supplementary Documents Volume 1.

Linux Man Page 24 Apr 1999 1

Apêndice C: comandante.java

```
import java.io.*;
import java.net.*;

class comandante {

        Socket soc;
        OutputStream os;
        InputStream is;

        //Comandante (location, port)
        //Esta rotina inicializa uma classe comandante. Ela abre uma conexão com        // o
robô no port especificado e inicializa os streamings de input e de output
        comandante (String robo, int port)
                        throws IOException {
                soc = new Socket(robo, port);
                os = soc.getOutputStream();
                is = soc.getInputStream();
        }

        //Send – esta rotina envia a mensagem para o robô
        void send(String message) {
                try {
                        os.write(message.getBytes());
                } catch (IOException e) {
                        System.err.println("Error sending message");
                }
        }

        //get – retorna as mensagens enviadas pelo robô
        String get() {
                int c;
                StringBuffer str = new StringBuffer(100);
                try {
                        while (( c = is.read()) != 9) {
                                str.append((char) c);
                        }
                } catch (IOException e) {
                        System.err.println("IOException in receiving data");
                }
                return str.toString();
        }
```

```
        //finalize – finaliza o socket e os dois streamings
        public void finalize() {
                try {
                        is.close();
                        os.close();
                        soc.close();
                } catch (IOException e) {
                        System.err.println("IOException:closing connection");
                }
        }

        public static void main(String[] args) {
                String response = new String();
                try {
                        comandante c = new comandante("localhost", 2000);
                        c.send("PrimeiroTeste");
                        response = c.get();
                        System.out.print(response);

                } catch (IOException i) {
                        System.err.println("IOException in connection host");
                }
        }
}
```

Apêndice D: CRM

```
import java.applet.Applet;
import java.awt.*;
import java.awt.Graphics;
import java.awt.event.*;
import java.awt.Frame;
import java.io.*;
import java.net.*;

public class crm extends Applet implements ActionListener, AdjustmentListener {

        comandante com;
        private String[][]      buttonText = {      {"1", "2", "3"},
                                                    {"4", "5", "6"},
                                                    {"7", "8", "9"},
                                                    {"Clear", "0", "Send"}};

        StringBuffer buffer;
        Button button[][] = new Button[4][3];
        Button server, ip;
        Label  label, vVal;
        Checkbox move, rotate;
        CheckboxGroup g = new CheckboxGroup();
        TextField text, inText;
        TextArea outText;
        Scrollbar v;

        public void init() {
                buffer = new StringBuffer();
        }

        public void start() {

                int i, j;
                GridBagLayout gridbag = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                String host = getCodeBase().getHost();

                setLayout(gridbag);
                c.fill = GridBagConstraints.HORIZONTAL;

                label = new Label("  ");
                c.gridx = 0;
```

```
c.gridy = 0;
gridbag.setConstraints(label, c);
add(label);

server = new Button("Show Server");
server.addActionListener(this);
c.gridx = 1;
c.gridy = 0;
gridbag.setConstraints(server, c);
add(server);

ip = new Button("Server IP");
ip.addActionListener(this);
c.gridx = 2;
c.gridy = 0;
gridbag.setConstraints(ip, c);
add(ip);

for ( i=4 ; i<12 ; i++ ) {
        label = new Label("    ");
        c.gridx = i;
        c.gridy = 0;
        gridbag.setConstraints(label, c);
        add(label);
}

label = new Label("Comandos de Movimento");
c.gridx = 1;
c.gridy = 1;
c.gridwidth = 2;
gridbag.setConstraints(label, c);
add(label);

label = new Label("Outros movimentos");
c.gridx = 5;
c.gridy = 1;
gridbag.setConstraints(label,c);
add(label);

c.gridwidth = 1;
for (i=4; i<8; i++) {
        for (j=0; j<3; j++) {
                button[i-4][j] = new Button(buttonText[i-4][j]);
                button[i-4][j].addActionListener(this);
                c.weightx = 1;
                c.gridx = j + 1;
```

```
                c.gridy = i;
                gridbag.setConstraints(button[i-4][j], c);
                add(button[i-4][j]);
        }
}

move = new Checkbox("Move", g, true);
c.gridx = 1;
c.gridy = 3;
gridbag.setConstraints(move, c);
add(move);

label = new Label("        ");
c.gridx = 2;
c.gridy = 3;
gridbag.setConstraints(label, c);
add(label);

rotate = new Checkbox("Rotate", g, false);
c.gridx = 3;
c.gridy = 3;
gridbag.setConstraints(rotate, c);
add(rotate);

label = new Label("Mensagens vindas do robô");
c.gridx = 5;
c.gridy = 3;
c.gridwidth = 1;
gridbag.setConstraints(label, c);
add(label);

text = new TextField(20);
text.setEditable(false);
text.setBackground( Color.white );
c.gridx = 1;
c.gridy = 2;
c.gridwidth = 3;
gridbag.setConstraints(text, c);
add(text);

inText = new TextField(20);
inText.addActionListener(this);
c.gridx = 5;
c.gridy = 2;
c.gridwidth = 5;
gridbag.setConstraints(inText, c);
```

```
            add(inText);

            label = new Label("Velocidade:");
            c.gridwidth = 1;
            c.gridx = 1;
            c.gridy = 8;
            gridbag.setConstraints(label, c);
            add(label);

            v = new Scrollbar(Scrollbar.HORIZONTAL, 0, 2, 1, 102);
            v.addAdjustmentListener(this);
            c.gridx = 1;
            c.gridy = 9;
            c.gridwidth = 3;
            gridbag.setConstraints(v, c);
            add(v);

            vVal = new Label(v.getValue() + "%");
            c.gridx = 4;
            c.gridy = 9;
            c.gridwidth = 1;
            gridbag.setConstraints(vVal, c);
            add(vVal);

            outText = new TextArea(10,10);
            c.gridwidth = 5;
            c.gridheight = 8;
            c.gridx = 5;
            c.gridy = 4;
            gridbag.setConstraints(outText, c);
            add(outText);

            try {
                  com = new comandante(host, 2000);
            } catch (IOException e) {
                  System.out.println("Não foi possível conectar");
                  outText.append("Não foi possível se conectar ao robô");
            }
            repaint();
}

public void stop() {
      com.finalize();
}

public void destroy() {
```

```
        }

public void adjustmentValueChanged(java.awt.event.AdjustmentEvent e) {

        if ( e.getSource() == v ) {
                vVal.setText(v.getValue() + "%");
                return;
        }
}

public void actionPerformed ( ActionEvent e) {
        int i, j, a = 0, b = 0;
        InetAddress address;
        String host = getCodeBase().getHost();

        if ( e.getSource() == server ) {
                outText.append("Server: " + getCodeBase().getHost() + "\n");
                return;
        }

        if ( e.getSource() == ip ) {
                try {
                        outText.append("Server IP: " + InetAddress.getByName(host) +
"\n");
                } catch (UnknownHostException ex) {
                        System.out.println("Could not get IP address: Unknown Host");
                }
                return;
        }

        if ( e.getSource() == inText ) {
                com.send( inText.getText() );
                inText.setText("");
                outText.append( com.get() + "\n" );
                return;
        }

        for (i=0; i<4; i++) {
                for (j=0; j<3; j++) {
                        if ( e.getSource() == button[i][j] ) {
                                a = i;
                                b = j;
                        }
                }
        }
```

```
        if ( a == 3 ) {
                if ( b == 0 ) {
                        text.setText("");
                } else if ( b == 2 ) {
                        String aux;
                        String value = text.getText();
                        int vel = 1;

                        if ( value.length() == 0) return;

                        if (g.getSelectedCheckbox() == move) {
                                aux = "move ";
                                vel = v.getValue();
                                com.send( aux + text.getText() + " " + vel);
                        } else {
                                aux = "rotate ";
                                com.send( aux + text.getText() );
                        }

                        outText.append( com.get() + "\n");
                        text.setText("");
                } else {
                        text.setText ( text.getText() + buttonText[a][b] );
                }
        } else {
                text.setText( text.getText() + buttonText[a][b] );
        }
}

public void paint(Graphics g) {
    //Draw a Rectangle around the applet's display area.
    g.drawRect(0, 0, size().width - 1, size().height - 1);
  }
}
```

Apêndice E: geradorImg.sh

```
while [ true ]; do
        ./snap -m -c 400 -r 240 -o teste.pgm;
        ppmtojpeg teste.pgm > teste.jpg;
done
```

Apêndice F: imgServer.html

```
<HTML>
<HEAD>
<TITLE>Imagem obtida a partir da camera do robô</TITLE>
</HEAD>

<BODY BGCOLOR="#FFFFFF">
<meta http-equiv="Refresh" content="2">
        <center"><IMG SRC="teste.jpg"></center>
</BODY>
```

Apêndice G: PER

```
/*************************************************************************
                PER   -       Programa embarcado no robô

        Este módulo fica "escutando" no port 2000 e após aceitar um pedido de
conexão passa os comandos recebidos pela rede para o robô pela porta serial
*************************************************************************/

# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netinet/tcp.h>
# include <stdio.h>
# include <unistd.h>

# define MSG_MAX 50
# define DEFPORT   2000

# ifndef TRUE
# define TRUE 1
# define FALSE 0
# endif

char buf[MSG_MAX];

int main(int argc, char * argv[])
{
        int server_sock, client_sock;
        struct sockaddr_in server, client;
        int server_len, client_len;
        char * device;
        FILE *f;

        setbuf (stdout, NULL);

//Verificando se o usuário passou a informação necessária
        if(argc != 2){
                printf("\r\nUsage: per <device>");
                exit(1);
        }

//Acertando o ponteiro para o device
```

```
        device = argv[1];

//Criando um socket
        if ((server_sock = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

//Setando a estrutura de endereço do port em que o programa escutará
        bzero ((char *) &server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(DEFPORT);
        server_len = sizeof (server);

//Associando este port a este programa
        if (bind (server_sock, (struct sockaddr *) &server, server_len) == -1) {
                perror("bind");
                exit(1);
        }

//Inicializando a fila de conexões
        if (listen (server_sock, 10) == -1) {
                perror("listen");
                exit(1);
        }

//Abrindo conexão com a porta serial
        if ( (f = fopen (device, "w")) == NULL ){
                printf("\r\nErro no fopen");
                exit(1);
        }
//Aceitando conexões e recebendo comandos
        client_len = sizeof(client);
        while (1){
                client_sock=accept(server_sock,(struct sockaddr *)&client,
                    &client_len);

                while (1) {
                        int msg_len;
                        char x[2 * MSG_MAX];
                        char end = (char) 9;
                        printf("\r\nServer Waiting:");

                        memset( (void *) buf, 0, MSG_MAX);
                        memset( (void *) x, 0, 2*MSG_MAX);
```

```
                msg_len = (int) read(client_sock, &buf, MSG_MAX);
                printf("Message len: %d", msg_len);
                if (msg_len == 0 || msg_len == -1) {
                        close(client_sock);
                        break;
                }

                printf("   Message:%s", buf);
                fprintf(f, "%s\n", buf);

                strcpy(x, "Recebido");
                strncat(x, buf, msg_len);
                x[msg_len + 10] = end;
                printf("\nSending: %s", x);
                write(client_sock, x, msg_len + 11);
        }
    }
}
```