Aléxis Rodrigues de Almeida

SISTEMA DE CONTROLE DE ROBÔ BASEADO NA PLATAFORMA LINUX: UMA PROPOSTA

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Wilian Soares Lacerda (DCC-UFLA)

Lavras
Minas Gerais - Brasil
2004

Aléxis Rodrigues de Almeida

SISTEMA DE CONTROLE DE ROBÔ BASEADO NA PLATAFORMA LINUX: UMA PROPOSTA

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *18 de Setembro de 2004*

Prof. Luciano Mendes dos Santos (DCC-UFLA)

Prof. Roberto Alvez Braga Junior (DEG-UFLA)

Prof. Wilian Soares Lacerda (DCC-UFLA)
(Orientador)

Lavras
Minas Gerais - Brasil

Agradecimentos

A Deus por ter me dado forças nos momentos em que os obstáculos pareciam intransponíveis.

Ao professor Wilian por ter me indicado um tema de estudo tão fascinante.

Resumo

Este trabalho apresenta todos os componentes de *hardware* e *software* necessários para controle de um robô a partir de um computador tipo PC. A solução, baseada em um sistema Linux, provê mecanismos de controle do robô Monty a partir da porta serial do computador. O robô é ligado ao computador por meio de um cabo, que contém, em uma de suas extremidades, um circuito conversor de sinais entre os padrões TTL e RS232. A comunicação entre o computador e o robô se dá por meio do protocolo RS232. A linguagem utilizada para controlar o robô é formada por um subconjunto de comando da linguagem LOGO. Esses comandos básicos podem ser combinados de diversas maneiras, gerando, assim, um conjunto maior e mais complexo de ações por parte do robô. O sistema prevê a execução de comandos tanto de forma interativa quanto em bloco, de maneira que um conjunto de comandos pode ser previamente armazenado em um arquivo e submetido, em bloco, para execução pelo robô.

À minha esposa Gilzélia e minhas filhas Rebeca e Amanda, por todos os momentos que não pudemos estar juntos nos últimos 16 meses.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

INTRODUÇÃO

Há muito que se fala em robótica e todas as facilidades e avanços que essa ciência pode trazer para o dia-a-dia da sociedade. Atualmente, algumas discussões ainda flutuam em torno do mito do robô totalmente autônomo, com inteligência própria e até, em alguns casos, com capacidade de expressar sentimentos próprios. Entretanto, o que se tem realmente consolidado ultimamente é o uso da robótica em situações práticas de alta periculosidade ou insalubridade para o ser humano, ou em tarefas que exigem extrema precisão em sua execução, ou simplesmente em atividades com um alto grau de repetitividade.

Dentro desse espectro de possíveis atuações, os robôs atualmente são utilizados, por exemplo, para desarmar bombas, limpar dutos de ar condicionado, montar placas de circuito impresso com alto índice de integração ou participar da linha de montagem de automóveis. A partir desses exemplos é possível perceber que a aplicação prática dos robôs nos dias de hoje ainda está muito longe da visão romântica dos livros e filmes de ficção científica. Apesar disso, esses incríveis equipamentos tornaram-se indispensáveis em diversos setores produtivos em todo o mundo.

1.1 Objetivos

O objetivo principal do trabalho é fornecer à comunidade Linux uma descrição detalhada de todos os elementos necessários para a construção de um sistema de controle remoto de um robô a partir de uma estação Linux. Como objetivos específicos podem ser citados:

- Construir um programa para o robô que possibilite receber os comandos do computador, executá-los e fornecer ao programa controlador o resultado da execução desses comandos.

- Construir um cabo conector capaz de interconectar o robô e o computador.

- Construir um programa controlador que seja capaz de interpretar comandos, fornecidos pelo usuário em uma linguagem de alto nível, e enviá-los ao robô, para execução.

Capítulo 2

REVISÃO DA LITERATURA

2.1 O robô

Conforme definição do dicionário Aurélio, em sua 1^a edição, o termo robô origina-se do francês *robot*, que por sua vez vem do tcheco *robota*, palavra criada pelo escritor tchecoeslovaco, Karel Chapek, e que significa "trabalho forçado". Segundo Aurélio, um dos significados do termo robô é: *Mecanismo automático, em geral com aspecto semelhante ao de um homem e que realiza trabalhos e movimentos humanos*.

Os robôs podem ser divididos em dois grandes grupos: autônomos e controlados.

2.1.1 Robôs autônomos

Os robôs autônomos são aqueles que já trazem em sua memória a programação necessária para desempenhar todas as atividades que lhe são atribuídas. Ainda se pode dividir esse grupo em dois outros diferentes.

No primeiro subgrupo estão os robôs autônomos com seqüência de passos predeterminada, tendo, portanto, atuação limitada e específica. Por exemplo, no caso de um braço mecânico, exemplificado na Figura 2.1, responsável por soldar a carroceria de um automóvel na linha de produção de uma indústria automobilística, a programação é feita de tal forma que o robô aplicará solda sempre nos mesmos pontos, partindo sempre de uma mesma posição de repouso. Caso o tipo de carroceria seja alterado, a programação do robô deverá ser adaptada para essa nova realidade.

No segundo subgrupo, estão os robôs autônomos com capacidade de tomada de decisão a partir de informações retiradas do seu próprio ambiente de atuação, através de sensores analógicos, tais como: sensor de luz, de som, de ultra-som, de contraste, de toque, etc. Essas informações fornecerão subsídios para que o

Figura 2.1: Exemplo de robô industrial

Fonte: Revista Robôs, n. 25, ano 2002

robô possa optar por uma determinada seqüência de passos de acordo com o valor captado pelo sensor.

Um exemplo desse tipo de robô é o aspirador de pó automático (Figura 2.2), já disponível em lojas de eletrodomésticos. Esse robô atua em um espaço delimitado e muda de direção sempre que encontra um obstáculo no ambiente em que está atuando. Ou seja, o valor recebido dos sensores determina a direção que o robô deve tomar nos próximos movimentos. Salienta-se que esse é apenas um exemplo das inúmeras possibilidades dentro dessa categoria de robôs. Outros exemplos de robôs autônomos podem ser vistos nas figuras 2.3 e 2.4.

2.1.2 Robôs controlados

Por outro lado, os robôs controlados são aqueles cuja programação se restringe apenas a um algoritmo interpretador de comandos. Comandos esses que são envi-

Figura 2.2: Exemplo de robô aspirador

Fonte: http://www.odebate.com.br/noticias.asp?ID=19975

Figura 2.3: Robô jardineiro

Fonte: http://www.inovacaotecnologica.com.br

ados por uma unidade controladora remota.

Nesse caso, o meio de conexão entre o robô e a estação controladora pode ser de diversos tipos, por exemplo: cabo, ultra-som, infravermelho ou rádio. Um exemplo desse tipo é o robô cirurgião, visto na Figura 2.5, onde está sendo controlado a partir de comandos dados pelos médicos cirurgiões.

2.1.3 O robô Monty

O robô, objeto do presente estudo, foi construído com base no microcontrolador PIC16F84, fabricado pela empresa MICROCHIP, e foi batizado de Monty.

O projeto do robô Monty é de autoria da editora espanhola F&G Editores que

Figura 2.4: Robô vigilante

Fonte: http://www.geocities.com/jcrobotics/gol_nort.pdf

Figura 2.5: Exemplo de robô cirurgião

Fonte: http://www.dw-world.de/brazil/0 3367 7115_A_1184292_1_A 00.html

o publicou, em vários países, por meio de uma obra em forma de fascículos semanais. Essa obra, que discorre em profundidade sobre as disciplinas listadas na Tabela 2.1, foi a principal base bibliográfica para este trabalho.

O Monty (ver Figura 2.6), usando toda a força do PIC16F84, conta com os dispositivos listados a seguir, montados em um conjunto de cinco placas de CI totalmente integradas.

- Dois sensores de toque, posicionados na parte dianteira, um em cada lateral do robô, estando conectados às linhas RB2 e RB3.
- Dois sensores óticos de raios infravermelhos por reflexão capazes de distin-

Fundamentos de eletrônica
Microprocessadores
Sensores e atuadores
Comunicações
Motores
Microcontroladores
Robótica industrial e inteligência artificial
Programação
Práticas de laboratório

Tabela 2.1: Disciplinas do curso de robótica

guir entre superfícies claras e escuras, estando conectados às linhas RB0 e RB1.

- Um sensor de ultra-som, conectado à linha RB6.
- Um sensor de limiar sonoro, conectado à linha RB7.
- Dois sensores de luz, conectados em paralelo à linha RB4.
- Um motor acionador da pinça controlado a partir da linha RB5.
- Dois motores conectados às linhas RA0,RA1,RA2 e RA3, responsáveis pelo movimento do robô.

A linha RA4 foi reservada para a utilização de um *encoder*, que poderia ser usado para controlar a velocidade do robô.

2.2 Diferença entre microcontrolador e microprocessador

Conforme (FGEDITORES, 2002), o microcontrolador é um computador construído dentro de um único circuito integrado, enquanto o microprocessador é apenas um dos componentes básicos de um computador. Uma outra descrição da diferença entre o microprocessador e o microcontrolador pode ser encontrada em (MATIC, 2000), onde podem ser encontradas especialmente muitas e importantes informações sobre os microcontroladores da família PIC.

Na Figura 2.7 é possível ver um diagrama de blocos simplificado de um computador. Nessa figura vê-se claramente que o microprocessador atua em conjunto com a memória e o módulo de entradas e saídas para prover as diversas funcionalidades existentes em um computador. Por outro lado, como pode ser visto na Figura 2.8, o microcontrolador possui em seu interior todos esses elementos totalmente integrados dentro de um único *chip*.

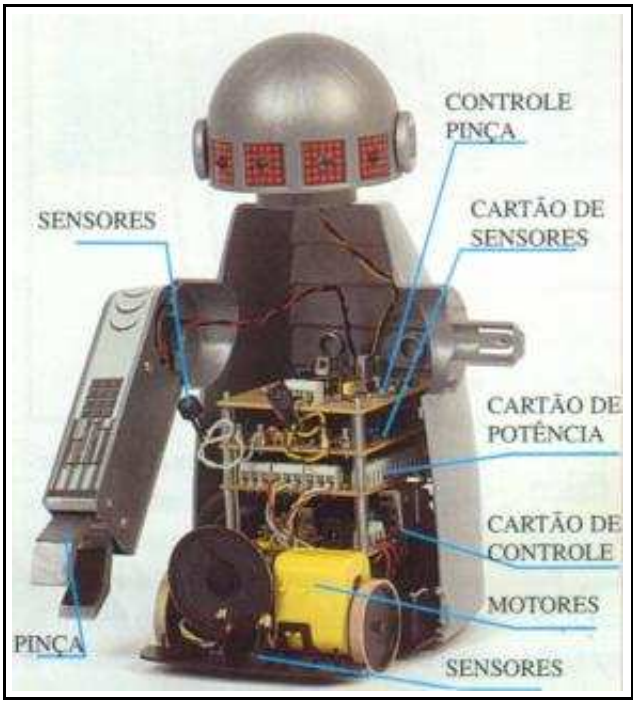

Figura 2.6: Robô Monty

Fonte: Revista Robôs, n. 53, ano 2002

Figura 2.7: Diagrama básico do computador

Fonte: Revista Robôs, n. 4, ano 2002

Figura 2.8: Componentes do microcontrolador

Fonte: Revista Robôs, n. 52, ano 2002

Os microcontroladores estão muito presentes no dia-a-dia das pessoas. Em (FGEDITORES, 2002), é apresentada uma estimativa segundo a qual no ano de 2000, em uma residência americana padrão, encontravam-se cerca de 240 microcontroladores. Esses pequenos computadores são encontrados, por exemplo, em máquinas de lavar, fornos microondas, aparelhos de TV, celulares, controles de iluminação, alarmes contra roubo, elevadores, alarmes de incêndio, ar-condicionados, em vários componentes de um automóvel e dentro do próprio computador ou em seus periféricos.

2.3 O microcontrolador PIC16F84

O PIC16F84 é um microcontrolador bastante versátil, possuindo, entre outras, as características listadas na Tabela 2.2.

Na Figura 2.9 é possível ver uma imagem do PIC16F84 em seu encapsulamento de 18 pinos, enquanto na Figura 2.10 pode-se ver a arquitetura básica desse microcontrolador.

O PIC16F84 pode tabalhar numa faixa de velocidades de 4 a 10MHz, sendo que a fase de busca de cada instrução dura 4 ciclos enquanto que a fase de execução dura mais 4 ciclos. Isso faria com que cada instrução consumisse 8 ciclos

Recurso	Valor
Memória de programa	1K x 14, tipo FLASH
Memória de dados RAM	68 bytes
Memória de dados EEPROM	64 bytes
Níveis de pilha	8
Jogo de instruções	35 de 14 bits
Tempo de execução das instruções normais	4 ciclos
Tempo de execução de salto	8 ciclos
Fontes de interrupção	4
Frequencia máxima de trabalho	10MHz
Linhas de E/S digitais	13
Temporizadores	um de usuário e um cão de guarda
Voltagem de alimentação	de 2 a 6VDC
Voltagem de gravação	de 12 a 14 VCD
Encapsulamento	DIP de 18 pernas

Tabela 2.2: Características do PIC16F84

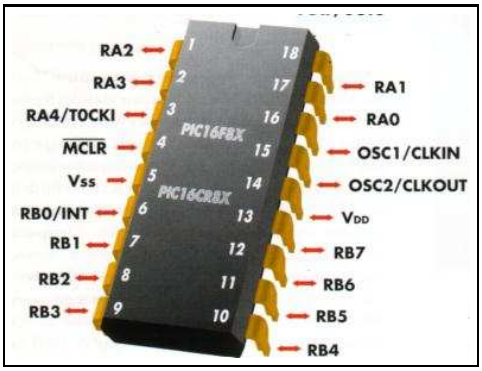

Figura 2.9: Encasulamento do PIC16F84

Fonte: Revista Robôs, n. 4, ano 2002

de máquina. Porém, graças ao seu paralelismo, o microcontrolador consegue executar duas fases simultaneamente, o que faz com que uma instrução normal seja executada em apenas 4 ciclos, enquanto que as instruções de salto só podem ser executadas realmente em 8 ciclos.

O programa a ser executado pelo PIC16F84 é armazenado em sua memória flash, que pode ser gravada a partir de um computador, no qual o PIC deve ser conectado através da interface serial ou da interface paralela. O processo de gravação exige ainda um *software* gravador, que é executado no computador, bem como um

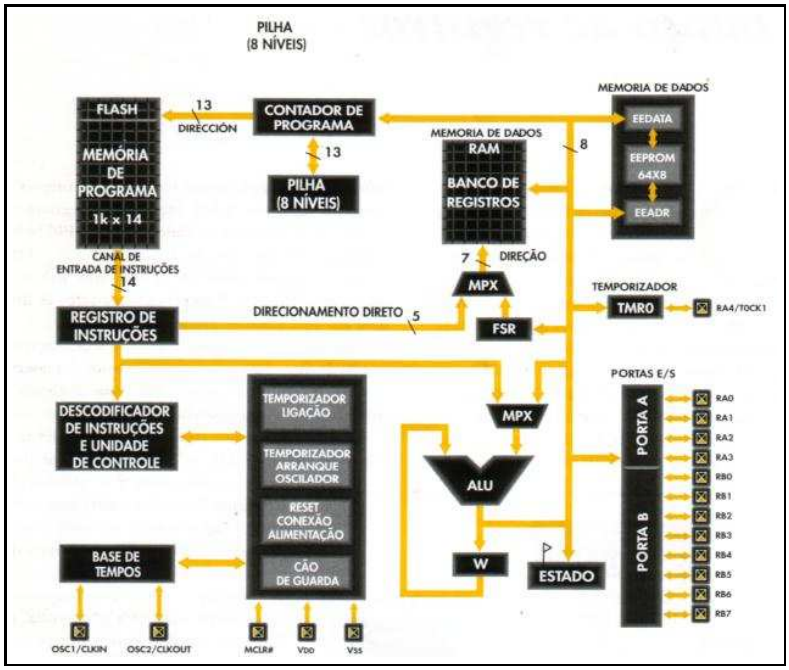

Figura 2.10: Arquitetura do PIC16F84

Fonte: Revista Robôs, n. 7, ano 2002

circuito eletrônico responsável por efetivamente gerar os comandos de gravação da memória *flash*. Na Internet é possível encontrar várias soluções de *hardware* para gravação dos microcontroladores da família PIC. Em (MADSEN, 2000) podem ser encontrados alguns exemplos. Um desses exemplos pode ser visto na Figura 2.11.

Em (PALONEN, 2003) podem ser encontradas mais informações sobre a utilização dos microcontroladores PIC a partir do sistema operacional Linux. Nesse *site* é possível encontrar *softwares* de gravação para Linux, compilador Assembly, além de uma série de informações no formato passo-a-passo sobre como gravar o PIC ou como compilar um programa.

2.4 Comunicação serial e os padrões RS232 e TTL

O padrão RS232 é utilizado pelo computador para estabelecer comunicação serial com periféricos a ele conectados. A comunicação serial pode se dar nos modos síncrono ou assíncrono.

Na comunicação serial assíncrona, as duas pontas a serem conectadas, chamados de *hosts*, devem conhecer previamente alguns parâmetros, que devem ser

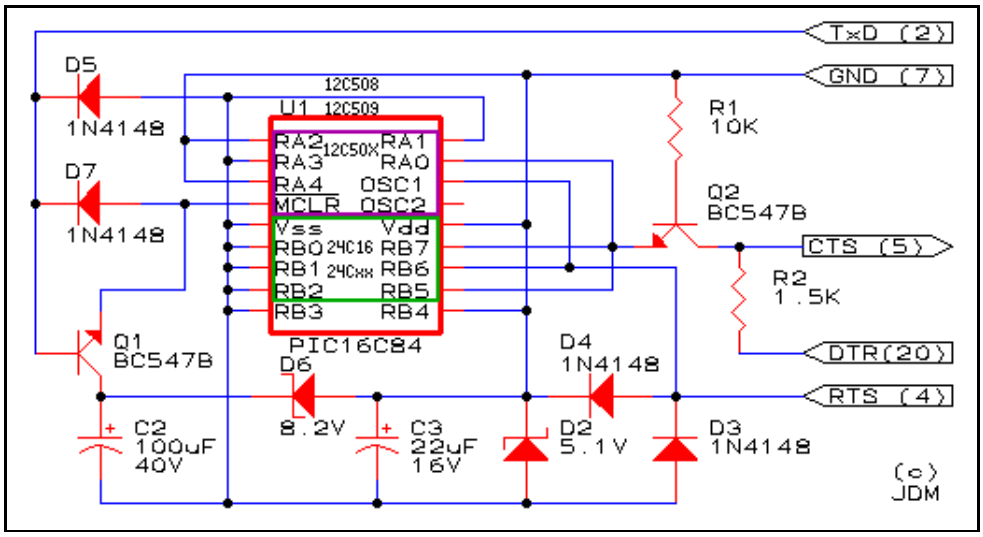

Figura 2.11: Exemplo de gravador do PIC

Fonte: http://www.jdm.homepage.dk/newpic.htm

configurados de forma idêntica em ambas as pontas, sob pena da comunicação não se realizar.

Ambas as pontas devem utilizar a mesma quantidade de *bits* de dados e de *stop*, assim como o mesmo tipo de *bit* de paridade. Um exemplo de configuração poderia ser 8N2, que significa: 8 *bits* de dados, 2 *bits* de *stop* e nenhum *bit* de paridade.

Um outro parâmetro muito importante no processo de comunição de dados é a velocidade de transmissão. Esse valor é medido em *bauds*, que é a quantidade de bits transmitida em cada intervalo de tempo em que trasmissor e receptor esperam que seja transmido de um novo conjunto de bits (CORREIA; SILVA, 2002). Para efeitos práticos, esse valor pode ser interpretado como 1 *bit* por segundo, considerando-se, portanto, que cada variação de tensão representará um único bit de dados. No padrão RS232, a velocidade de transmissão deve ser escolhida dentre um dos valores padrões em *bits* por segundo ou, simplesmente, bps: 330, 660, 1200, 2400, 4800, 9600, 19200, 38400, 57600 ou 115200 bps.

O RS232 define os seguintes níveis de voltagem para representar cada *bit* transmitido:

- Sinais de saída: +5 a +15VCD para o nível lógico alto e -5 a -15DCV para o nivel lógico baixo.
- Sinais de entrada: +3 a +15VCD para o nível lógico alto e -3 a -15DCV para o nivel lógico baixo.

O TTL, por sua vez, usado pelo robô, normalmente trabalha com os níveis de sinais mostrados abaixo.

- Sinais de saída: mínimo de +2 para o nível lógico alto e máximo de +0.8 para o nivel lógico baixo.
- Sinais de entrada: mínimo de +3.5 para o nível lógico alto e máximo de +0.8 para o nivel lógico baixo.

Em (CUGNASCA, 2002) são encontradas informações sobre comunicação serial apresentadas de forma bastante prática e objetiva. Mais informações podem ser encontradas em (LACERDA, 2002).

2.5 A linguagem LOGO

Segundo (CHELLA, 2002) e (SALVADOR, 2001), a linguagem LOGO foi criada por Seymourt Papert no final da década de 60 com a principal finalidade de ser uma linguagem de programação para crianças. Por ter como uma das principais características a simplicidade, a linguagem LOGO é muito utilizada em processos de aprendizagem de crianças. A partir de comandos simples de direcionamento, o usuário da linguagem pode comandar o que se conhece como uma tartaruga de solo.

Essa tartatura, que hoje normalmente é implementada como um simples sinal gráfico na tela do computador, inicialmente era uma contrução eletro-mecânica, ligada a um computador e capaz de receber deste, os comandos para sua movimentação. Isso possibilitava que o aprendiz constatasse, na prática, a relação entre os comandos enviados ao computador e o movimento da tartatura originado a partir desses comandos, facilitando, assim, o processo de aprendizagem.

Capítulo 3

DESENVOLVIMENTO DO HARDWARE

A parte de *hardware* do sistema de controle remoto do robô apresentado neste trabalho, consiste basicamente de três módulos:

- **Computador do tipo PC**: onde será a implementado o programa de controle do robô.
- **Robô**: construído sobre um microcontrolador PIC16F84.
- **Cabo conversor**: responsável por interconectar fisicamente os dois módulos anteriores, além de converter os níveis de sinais entre o padrão RS232, utilizado pelo computador, e o padrão TTL, utilizado pelo PIC16F84.

Nas seções a seguir serão descritos cada um desses três módulos.

3.1 O computador

O computador utilizado nas experiências foi um Pentium 133 MHz, com 48 Mb de memória RAM, gerenciado pelo sistema operacional Linux na sua distribuição Conectiva versão 8.

Para efeito do controle propriamente dito, os parâmetros velocidade e memória, mencionados no parágrafo anterior, não têm muita relevância, uma vez que a comunicação entre o computador e o robô se dá em baixa velocidade e, portanto, mesmo máquinas com pouca capacidade de processamento estarão aptas a fazer o controle.

Para a estação controladora apresentada neste modelo, é indispensável que ela possua: teclado, monitor e uma saída serial padrão RS232.

Para comunicação com o robô a porta serial do computador foi configurada no modo assíncrono, com a configuração 9600 8N2:

- Velocidade: 9600 bps
- *bit* de dados: 8
- *bit* de paridade: Nenhum
- *bit* de *stop*: 2

3.2 O robô

Para atingir os propósitos do presente trabalho, algumas alterações se fizeram necessárias nas conexões elétricas do robô. Foram desabilitados o controle de pinça e os sensores de luz. Essas entradas passaram, então, a ser utilizadas para as linhas de recepção e transmissão de dados entre o robô e o computador.

No projeto aqui apresentado, a gravação do programa na memória *flash* do PIC16F84 se deu através da própria placa controladora do robô Monty. Essa placa acumula duas funções que podem ser selecionadas a partir de uma chave comutadora específica. Com a chave de seleção na posição RUN, o PIC16F84 apenas executa o programa nele gravado. Por outro lado, com a chave na posição PROG, o microcontrolador passa a gravar em sua memória os dados recebidos da porta paralela do computador.

O MPLAB, software gravador fornecido juntamente com o robô Monty, funciona no sistema operacional DOS. Na Figura 3.1 pode-se ver uma imagem da tela principal desse programa. O processo de gravação do MPLAB se dá por meio da porta paralela do computador.

3.3 Interface de comunicação

O robô controlado é ligado à estação controladora através de um cabo flexível associado a um circuito de conversão de sinais entre os padrões utilizados pelo computador e pelo robô.

3.3.1 Circuito conversor entre RS232 e TTL

A utilização de diferentes voltagens para representar o mesmo nível lógico no padrão RS232 e no padrão TTL tornou necessário interpor, entre o robô e o computador, um circuito conversor de sinais. Esse circuito, baseado no *chip* MAX232, de fabricação da Texas Instrument (TEXASINSTRUMENT, 2002), recebe os sinais do robô no padrão TTL e entrega ao computador o mesmo sinal no padrão RS232 e vice-versa. Na Figura 3.2 pode ser vista a configuração padrão desse circuito.

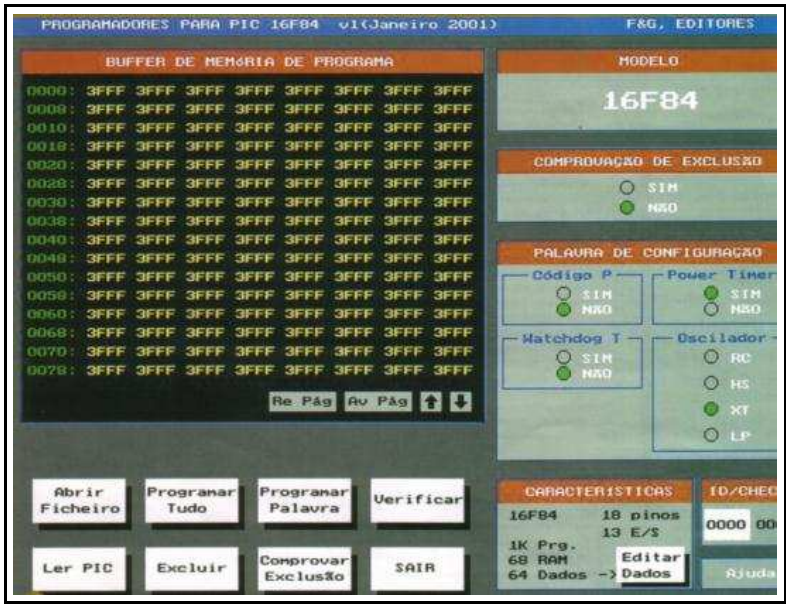

Figura 3.1: Tela do programa MPLAB

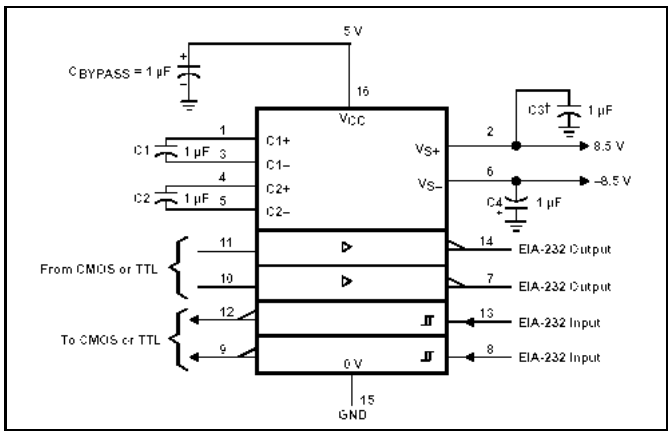

Figura 3.2: Circuito conversor baseado no MAX232

Fonte: TEXAS Instrument

3.4 Conexões

No lado do computador foi utilizado um conector DB9, do qual somente foram utilizados os pinos 2, 3 e 5, conectados, no circuito conversor, aos pinos de transmissão, recepcão e terra, respectivamente. Essa inversão dos pinos 2 e 3 é neces-

sária, visto que a linha de transmissão da saída do computador deve ser ligada à linha de recepção do MAX232 e vice-versa.

No lado do robô, os fios de transmissão e recepção foram ligados diretamente às linhas RB5 e RB4 da placa de controle, promovendo, assim, a mesma inversão que foi realizada no conector do computador. De forma que, do ponto de vista do PIC16F84, a linha RB4 é de transmissão, enquanto que a linha RB5 é de recepção.

Na Figura 3.3 pode ser vista uma representação esquemática do cabo conversor, onde pode-se perceber, inclusive, que a alimentação do circuito conversor foi retirada do próprio robô.

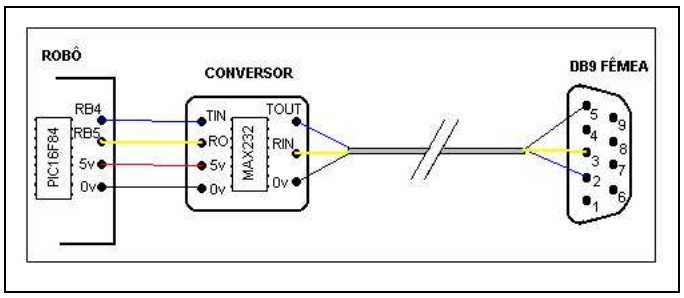

Figura 3.3: Cabo conversor

Capítulo 4

DESENVOLVIMENTO DO SOFTWARE

4.1 O software do PIC16F84

O programa do PIC16F84, cuja arquitetura pode ser vista na Figura 4.1, foi desenvolvido em Assembly e apresenta três funcionalidades que serão detalhadas nas três subseções seguintes.

No programa do robô foi utilizado o conceito de primitiva: um conjunto simples e reduzido de comandos que devidamente agrupados podem criar ações mais complexas. São reconhecidas pelo robô as seguintes primitivas: <A>, <P>, <E>, <D>, <R>, <S>, <s> e <V>. A primitiva <V> retorna a versão corrente do programa do PIC. As demais primitivas serão detalhadas a seguir.

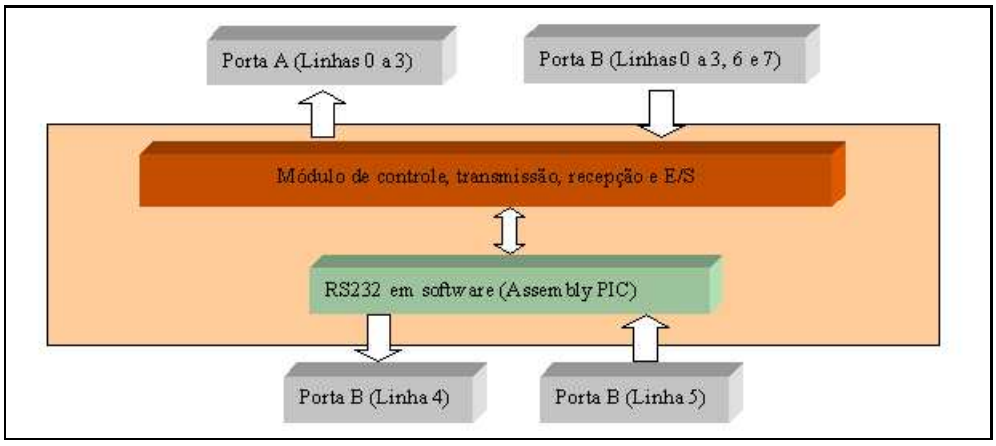

Figura 4.1: Arquitetura do programa do robô

4.1.1 Acionamento dos motores

O robô Monty possui dois motores que são responsáveis pelo seu movimento. Cada um desses motores é controlado por duas linhas digitais do PIC16F84. Caso as duas linhas apresentem em sua saída valores iguais, o motor interromperá o seu funcionamento. Se apenas uma das linhas apresentar valor 1, então o motor será acionado em um ou outro sentido, dependendo de qual linha foi acionada. Na Tabela 4.1 pode ser vista a correlação entre o movimento do motor e os valores presentes nas linhas digitais RB0 e RB1.

RB0	RB1	Motor
0	0	Para de girar
0	1	Gira para trás
1	0	Gira para frente
1	1	Para de girar

Tabela 4.1: Controle do motor

As primitivas aceitas pelo robô, para controle dos motores, podem ser vistas na Tabela 4.2.

Primitiva	Comando	Descrição
<A>	Avançar	Os dois motores giram para frente
<E>	Esquerda	O motor esquerdo gira para trás enquanto o motor direito gira para frente
<D>	Direita	O motor direito gira para trás enquanto o motor esquerdo gira para a frente
<R>	Retornar	Os dois motores giram para trás
<P>	Parar	Os dois motores param

Tabela 4.2: Primitivas para controle dos motores do robô

4.1.2 Leitura dos sensores

Com as adaptações efetuadas no Monty especificamente para este trabalho, o robô passou a gerenciar os seguintes sensores:

1. ***Bumpers* esquerdo e direito**. Sensores de toque que geram em sua saída o valor 1 sempre que são acionados, ou seja, sempre que o robô tocar em um obstáculo à sua esquerda ou à sua direita.

2. **Sensores de contraste esquerdo e direito**. Sensores de reflexão capazes de diferenciar entre superfícies claras e escuras. Sempre que o sensor pas-

sar sobre uma superfície de cor preta, será gerado o valor 1 em sua saída. Este sensor é indispensável na tarefa de seguir uma determinada trajetória marcada no solo.

3. **Ultrasom**. Sensor capaz de detectar movimentos. Sempre que houver movimento à frente do robô, este sensor gerará o valor 0 em sua saída.

4. **Limiar sonoro**. Sensor responsável por detectar um sinal sonoro acima de uma determinada intensidade. Quando detectado um determinado volume de som, este sensor retornará o valor 1.

O programa do robô é capaz de informar os valores desses sensores. Isso se dá através das primitivas <S> e <s>. Os possíveis valores retornados em resposta à diretiva <S> podem ser vistos na Tabela 4.3. Havendo mais de um valor, eles serão separados por ",". Na Figura 4.2 pode ser vista a *string* retornada após o recebimento da diretiva <s> e o significado de cada dígito nessa *string*.

Sinal	Sensor	Linha do PIC
BEsq	O bumper esquerdo	B1
BDir	O bumper direito	B0
CEsq	Contraste Esquerdo	B2
CDir	Contraste Direito	B3
USom	Ultra-som	B6
Som	Limiar sonoro	B7

Tabela 4.3: Sinais dos sensores retornados pelo robô

```
<100101>
  ||||||Som
  |||||USom
  ||||BDir
  |||BEsq
  ||Cdir
  |CEsq
```

Figura 4.2: Resposta ao comando <s>

4.1.3 Comunicação com o computador

Foi utilizado ainda um módulo de comunicação serial sem o qual não seria possível a transmissão de dados entre o computador e o microcontrolador. Essa necessidade

se deu pelo fato do PIC16F84 não possuir em seu *hardware* uma porta específica para comunicação serial. A implementação desse módulo foi baseada na biblioteca rs232low, distribuída com robô Monty.

A configuração dos parâmetros de comunicação serial do robô foi fixada dentro do próprio fonte. Os parâmetros utilizados foram 8N2, ou seja, 8 *bits* de dados, nenhum *bit* de paridade e 2 *bits* de *stop*.

O módulo rs232low exporta simplesmente duas funções, além de algumas variáveis. As duas funções, RxD e TxD, são responsáveis, respectivamente, por receber e enviar um caractere para o computador.

4.1.4 Descrição dos principais trechos do programa

Na Figura 4.3 pode ser visto um diagrama de fluxo do programa do robô. A seguir serão apresentados os principais trechos do programa.

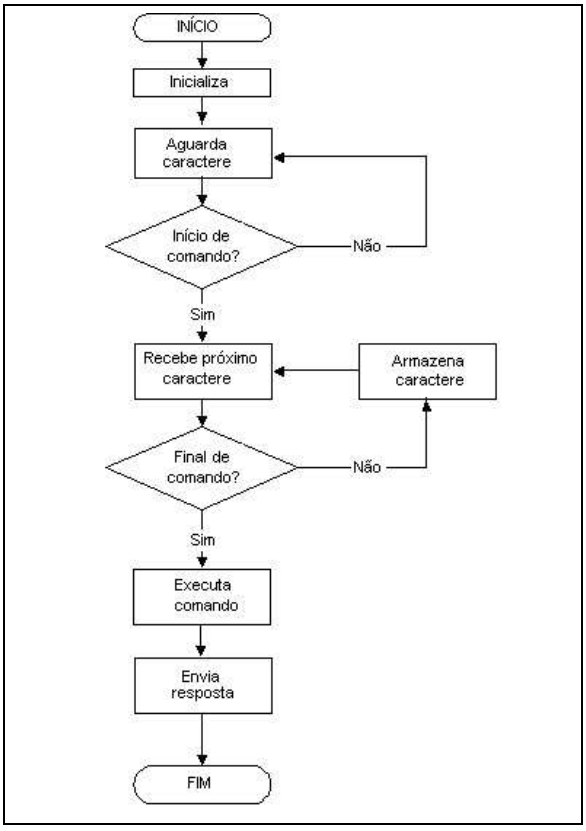

Figura 4.3: Diagrama de fluxo do programa do robô

Após a inicialização dos registradores de interrupção e de temporização, o programa entra em seu *loop* principal, cujo código fonte é mostrado na Figura 4.4. Na

linha 1, o temporizador "cão de guarda" é zerado para evitar que o PIC16F84 seja reinicializado em situação normal de execução. Nas linhas 2 a 4, é enviado o caractere de pronto para a estação controladora. Na linha 5, o programa aguarda até que receba um caractere qualquer do computador. Nas demais linhas, o programa verica se o primeiro caractere recebido é um <, significando início de comando. Em caso afirmativo, desvia para bloco RECEBE_COMANDO, caso contrário, retorna ao início do *loop*.

```
  AGUARDA_COMANDO

1               clrwdt                  ; atualiza cao de
2               movlw   '>'             ; envia sinal de p
3               movwf   Txdreg
4               call    TxD             ;

5               call    RxD             ; aguarda primeiro

6               movlw   '<'             ; é o caractere de
7               subwf   Rxdreg,W
8               btfsc   STATUS,Z
9               goto    RECEBE_COMANDO  ; sim, então receb

10              goto    AGUARDA_COMANDO
```

Figura 4.4: Loop principal do programa

Na função RECEBE_COMANDO (Figura 4.5), o programa recebe do computador o restante do comando. Caso seja um comando válido, chama a função EXECUTA_COMANDO.

Inicialmente, nas linhas 1 a 3, há a preparação do controle do *buffer* de recepção. Na linha 4, o programa aguarda a chegada de um novo caractere. Nas linhas 5 a 8, é verificado se o caractere recebido é o indicador de fim de comando (caractere >). Em caso positivo, desvia para o bloco EXECUTA_COMANDO. Nesse momento, o comando estará armazenado na variável Buffer. Nas linhas 9 a 12 é verificado o limite do *buffer*. Caso esteja tudo certo, nas linhas 13 e 14 o caractere é armazenado no *buffer*. Finalmente, nas linhas 15 a 17, o ponteiro do *buffer* é incrementado e o programa passa a esperar um novo caractere.

Na Figura 4.6 pode ser visto um trecho do bloco EXECUTA_COMANDO, onde, para cada primitiva reconhecida pelo programa, é executado um conjunto de comandos similares àqueles que se encontram nas linhas 1 a 4. Uma exceção é o bloco de tratamento do comando <S>, que pode ser visto na Figura 4.7. Nas linhas 5 a 16 são testadas todas os sinais de entrada do PIC16F84, sendo chamada a rotina correspondente a cada sinal ativo.

```
   RECEBE_COMANDO
 1             clrf    Contador
 2             movlw   Buffer
 3             movwf   FSR
   RECEBE_CAR
 4             call    RxD             ; aguarda caractere
 5             movlw   '>'             ; é o caractere de fim de comando?
 6             subwf   Rxdreg,W
 7             btfsc   STATUS,Z
 8             goto    EXECUTA_COMANDO ; sim, então recebe o comando
 9             movf    Contador,W      ; Estourou o buffer?
10             sublw   TamMaxBuffer
11             btfsc   STATUS,Z
12             goto    ERRO            ; sim, chama erro e retorna
13             movf    Rxdreg,W        ; não, armazena o car e espera próximo
14             movwf   INDF            ;
15             incf    Contador,F
16             incf    FSR,F
17             goto    RECEBE_CAR
```

Figura 4.5: Bloco RECEBE_COMANDO

```
   AVANCA
1              movlw   b'00000110'     ; avança
2              movwf   PORTA

3              call    EnviaOk
4              goto    AGUARDA_COMANDO

   ESQUERDA
               movlw   b'00000101'     ; avança para a esquerda
               movwf   PORTA

               call    EnviaOk
               goto    AGUARDA_COMANDO
```

Figura 4.6: Trecho do bloco EXECUTA_COMANDO

4.2 O software do computador

Para controlar o robô remotamente, foi desenvolvido um programa em C, que será executado no computador e cuja arquitetura pode ser vista na Figura 4.8. O diagrama de fluxo desse programa pode ser visto na Figura 4.9.

A arquitetura da aplicação foi projetada de forma modular com o objetivo principal de permitir futuras expansões com um mínimo de alterações no programa. Assim, por exemplo, todo o tratamento de comunicação em baixo nível entre o robô e o computador foi isolado em módulos específicos, sendo um para cada tipo de comunicação. Cada módulo deve implementar e exportar as funções exigidas pelo módulo de controle.

Nas subseções a seguir serão apresentados os principais módulos do programa.

```
   SENSORES

 1              clrf    Eventos
 2              movlw   '<'
 3              movwf   Txdreg
 4              call    TxD

 5              btfsc   PORTB,1
 6              call    EvContrasteEsq
 7              btfsc   PORTB,0
 8              call    EvContrasteDir
 9              btfsc   PORTB,2
10              call    EvBumperEsq
11              btfsc   PORTB,3
12              call    EvBumperDir
13              btfss   PORTB,6
14              call    EvUltraSom
15              btfsc   PORTB,7
16              call    EvSom

17              movlw   '>'
18              movwf   Txdreg
19              call    TxD
20              call    EnviaEnter

21              goto    AGUARDA_COMANDO
```

Figura 4.7: Trecho do bloco de tratamento do comando <S>

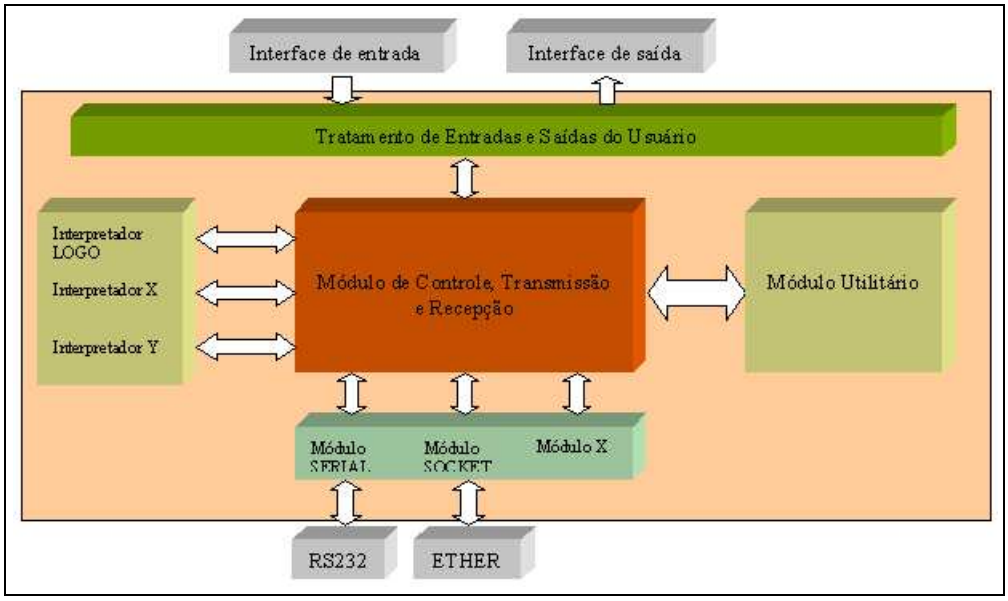

Figura 4.8: Arquitetura do programa do computador

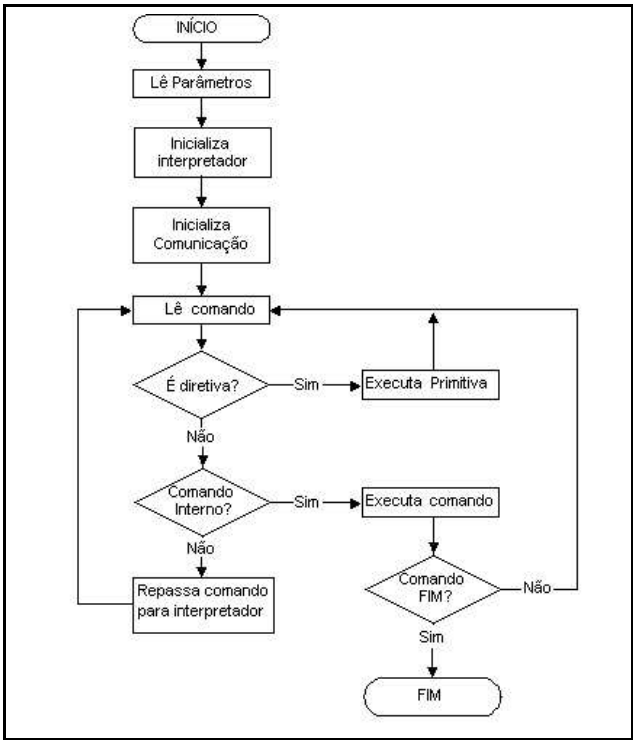

Figura 4.9: Diagrama de fluxo do programa do computador

4.2.1 Módulo de controle

O módulo de controle é responsável por fazer a integração entre todos os demais módulos. Na Figura 4.10 pode ser visto o *loop* principal desse módulo.

O comando fornecido pelo usuário é capturado pelo módulo de interface, que o entrega para o módulo de controle (linhas 3 e 4). Este, por sua vez, tentará executá-lo caso seja um comando interno ou primitiva (linhas 6 a 24). Em caso contrário, executará uma função do módulo interpretador (linhas 25 a 27) que se encarregará de transformar os comandos em primitivas, devolvendo-as, em seguida, ao módulo de controle, que as encaminhará ao módulo de comunicação.

As funções ExecutaComando e ExecutaPrograma (Figura 4.11) funcionam apenas com um *wrapper* de roteamento para a real função no módulo interpretador. Uma vez que podem existir vários módulos interpretadores no programa, o usuário deverá especificar o interpretador com o qual deseja trabalhar, sendo que o interpretador *default* é o LOGO. Essas funções, então, enviarão o comando para o módulo correspondente ao interpretador selecionado pelo usuário.

A função de inicialização do interpretador, chamada no momento da definição da linguagem a ser utilizada, passa para o módulo interpretador o endereço da fun-

```
1       do
2       {
3               Comando[0]=0;
4               Result=PegaComando(Comando);
5
6               if (Comando[0]=='<')
7                       ExecutaPrimitiva(Comando,Resp);
8               else {
9                       CodCmd=IdentificaComando(Comando,Param);
10                      switch(CodCmd)
11                      {
12                      case C_LING:
13                              DefineLinguagem(Param);
14                      break;
15                      case C_PROT:
16                              DefineProtocolo(Param);
17                      break;
18                      case C_EXEC:
19                              ExecutaProg(Param);
20                      break;
21                      case C_FIM:
22                              Finaliza();
23                              exit;
24                      break;
25                      default:
26                              ExecutaComando(Comando);
27                      break;
28              }
29      } while (Result);
```

Figura 4.10: Loop principal do módulo de controle do programa do computador

```
1       int ExecutaComando(char *Comando)
2       {
3               switch(gLinguagem)
4               {
5                       case L_LOGO:
6                               LOGO_ExecutaComando(Comando);
7                       break;
8               }
9       }

10      int ExecutaProg(char *Prog)
11      {
12              switch(gLinguagem)
13              {
14                      case L_LOGO:
15                              LOGO_ExecutaProg(Prog);
16                      break;
17              }
18      }
```

Figura 4.11: Funções ExecutaPrograma e ExecutaComando

ção ExecutaPrimitiva. Isso permite que as funções vistas na Figura 4.11 chamem a função ExecutaPrimitiva em forma de *callback*. Dessa maneira, toda a inteligência de execução das primitivas fica restrita ao módulo de controle, que atuará em conjunto com o módulo de comunicação para comandar a efetiva execução da primitiva pelo robô.

4.2.2 Módulo interpretador

Um módulo interpretador diferente poderá ser desenvolvido para cada linguagem que se deseje suportar no programa controlador. Neste projeto, foi implementado, a título de exemplo, apenas um subconjunto mínimo de comandos da linguagem LOGO. Qualquer módulo interpretador deve implementar e exportar as seguintes funções:

- **Inicializa**: Função chamada pelo módulo de controle no momento da definição da linguagem utilizada. O implementador do módulo deve aproveitar esta função para promover todas as inicializações necessárias ao interpretador. Uma inicialização obrigatória é o armazenamento, em uma variável local, do ponteiro para a função ExecutaPrimitiva, que será utilizada pelas funções ExecutaComando e ExecutaProg.

- **ExecutaProg**: Chamada pelo módulo de controle quando este receber um comando EXEC. Esta função deve abrir o arquivo cujo nome é passado como parâmetro e executá-lo segundo seu próprio engenho. Para cada comando que necessitar executar uma das primitivas do robô, deverá ser chamada a função ExecutaPrimitiva, implementada pelo módulo de controle. Um ponteiro para essa função é passado na chamada da função Inicializa.

- **ExecutaComando**: Esta função é chamada pelo módulo de controle sempre que este receber um comando que não seja uma primitiva nem um comando interno. Aqui, como na função ExecutaProg, caso seja necessário executar uma primitiva, a função deverá chamar, em forma de *callback*, a função ExecutaPrimitiva do módulo de controle.

- **Finaliza**: Chamada na finalização normal do programa. Pode ser usada para fazer todo o trabalho de "limpeza" de memória e/ou outras ações necessárias para que tudo seja finalizado corretamente.

Na figura 4.12 pode ser vista a função ExecPF, onde é executado o comando PF da linguagem LOGO. Observe-se nas linhas 9 e 12 a chamada à função ExecPrimitiva. Na linha 9 é executada a primitiva <A> fazendo os dois motores girarem para frente. Em seguida, na linha 11, é executado um *delay*. Finalmente, na linha 12, os motores são desligados por meio da execução da primitiva <P>.

Para se determinar o tempo de *delay* usado na chamada da função sleep, foi usada a fórmula 4.1:

$$TempoDelay = \frac{(N * K)}{50} \tag{4.1}$$

Onde N, dado em centímetros, é o parâmetro recebido pela função ExecPF, e K é o tempo, em segundos, necessário para o robô percorrer 50cm. O valor de K é de difícil precisão devido a pelo menos duas variáveis:

```
1       int ExecPF(char * Param)
2       {
3           int Qtd;
4           char Resp[TAM_MAX_RESP];
5           if (Param)
6           {
7               sscanf(Param,"%d",&Qtd);
8               Resp[0]=0;
9               (*pExecPrim)(P_S_AVANCAR,Resp);
10              if (strncmp(Resp,"<OK>",4)!=0) return(0);
11              sleep(DELAY(Qtd));
12              (*pExecPrim)(P_S_PARAR,Resp);
13              if (strncmp(Resp,"<OK>",4)!=0) return(0);
14              return(1);
15          }
16          return(0);
17      }
```

Figura 4.12: Função ExecPF

1. **A superfície onde o robô está atuando**: caso a superfície se apresente demasiadamente lisa, poderá haver deslisamento das rodas, o que provocará uma diferença no valor.

2. **A intensidade da fonte de alimentação**: dependendo da força da alimentação, os motores poderão girar mais ou menos rápido, gerando, assim, consideráveis diferenças nos tempos.

De igual forma, para gerar o *delay* necessário às funções ExecPE e ExecPD, cujo parâmetro é um ângulo, foi utilizada a fórmula 4.2:

$$TempoDelay = \frac{(N * K)}{360} \tag{4.2}$$

Onde N é o ângulo passado no parâmetro de chamada da função, em graus, e K é uma constante que representa o tempo necessário para o robô girar 360^o.

4.2.3 Módulo de comunicação

Da mesma forma que no módulo interpretador, pode existir um módulo para cada tipo de comunicação. Neste trabalho foi implementado apenas o módulo de comunicação serial, baseado, principalmente, nas informações contidas em (SWEET, 2003). Cada módulo de comunicação deve implementar e exportar as seguintes funções:

- **Inicializa**: Função chamada pelo módulo de controle no momento da definição do protocolo utilizado. Aqui devem ser colocadas todas as ações necessárias à inicialização da comunicação com o robô. Por exemplo, a abertura da porta serial cuja identificação é passada como parâmetro.

- **EnviaPrimitiva**: Chamada pelo módulo de controle quando este receber uma solicitação de execução de uma primitiva. Esta função deve enviar a primitiva para o robô e, em seguida, receber a resposta do robô para repassá-la ao módulo de controle.

- **Finaliza**: Chamada na finalização normal do programa. Pode ser usada para executar procedimentos de finalização da comunicação assim como liberação de variáveis alocadas dinamicamente.

Na Figura 4.13 pode ser vista a função EnviaPrimitiva. Na linha 4, a primitiva é enviada para o robô, que deverá estar conectado à porta serial apontada pelo *handle* hPorta. Se tudo correu bem no envio da primitiva (linhas 5 a 9), então o programa aguardará a resposta do robô (linha 10), que será repassada ao módulo de controle.

```
1       int SERIAL_EnviaPrimitiva(char *Primitiva, char *Resposta)
2       {
3               int n;
4
5               n = EnviaStr(Primitiva);
6               if (n < 0) return(0);
7
8               n = RecebeStr(Resposta);
9               if (n>0) return(1);
10
11              return(0);
12      }
```

Figura 4.13: Função EnviaPrimitiva

4.3 Funcionamento do sistema

4.3.1 Conexão do robô à porta serial do computador

Antes de qualquer coisa, é necessário conectar o robô à porta serial do computador. Essa conexão é feita por meio do cabo conversor, cuja extremidade do computador é dotada de um conector DB9.

Normalmente, a primeira porta serial (/dev/ttyS0), que é acessada através de um conector DB9, é usada pelo *mouse*, restando, portanto, a porta serial 2. Em muitos computadores, entretanto, a segunda porta é dotada de um conector DB25, o que exige a utilização de um adaptador DB9(macho) x DB25(fêmea).

4.3.2 Testes da conexão

Para testar a conexão com o robô, pode ser utilizado o programa minicom ou qualquer outro programa de emulação de terminal serial.

A configuração do minicom é muito simples. Uma vez iniciado o programa, basta digitar "CTRL-A O" para acessar o menu de configuração. A partir desse menu, selecionar a opção "Configuração da Porta Serial" (Figura 4.14). Nesse menu, devem ser configuradas:

- Dispositivo Serial (Opção A): Para a serial 2, informar "/dev/ttyS1".
- Bps/Paridade/Bits (Opção E): Devem ser selecionados os valores mostrados na Figura 4.15.
- Controle de Fluxo por *Hardware* (Opção F): Deverá ser escolhida a opção "Não".
- Controle de Fluxo por *Software* (Opção G): Deverá ser escolhida a opção "Não".

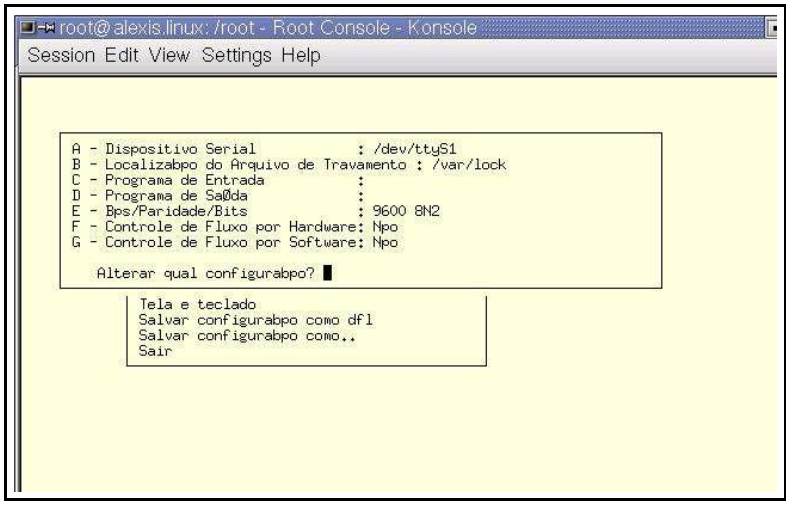

Figura 4.14: Menu principal do minicom

Depois de configurado o minicom, basta ligar o robô, colocando a chave "ON-OFF" na opção "ON". Salienta-se que a chave "RUN-PROG" deve estar na posição "RUN". Se tudo estiver corretamente configurado, deverá aparecer na tela do minicom a saudação inicial do robô, que consiste de duas linhas. Na primeira linha deverá surgir a versão do programa do robô, enquanto que na segunda linha deverá aparecer a string <OK>. Finalmente, deverá aparecer o sinal de pronto (caractere >). A partir desse momento, o robô estará apto a receber comandos.

4.3.3 Utilização do programa robo

Para executar o programa no computador basta executar o comando **robo** na linha de *prompt* do Linux. A sintaxe do comando pode ser vista a seguir:

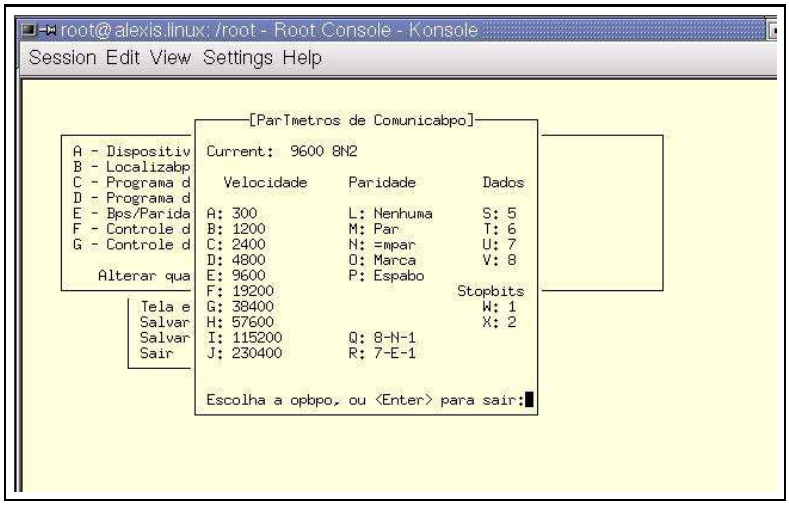

Figura 4.15: Confi guração de Bps/Paridade/Bits da porta seiral

robo [ling logo][prot serial:/dev/ttyS1]

Os dois parâmetros são opcionais. Entretanto, caso o parâmetro **prot** não seja informado na execução do programa, o usuário deverá executar o comando **prot**, dentro do programa, visto que, enquanto não for definido um protocolo válido, não será possível estabelecer comunicação com o robô.

Com relação ao parâmetro **ling**, que define o interpretador a ser utilizado pelo programa, o usuário poderá interagir com o robô, mesmo sem defini-lo, uma vez que, por *default*, é assumida a linguagem LOGO.

Nessa primeira versão do programa, o único protocolo aceito é o **serial**, enquanto que a única linguagem suportada é o LOGO. No caso da linguagem, foi implementado apenas um subconjunto dos comandos do LOGO.

Dessa forma, após executar o programa, o usuário poderá digitar:

- Um dos comandos internos mostrados na Tabela 4.4,
- Uma das primitivas do robô, mostradas na Tabela 4.5 ou
- Um dos comandos suportados pelo interpretador carregado no momento. No caso do interpretador LOGO implementado aqui, somente são suportados os comandos listados na Tabela 4.6.

Comando	Significado
PROT <protocolo>:<device>	Define um protocolo
LING <linguagem>	Define uma linguagem
EXEC <arquivo>	Solicita que o interpretador execute o programa contido em <arquivo>
FIM	Finaliza o programa

Tabela 4.4: Comandos internos

Comando	Significado
<A>	Avança
<E>	Gira para a esquerda
<D>	Gira para a direita
<R>	Retorna
<P>	Para os motores
<V>	Informa a versão
<S>	Retorna a lista de sensores que está recebendo um evento externo
<s>	Retorna o status de todos os sensores

Tabela 4.5: Primitivas

Comando	Significado
PF <N>	Avança N centímetros
PE <N>	Gira N graus para a esquerda
PD <N>	Gira N graus para a direita
PT <N>	Retorna N centímetros
REPITA <N>[<comando>]	Repete <comando> N vezes
DIGA <X>	Escreve o resultado da primitiva X (V, S ou s)

Tabela 4.6: Comandos da linguagem LOGO

4.3.4 Testes de velocidade de transmissão

Nessa fase de testes constatou-se a necessidade de se colocar uma pausa após o envio de cada caractere para o robô. Essa necessidade explica-se pelo fato do robô não suportar controle de transmissão por *software* ou por *hardware*. Dessa forma, o programa do computador deve aguardar o tempo necessário para que o robô processe o caractere anteriormente enviado. Esse tempo foi calculado com base na fórmula:

$$Pausa = (InstrNorm * TempoInstrNorm) + (InstrSalto * TempoInstrSalto) \quad (4.3)$$

onde:

- InstrNorm: Quantidade de instruções normais no bloco de tratamento de um caractere.
- TempoInstrNorm: Tempo necessário para execução de uma instrução normal.
- InstrSalto: Quantidade de instruções de salto no bloco de tratamento de um caractere.
- TempoInstrSalto: Tempo necessário para execução de uma instrução de salto.

Para se obterem os valores a serem aplicados na equação 4.3, foram assumidas as seguintes constantes:

- Velocidade de processamento do PIC16F84: 4MHz.
- Ciclos necessários para execução de uma instrução normal: 4.
- Ciclos necessários para execução de uma instrução de salto: 8.
- Quantidade de instruções normais no bloco de recepção: 11.
- Quantidade de instruções de salto no bloco de recepção: 2.

Aplicando-se então as constantes acima no equação 4.3, foi obtido o seguinte valor para a variável Pausa:

$$Pausa = (10 * 1\mu) + (3 * 2\mu s) = 16\mu s \quad (4.4)$$

A esse valor foi acrescentada uma margem de segurança de 2μs. Dessa forma, cada primitiva, cujo tamanho é de 3 caracteres, será transmitida para o robô em aproximadamente 54μs.

Capítulo 5

CONCLUSÃO

5.1 Resultados

Nos vários testes realizados, o sistema de controle remoto do robô mostrou-se bastante estável. Inicialmente foram utilizados programas emuladores de terminal serial. No ambiente gráfico do Linux (KDE) foi utilizado o terminal do kppp, enquanto no modo texto utilizou-se o minicom. Ambos se mostraram bastante fáceis de configurar.

Nesses primeiros testes, as diretivas foram digitadas diretamente no terminal, que as enviava para o robô. As respostas do robô eram capturadas e exibidas pelo próprio terminal. Esse método de testes possibilitou identificar a necessidade de vários ajustes na programação do robô. Nessa fase, foram realizadas cerca de 50 gravações no microcontrolador.

Na segunda fase de testes, já com o programa do computador em funcionamento, foram realizados testes de execução em bloco, interpretação da linguagem de comandos e, principalmente, o envio de caracteres em uma alta taxa de transmissão de forma a determinar a real capacidade de recepção do robô. Nessa fase, conforme detalhado no capítulo 4, foi necessário acrescentar uma pausa após a transmissão de cada caractere, o que não inviabilizou o funcionamento do sistema.

Na Figura 5.1 pode ser visto o robô conectado ao computador, mas ainda sem a carcaça. Na Figura 5.2, já com o robô dentro da carcaça, está sendo feita a medição dos tempos para o cálculo da sua velocidade.

5.2 Discussão

A partir deste trabalho, constatou-se a relativa facilidade de se construir um sistema de controle de um robô, desde que este seja capaz de se comunicar por meio de um protocolo de comunicação padrão, tal como o protocolo serial.

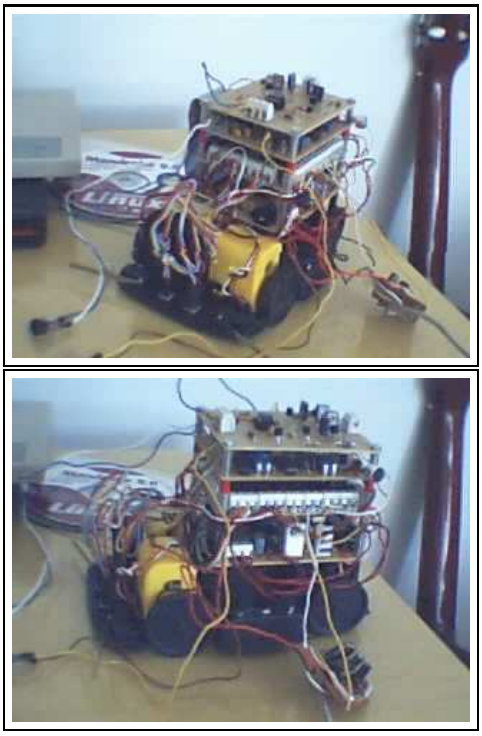

Figura 5.1: Robô na fase de testes

Figura 5.2: Medição da velocidade do robô

As ferramentas necessárias para se construir o programa de controle são facilmente encontradas na forma de *software* livre. Neste trabalho, por exemplo, foi utilizada a linguagem C, cujo compilador normalmente acompanha as distribuições Linux.

A construção do programa do robô baseado no microcontrolador PIC, embora

mais complexa do que a programação do computador, também é amplamente documentada tanto na Internet como em livros especializados. Uma combinação dessas duas fontes é o livro (MATIC, 2000), que pode ser encontrado em meio eletrônico gratuitamente na Web.

A construção de um circuito eletrônico para gravar o microcontrolador pode ser o ponto considerado mais complexo na implementação do sistema apresentado, mas também podem ser encontrados vários exemplos desse circuito na Internet.

Uma alternativa para fugir do desenvolvimento do programa para o PIC e, consequentemente, da sua gravação, é buscar soluções tais como o BasicStep, do fabricante brasileiro Tato Equipamentos. Esse produto, cuja descrição básica pode ser vista em (ANGNES, 2003), é uma solução integrada de microcontrolador, circuito de *hardware* para gravação, linguagem de programação e IDE para desenvolvimento e gravação do software.

5.3 Proposta de continuidade do projeto

São muitas as possibilidades de continuidade do projeto. Entre elas, destaca-se a possibilidade de desenvolvimento de um driver que substituiria parte do programa de controle. O driver poderia implementar também um *handler* para tratamento de interrupções vindas do robô.

Também poderia se pensar em construir um interpretador de comandos mais completo. Este programa, utilizando recursos de inteligência artificial, seria capaz de utilizar os sinais captados pelos sensores do robô para promover ações mais complexas como:

- Seguir uma trajetória predeterminada, desviando dos obstáculos encontrados no percurso ou
- "Varrer" um determinado ambiente à procura de determinado objeto.

Se forem consideradas mudanças na parte do robô, as possibilidades são inúmeras. Por exemplo, poderia ser implementada uma comunicação via rádio, o que traria uma maior liberdade de movimentos para o robô. Também pode se pensar em implementar uma comunicação sobre o protocolo IP, o que garantiria o controle do robô a partir de qualquer ponto de uma rede. Já existe, inclusive, projetos de implementação de uma versão simplificada da pilha TCP/IP no PIC16F84. Uma lista bastante variada, contendo esse e outros projetos baseados no PIC, pode ser encontrada na Internet, em (AID, 2003).

Outro incremento possível seria a instalação de uma microcâmera de forma que o robô pudesse transmitir seu posicionamento para a estação de controle. Nesse caso, o microcontrolador deveria ser substituído por uma versão mais potente como o PIC16F87.

O projeto do robô poderia ainda evoluir para um controlador PLC. Assim, o programa do computador poderia ser, na verdade, um painel de controle de equipamentos industriais. Mais informações sobre controladores PLC podem ser encontradas em (MATIC, 2001).

Como pode-se observar, as possibilidade são inúmeras se consideradas todas as combinações possíveis entre os elementos do sistema. O conjunto de implementações possíveis no *hardware* e no *software* podem, enfim, conduzir a aplicações completamente distintas, cabendo, portanto, ao pesquisador, apenas decidir que rumo deverá seguir.

Referências Bibliográficas

AID. *Links para projetos baseados no PIC*. http://www.interaccess.org/aid/links.html: [s.n.], 2003.

ANGNES, D. L. *Introdução ao Microontrolador Basic Step*. 2003.

CHELLA, M. T. *Ambiente de Robótica Educacional com Logo*. http://www.nied.unicamp.br/~siros/doc/artigo_sbc2002_wie_final.PDF: [s.n.], 2002.

CORREIA, L. H. A.; SILVA, R. M. de A. *Redes de Computadores*. 1. ed. Lavras: UFLA/FAEPE, 2002.

CUGNASCA, C. E. *Comunicação Serial - Revisão*. 2002.

FGEDITORES. *Robôs*. 1. ed. Madrid: FG Editores, 2002.

LACERDA, W. S. *Arquitetura de Computadores*. 1. ed. Lavras: UFLA/FAEPE, 2002.

MADSEN, J. D. *PIC-Programmer 2 for PIC16C84 etc*. http://www.jdm.homepage.dk/newpic.htm: [s.n.], 2000.

MATIC, N. *PIC microcontrollers*. 1. ed. Belgrade: mikroElektronika, 2000.

MATIC, N. *Introduction to PLC controllers*. 1. ed. Belgrade: mikroElektronika, 2001.

PALONEN, H. *Penguin PIC'n*. http://www.yty.net/pic/index.html: [s.n.], 2003.

SALVADOR, M. B. L. *Introdução ao Superlogo*. http://www.dma.ufs.br/slogo.ppt: [s.n.], 2001.

SWEET, M. R. *Serial Programming Guide for POSIX Operating Systems*. http://www.easysw.com/ mike/serial/: [s.n.], 2003.

TEXASINSTRUMENT. *MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS*. 2002. SLLS047I - FEBRUARY 1989 - REVISED OCTOBER 2002.

Apêndice A

Diagrama do circuito conversor

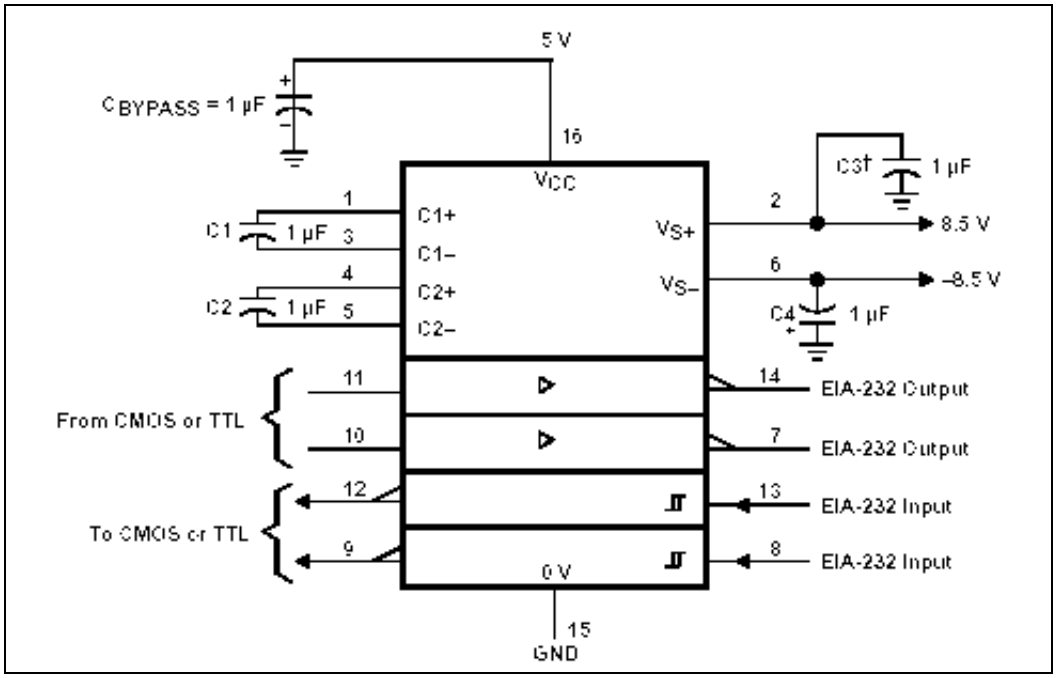

Figura A.1: Circuito conversor RS232 x TTL

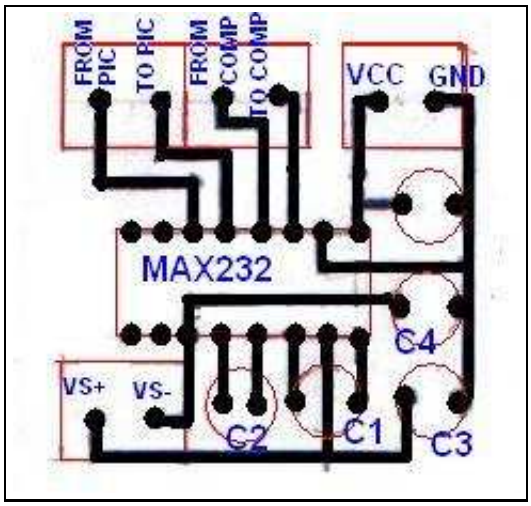

Figura A.2: Circuito impresso - lado da solda

Apêndice B

Listagem do programa do robô

```
************************************************
; Pós-graduação em Administração de Redes Linux/UFLA
; Projeto de integração Monty x Linux
; Programa de controle do robô Monty
;
; Autor:  Aléxis Rodrigues de Almeida
; Data primeira versão:  18/07/2004
;
; Este programa é responsável por estabelecer uma comunicação serial
; com o programa controlador no Linux através da porta serial do
; computador.  O programa também gerencia os sensores suportados
; pelo robô e repassa os valores capturados para o computador.  Uma
; terceira função do programa é executar os comandos enviados pelo
; programa do computador.
;--------------------------------------------

        list    p=16f84
        include "p16f84.inc"

CLKIN           EQU         .4000000
BAUDIOS         EQU         .9600
T_MODO          EQU         1
R_MODO          EQU         1
T_Nbit          EQU         8
R_Nbit          EQU         8
Sbit            EQU     2
Rs232_var   EQU             0xC


        cblock 0x31

        PontTab
        IdEvento
        Contador
        Eventos
        Buffer

        endc

TamMaxBuffer    equ             (0x3F-Buffer+1)
```

```
        ORG 0x00
        goto    INICIO
        ORG 0x04
        goto    INTER
        ORG 0x05

Tabela          movwf       PCL

sCRLF           EQU         \$
        dt  0x0D,0x0A,0x00

sBE     EQU     \$
        dt  "BEsq",0x00
sBD     EQU     \$
        dt  "BDir",0x00
sCE     EQU     \$
        dt  "CEsq",0x00
sCD     EQU     \$
        dt  "CDir",0x00
sUS     EQU     \$
        dt  "USom",0x00
sS      EQU     \$
        dt  "Som",0x00

sOk     EQU     \$
        dt  "<OK>",0x0D,0x0A,0x00
sErr            EQU     \$
        dt  "<ERR>",0x0D,0x0A,0x00
sVer            EQU     \$
        dt  "<MONTY - Linux v1.0.2>",0x0D,0x0A,0x00

        INCLUDE "RS232LOW.INC"


INICIO

        clrf    PORTA
        bsf STATUS,RP0

        movlw   b'10001111'
        movwf   OPTION_REG

;       movlw   b'00000000'
;       movwf   INTCON

        movlw   b'00000000'
        movwf   TRISA
        movlw   b'11101111'
        movwf   TRISB

        movlw   b'00000111'
        movwf   TMR0

        bcf STATUS,RP0
        call    DELAY1S
        call    DELAY1S

        call    EnviaVersao
        Call    EnviaOk
```

```
AGUARDA_COMANDO

        clrwdt                  ; atualiza cao de guarda
        movlw   '>'                     ; envia sinal de pronto
        movwf   Txdreg
        call    TxD             ;

        call    RxD             ; aguarda primeiro caractere

        movlw   '<'                     ; é o caractere de inicio de comando?
        subwf   Rxdreg,W
        btfsc   STATUS,Z
        goto    RECEBE_COMANDO      ; sim, então recebe o comando


        goto    AGUARDA_COMANDO

RECEBE_COMANDO
        clrf    Contador
        movlw   Buffer
        movwf   FSR
RECEBE_CAR
        call    RxD             ; aguarda caractere

        movlw   '>'                     ; é o caractere de fim de comando?
        subwf   Rxdreg,W
        btfsc   STATUS,Z
        goto    EXECUTA_COMANDO     ; sim, então recebe o comando

        movf    Rxdreg,w
        movwf   Txdreg
        call    TxD
        call    EnviaEnter

        movf    Contador,W      ; Estourou o buffer?
        sublw   TamMaxBuffer
        btfsc   STATUS,Z
        goto    ERRO                    ; sim, chama erro e retorna

        movf    Rxdreg,W        ; não, armazena o car e espera próximo
        movwf   INDF                    ;

        incf    Contador,F
        incf    FSR,F
        goto    RECEBE_CAR

EXECUTA_COMANDO

        movlw   Buffer
        movwf   FSR

        movf    INDF,W
        sublw   'A'
        btfsc   STATUS,Z
        goto    AVANCA

        movf    INDF,W
        sublw   'E'
        btfsc   STATUS,Z
        goto    ESQUERDA
```

```
        movf    INDF,W
        sublw   'D'
        btfsc   STATUS,Z
        goto    DIREITA

        movf    INDF,W
        sublw   'P'
        btfsc   STATUS,Z
        goto    PARA

        movf    INDF,W
        sublw   'R'
        btfsc   STATUS,Z
        goto    RETROCEDE

        movf    INDF,W
        sublw   'S'
        btfsc   STATUS,Z
        goto    SENSORES

        movf    INDF,W
        sublw   's'
        btfsc   STATUS,Z
        goto    sensores

        movf    INDF,W
        sublw   'V'
        btfsc   STATUS,Z
        goto    ENVIA_VER

        goto    ERRO

AVANCA
        movlw   b'00000110'         ; avança
        movwf   PORTA

        call    EnviaOk
        goto    AGUARDA_COMANDO

ESQUERDA
        movlw   b'00000101'         ; avança para a esquerda
        movwf   PORTA

        call    EnviaOk
        goto    AGUARDA_COMANDO

DIREITA
        movlw   b'00001010'         ; avança para a direita
        movwf   PORTA

        call    EnviaOk
        goto    AGUARDA_COMANDO

PARA
        movlw   b'00000000'         ; para
        movwf   PORTA

        call    EnviaOk
        goto    AGUARDA_COMANDO
```

```
RETROCEDE
        movlw   b'00001001'         ; retrocesso
        movwf   PORTA

        call    EnviaOk
        goto    AGUARDA_COMANDO

SENSORES
;       incf    FSR,F
;       movf    INDF,W
;       sublw   '0'
;       btfsc   STATUS,Z
;       goto    DESLIGA
;       bsf     PORTB,7
;       call    EnviaOk

        clrf    Eventos
;       call    EnviaEnter
        movlw '<'
        movwf   Txdreg
        call    TxD

        btfsc   PORTB,1
        call    EvContrasteEsq
        btfsc   PORTB,0
        call    EvContrasteDir
        btfsc   PORTB,2
        call    EvBumperEsq
        btfsc   PORTB,3
        call    EvBumperDir
        btfss   PORTB,6
        call    EvUltraSom
        btfsc   PORTB,7
        call    EvSom

        movlw '>'
        movwf   Txdreg
        call    TxD
        call    EnviaEnter

        goto    AGUARDA_COMANDO

;-----
sensores

;       call    EnviaEnter

        movlw '<'
        movwf   Txdreg
        call    TxD

        movlw '0'
        btfsc   PORTB,1         ;EvContrasteEsq
        movlw '1'
        movwf   Txdreg
        call    TxD

        movlw '0'
        btfsc   PORTB,0         ;EvContrasteDir
```

```
        movlw ’1’
        movwf   Txdreg
        call    TxD

        movlw ’0’
        btfsc   PORTB,2         ;EvBumperEsq
        movlw ’1’
        movwf   Txdreg
        call    TxD

        movlw ’0’
        btfsc   PORTB,3 ;EvBumperDir
        movlw ’1’
        movwf   Txdreg
        call    TxD

        movlw ’0’
        btfss   PORTB,6         ;EvUltraSom
        movlw ’1’
        movwf   Txdreg
        call    TxD

        movlw ’0’
        btfsc   PORTB,7         ;EvSom
        movlw ’1’
        movwf   Txdreg
        call    TxD

        movlw ’>’
        movwf   Txdreg
        call    TxD

        call    EnviaEnter

        goto    AGUARDA_COMANDO

;-----

DESLIGA
        bcf PORTB,7

        call    EnviaOk
        goto    AGUARDA_COMANDO

ENVIA_VER
        call    EnviaVersao
        goto    AGUARDA_COMANDO

ERRO
        call    EnviaErro
        goto    AGUARDA_COMANDO

;----------------------------------------
EnviaEnter
        movlw   sCRLF
        movwf   PontTab
        goto    EnviaString

;----------Subrotinas de Tratamento de Eventos --------
```

```
EvContrasteEsq
        call    EnviaSeparador
        incf    Eventos,F
        movlw   sCE
        movwf   PontTab
        goto    EnviaString

EvContrasteDir
        call    EnviaSeparador
        incf    Eventos,F
        movlw   sCD
        movwf   PontTab
        goto    EnviaString

EvBumperEsq
        call    EnviaSeparador
        incf    Eventos,F
        movlw   sBE
        movwf   PontTab
        goto    EnviaString

EvBumperDir
        call    EnviaSeparador
        incf    Eventos,F
        movlw   sBD
        movwf   PontTab
        goto    EnviaString

EvUltraSom
        call    EnviaSeparador
        incf    Eventos,F
        movlw   sUS
        movwf   PontTab
        goto    EnviaString

EvSom
        call    EnviaSeparador
        incf    Eventos,F
        movlw   sS
        movwf   PontTab
        goto    EnviaString

EnviaSeparador
        movf    Eventos,W
        andwf   Eventos,F
        btfsc   STATUS,Z
        return
        movlw ','
        movwf   Txdreg
        call    TxD
        return

;----Subrotina EnviaOk -----------------------------
EnviaOk

        movlw   sOk
        movwf   PontTab
        goto    EnviaString

;----Subrotina EnviaErro ----------------------------
```

```
EnviaErro
        movlw   sErr
        movwf   PontTab
        goto    EnviaString

;----Subrotina EnviaVersao -----------------------------
EnviaVersao
        movlw   sVer
        movwf   PontTab
        goto    EnviaString

;----Subrotina EnviaString -----------------------------

EnviaString
        call    Tabela

        movwf   Txdreg
        andwf   Txdreg,F
        btfsc   STATUS,Z
        return

        call    TxD
        incf    PontTab,F
        movf    PontTab,W
        goto    EnviaString
        return

;-----------------------------------------------
DELAY1S:
        bcf STATUS,RP0
        movlw   .100
        movwf   Contador

DELAY10MS:  movlw           0xD8
        movwf   TMR0
        clrwdt                  ; atualiza cao de guarda

DEL10:          btfss       INTCON,2
        goto    DEL10
        bcf INTCON,2
        decfsz  Contador,F
        goto    DELAY10MS
        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INTER
        movlw   '.'                     ;
        movwf   Txdreg
        call    TxD             ;
        bcf INTCON,INTF
        retfie


end
```

Apêndice C

Listagem do programa do computador

```
*******************************************
/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

main.c:  módulo principal
data:  01/09/2004
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "com_serial.h"
#include "int_logo.h"
#include "interface.h"
#include "util.h"

#include "main.h"

int main(int argc, char * argv[])
{
    char Comando[TAM_MAX_CMD+1];
    char Param[TAM_MAX_CMD+1];
    char Resp[TAM_MAX_CMD+1];
    int Result;
    int CodCmd;
    if (!ProcessaParam(argc,argv))
    {
        Help(argv[0]);
        exit(1);
    }

    do
    {
        Comando[0]=0;
        Result=PegaComando(Comando);
```

```
        if (Comando[0]=='<')
        {
            ExecutaPrimitiva(Comando,Resp);
            Mensagem(MSG_AVISO,Resp);
        }
        else
        {
            CodCmd=IdentificaComando(Comando,Param);

            switch(CodCmd)
            {
                case C_LING:
                    DefineLinguagem(Param);
                break;

                case C_PROT:
                    DefineProtocolo(Param);
                break;

                case C_EXEC:
                    if (!ExecutaProg(Param))
                        Mensagem(MSG_ERRO,"ERRO!");
                break;
                case C_FIM:
                    Finaliza();
                    return(0);
                break;
                default:
                    if (!ExecutaComando(Comando))
                        Mensagem(MSG_ERRO,"ERRO!");
                break;
            }
        }
    } while (1);


    return(0);
}

int IdentificaComando(char *Comando, char * Param)
{
    char *TabelaCmd[]={C_S_LING, C_S_PROT, C_S_EXEC, C_S_FIM};

    return(PegaIndTabela(Comando,TabelaCmd,QTD_CMDS,Param));
}

int ExecutaComando(char *Comando)
{
    switch(gLinguagem)
    {
        case LING_LOGO:
            return(LOGO_ExecutaComando(Comando,0));
        break;
    }
    return(0);
}

int ExecutaProg(char *Prog)
{
    switch(gLinguagem)
```

```
    {
        case LING_LOGO:
            return(LOGO_ExecutaProg(Prog));
        break;
    }
    return(0);
}

int ExecutaPrimitiva(char *Primitiva,char *Resposta)
{
    int IdPrimitiva,Retorno;
    char Resp[TAM_MAX_RESP+1];

    IdPrimitiva=IdentificaPrimitiva(Primitiva);

    switch(gProtocolo)
    {
        case PROT_SERIAL:
            Retorno=SERIAL_EnviaPrimitiva(Primitiva,Resp);
            if (Resposta) strcpy(Resposta,Resp);
            return(Retorno);
        break;

    }
    return(0);
}

void Finaliza()
{
    switch(gLinguagem)
    {
        case LING_LOGO:
            LOGO_Finaliza();
        break;
    }
}

int DefineLinguagem(char *Ling)
{
    int IdLing;

    IdLing=IdentificaLinguagem(Ling);

    switch(IdLing)
    {
        case LING_LOGO:
            if (LOGO_Inicializa(&ExecutaPrimitiva))
            {
                Mensagem(MSG_AVISO,"O interpretador atual é o %s.",Ling);
                gLinguagem=IdLing;
                return(1);

            }
            else
                Mensagem(MSG_ERRO,"Inicializando interpretador.");
        break;
    }
    return(0);
}
```

```
int DefineProtocolo(char *Prot)
{
    char Device[100];
    char Resposta[TAM_MAX_RESP+1];
    int IdProt;

    IdProt=IdentificaProtocolo(Prot,Device);

    switch(IdProt)
    {
        case PROT_SERIAL:
            if (SERIAL_Inicializa(Device,Resposta))
            {
                Mensagem(MSG_AVISO,Resposta);
                Mensagem(MSG_AVISO,
                    "O protocolo atual é %s, no device %s.",
                    PROT_S_SERIAL,Device);
                gProtocolo=IdProt;
                return(1);
            }
            else
                Mensagem(MSG_ERRO,"Inicializando porta serial");
        break;
    }
    return(0);
}

int IdentificaPrimitiva(char *Primitiva)
{
    char *TabelaDir[]={P_S_PARAR, P_S_AVANCAR, P_S_ESQUERDA,
            P_S_DIREITA, P_S_RETORNAR, P_S_SENSORES,
            P_S_SENS_RED, P_S_VERSAO};

    return(PegaIndTabela(Primitiva,TabelaDir,QTD_PRIM,NULL));
}


int IdentificaLinguagem(char *Linguagem)
{
    char *TabelaLing[]={LING_S_LOGO};

    return(PegaIndTabela(Linguagem,TabelaLing,QTD_LING,NULL));
}

int IdentificaProtocolo(char *Protocolo, char *Device)
{
    char *TabelaProt[]={PROT_S_SERIAL};

    return(PegaIndTabela(Protocolo,TabelaProt,QTD_PROT,Device));
}

int ProcessaParam(int argc, char *argv[])
{
    int i;

    gProtocolo=-1;
    gLinguagem=-1;

    for (i=1; i<argc; i++)
    {
```

```
        if (strcmp(argv[i],C_S_LING)==0)
        {
            if (!DefineLinguagem(argv[++i])) return(0);
        }
        else if (strcmp(argv[i],C_S_PROT)==0)
            {
            if (!DefineProtocolo(argv[++i])) return(0);
                }
        else if (strcmp(argv[i],--help")==0)
        {
            return(0);
        }
    }
    if (gLinguagem<0)
    {
        if (!DefineLinguagem(LING_S_LOGO)) return(0);
    }
    return(1);
}


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

main.h:  include do módulo principal
data:  01/09/2004
*/


//definições gerais
#define TAM_MAX_CMD 256
#define TAM_MAX_DIR 256
#define TAM_MAX_RESP 256

//definições das linguagens suportadas
#define LING_LOGO   0

#define QTD_LING 1

#define LING_S_LOGO "logo"

//definições dos protocolos suportados
#define PROT_SERIAL 0

#define QTD_PROT 1

#define PROT_S_SERIAL "serial"

//definições dos comando globais
#define C_LING  0
#define C_PROT  1
#define C_EXEC  2
#define C_FIM   3

#define QTD_CMDS 4

#define C_S_LING "ling"
#define C_S_PROT "prot"
#define C_S_EXEC "exec"
```

```
#define C_S_FIM "fim"

//definições das primitivas
#define P_PARAR                 0
#define P_AVANCAR   1
#define P_ESQUERDA          2
#define P_DIREITA   3
#define P_RETORNAR          4
#define P_SENSORES          5
#define P_SENS_RED          6
#define P_VERSAO    7

#define QTD_PRIM 8

#define P_S_PARAR   "<P>"
#define P_S_AVANCAR     "<A>"
#define P_S_ESQUERDA    "<E>"
#define P_S_DIREITA "<D>"
#define P_S_RETORNAR    "<R>"
#define P_S_SENSORES    "<S>"
#define P_S_SENS_RED    "<s>"
#define P_S_VERSAO  "<V>"

//#define P_STR(i) P_S(i)
//variáveis globais
int gLinguagem;
int gProtocolo;

//Declarações da funções
int ExecutaPrimitiva(char * Primitiva,char *Resp);
int ExecutaProg(char *Prog);
int DefineLinguagem(char *Ling);
int DefineProtocolo(char *Prot);

int EnfileiraComando(char *Comando);
int ExecutaComando(char *Comando);
int IdentificaComando(char *Comando, char *Param);
int ProcessaParam(int argc, char *argv[]);
void Finaliza();


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

util.c:  módulo com funções utilitárias
data:  01/09/2004
*/

#include <string.h>
#include "util.h"


int PegaIndTabela(char *Elemento,char *Tab[],int QtdElem,char *Param)
{
    char Elem[256];
    char *pElem, *pParam;
    int i;

    strcpy(Elem,Elemento);
```

```
    pElem=strtok(Elem,":");

    if (!pElem) return(-1);

    pParam=strtok(0L,);
    if (pParam && Param) strcpy(Param,pParam);

    for (i=0; i<QtdElem; i++)
    {
        if (strcmp(pElem,Tab[i])==0) return(i);
    }

    return(-1);
}


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

util.h:  include do módulo utilitário
data:  01/09/2004
*/

int AbreArquivos();
void Erro();
int PegaIndTabela(char *Elemento,char *Tab[],int QtdElem,char *Param);


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

int_logo.c:  módulo interpretador da linguagem LOGO
data:  01/09/2004
*/

#include <string.h>
#include <stdio.h>
#include "int_logo.h"
#include "main.h"
#include "interface.h"

int (*pExecPrim)(char *Dir,char *p);

int IdentificaCmdLogo(char *Comando);
char *ProxToken(char *Ini,char *Token,char Car);


int LOGO_Inicializa(int (*ExecPrimitiva)(char *Dir,char *p))
{
    pExecPrim=ExecPrimitiva;
    return(1);
}

int LOGO_Finaliza()
{
    return(1);
}
```

```
int LOGO_ExecutaComando(char *Comando, int IndLoop)
{
    char Cmd[TAM_MAX_CMD];
    char Param[TAM_MAX_CMD];
    char *pCont;
    int Id;

    if (strlen(Comando)==0) return(0);
    if (strncmp(Comando,"//",2)==0) return(0);

    pCont=ProxToken(Comando,Cmd,' ');

    while (Cmd[0])
    {
        Id=IdentificaCmdLogo(Cmd);

        switch(Id)
            {
            case LOGO_INV:
                ; //return(0);
            case LOGO_PF:
                pCont=ProxToken(pCont,Param,' ');
                ExecPF(Param);
            break;
            case LOGO_PT:
                pCont=ProxToken(pCont,Param,' ');
                ExecPT(Param);
            break;
            case LOGO_PE:
                pCont=ProxToken(pCont,Param,' ');
                ExecPE(Param);
            break;
            case LOGO_PD:
                pCont=ProxToken(pCont,Param,' ');
                ExecPD(Param);
            break;
            case LOGO_DIGA:
                pCont=ProxToken(pCont,Param,' ');
                ExecDIGA(Param);
            break;
            case LOGO_REPITA:
            {
                char *FimBl,*IniBl,*Pont,*Bloco;
                int i,Vezes;

                pCont=ProxToken(pCont,Param,' ');
                sscanf(Param,"%d",&Vezes);

                IniBl=strchr(pCont,'[');

                while(*IniBl==' ' || *IniBl=='[') IniBl++;

                FimBl=strrchr(IniBl,']');
                pCont=FimBl+1;
                Bloco=(char *)malloc(strlen(IniBl)+1);
                strncpy(Bloco,IniBl,FimBl-IniBl);
                Bloco[FimBl-IniBl]=0;

                for (i=0; i<Vezes; i++)
                    LOGO_ExecutaComando(Bloco,i);
```

```
                free(Bloco);
            }
            break;

        }
        if (pCont)
            pCont=ProxToken(pCont,Cmd,' ');
        else
            Cmd[0]=0;
    }
    return(1);
}

int LOGO_ExecutaProg(char *Prog)
{
    FILE *hArq;
    char Linha[TAM_MAX_CMD+1];

    hArq=fopen(Prog,"rt");

    if (!hArq) return(0);

    Linha[0];
    fgets(Linha,TAM_MAX_CMD,hArq);

    while (!feof(hArq))
    {
        char *p;

        if (strlen(Linha)>0)
        {
            p=Linha+strlen(Linha);
            while (p>Linha && (*p=='\n' || *p=='\r' || *p==0)) p-;
            *(++p)=0;
            LOGO_ExecutaComando(Linha,0);
        }
        Linha[0];
        fgets(Linha,TAM_MAX_CMD,hArq);
    }
    fclose(hArq);
    return(1);
}

int ExecPF(char * Param)
{
    int Qtd;
    char Resp[TAM_MAX_RESP];

    if (Param)
    {
        sscanf(Param,"%d",&Qtd);
        Resp[0]=0;
        (*pExecPrim)(P_S_AVANCAR,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        sleep(DELAY_DIST(Qtd));
        (*pExecPrim)(P_S_PARAR,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        return(1);
    }
    return(0);
```

```
}

int ExecPE(char * Param)
{
    int Qtd;
    char Resp[TAM_MAX_RESP];

    if (Param)
    {
        sscanf(Param,"%d",&Qtd);
        Resp[0]=0;
        (*pExecPrim)(P_S_ESQUERDA,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        sleep(DELAY_ANG(Qtd));
        (*pExecPrim)(P_S_PARAR,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        return(1);
    }
    return(0);
}

int ExecPD(char * Param)
{
    int Qtd;
    char Resp[TAM_MAX_RESP];

    if (Param)
    {
        sscanf(Param,"%d",&Qtd);
        Resp[0]=0;
        (*pExecPrim)(P_S_DIREITA,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        sleep(DELAY_ANG(Qtd));
        (*pExecPrim)(P_S_PARAR,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        return(1);
    }
    return(0);
}

int ExecPT(char * Param)
{
    int Qtd;
    char Resp[TAM_MAX_RESP];

    if (Param)
    {
        sscanf(Param,"%d",&Qtd);
        Resp[0]=0;
        (*pExecPrim)(P_S_RETORNAR,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        sleep(DELAY_DIST(Qtd));
        (*pExecPrim)(P_S_PARAR,Resp);
        if (strncmp(Resp,"<OK>",4)!=0) return(0);
        return(1);
    }
    return(0);
}

int ExecDIGA(char * Param)
```

```
{
    int Qtd;
    char Resp[TAM_MAX_RESP];

    if (Param)
    {
        Resp[0]=0;
        switch(Param[0])
        {
            case 'S':
                (*pExecPrim)(P_S_SENSORES,Resp);
            break;
            case 's':
                (*pExecPrim)(P_S_SENS_RED,Resp);
            break;
            case 'V':
                (*pExecPrim)(P_S_VERSAO,Resp);
            break;
        }

        if (Resp[0]=='<')
        {
            Mensagem(MSG_AVISO,Resp);
            return(1);
        }
        else
            return(0);
    }
    return(0);
}


int IdentificaCmdLogo(char *Comando)
{
    char *Tabela[]={LOGO_S_PF, LOGO_S_PT, LOGO_S_PE,
            LOGO_S_PD, LOGO_S_REPITA, LOGO_S_DIGA};

    return(PegaIndTabela(Comando,Tabela,QTD_CMD_LOGO,NULL));
}

char *ProxToken(char *Ini,char *Token,char Car)
{
    char *p;

    Token[0]=0;
    if (!Ini) return(0L);
    while(*Ini==Car) Ini++;
    p=strchr(Ini,Car);
    if (p)
    {
        int Tam;
        Tam=p-Ini;
        strncpy(Token,Ini,Tam);
        Token[Tam]=0;
        return(p+1);
    }
    else
    {
        strcpy(Token,Ini);
        return(0L);
```

```
    }
}


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

int_logo.h:  include do módulo interpretador LOG
data:  01/09/2004
*/

// Comandos LOGO

#define LOGO_INV -1
#define LOGO_PF 0
#define LOGO_PT 1
#define LOGO_PE 2
#define LOGO_PD 3
#define LOGO_REPITA 4
#define LOGO_DIGA 5

#define QTD_CMD_LOGO 6

#define LOGO_S_PF "PF"
#define LOGO_S_PT "PT"
#define LOGO_S_PE "PE"
#define LOGO_S_PD "PD"
#define LOGO_S_REPITA "REPITA"
#define LOGO_S_DIGA "DIGA"

#define K_DIST 14
#define K_ANG 11

#define DELAY_DIST(Qtd) (Qtd * K_DIST)/50
#define DELAY_ANG(Qtd) (Qtd * K_ANG)/360

// Funções exportadas
int LOGO_Inicializa(int (*ExecPrimitiva)(char *Dir,char *p));
int LOGO_Finaliza();
int LOGO_ExecutaComando(char *Comando, int IndLoop);
int LOGO_ExecutaProg(char *Prog);


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

com_serial.c:  módulo driver do protocolo serial
data:  01/09/2004
*/

#include <stdio.h> /* Standard input/output definitions */
#include <string.h> /* String function definitions */
#include <unistd.h> /* UNIX standard function definitions */
#include <fcntl.h> /* File control definitions */
#include <errno.h> /* Error number definitions */
#include <termios.h>
#include <fcntl.h>
```

```
#include "com_serial.h"
#include "main.h"

int hPorta=-1;

int ConfiguraPorta(int hPorta);

int EnviaStr(char *Str);

int teste();

int SERIAL_Inicializa(char *Device,char *Resposta)
{
    int n;

    if (hPorta>=0) close(hPorta);

    hPorta=AbrePorta(Device);

    if (hPorta>=0)
    {
        if (!ConfiguraPorta(hPorta)) return(0);

//teste();
        n = EnviaStr(P_S_VERSAO);
        if (n==strlen(P_S_VERSAO))
        {
            n = RecebeStr(Resposta);
            if (strncmp(Resposta,"<MONTY",6)==0) return(1);
        }
    }
    return(0);
}

int SERIAL_Finaliza()
{
    if (hPorta>=0) close(hPorta);
    return(1);
}

int SERIAL_EnviaPrimitiva(char *Primitiva, char *Resposta)
{
    int n;

    n = EnviaStr(Primitiva);

    if (n < 0) return(0);

    n = RecebeStr(Resposta);

    if (n>0) return(1);

    return(0);
}


/*********************************************
AbrePorta(char *Dev)
- AbrePorta informada em "Dev".
- Retorna handler do arquivo ou -1 se houve erro.
```

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

int AbrePorta(char *Device)
{
    int hArq;

        hArq = open(Device, O_RDWR | O_NOCTTY | O_NDELAY);
    if (hArq != -1) fcntl(hArq, F_SETFL, 0);

    return (hArq);
}

int ConfiguraPorta(int hPorta)
{
    struct termios op;

        tcgetattr(hPorta, &op);

    op.c_ispeed &= ~CBAUD;
    op.c_ospeed &= ~CBAUD;
    op.c_ispeed |= B9600;
    op.c_ospeed |= B9600;

/* raw, timeout=1 segundo */
        op.c_cflag |= (CLOCAL | CREAD);
        op.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
        op.c_oflag &= ~OPOST;
        op.c_iflag |= IGNPAR;
    op.c_cc[VMIN] = 0;
    op.c_cc[VTIME] = 10; //timeout 1 segundo

    tcsetattr(hPorta, TCSANOW, &op);

    return(1);
}

int EnviaStr(char *Str)
{
    int i,k,m;
    struct timespec Delay,Aux;

    for(i=0;i<strlen(Str);i++)
    {
        if (write(hPorta, &Str[i], 1) < 1) return(i);
        //Delay.tv_sec=1;
        //Delay.tv_nsec=0;
        Delay.tv_sec=0;
        Delay.tv_nsec=1;
        nanosleep(&Delay,&Aux);
    }
    return(i);
}

int RecebeStr(char *Str)
{
    int Tent,Qtd;
    char *PontBuf,Buffer[TAM_MAX_RESP+1];
    char Car;
    int i,Cap;
```

```
    PontBuf = Str;
    Cap=0;
    Buffer[0]=0;
    while ((Qtd=read(hPorta, &Buffer, 50)) > 0) //3,10 ok
    {
        for (i=0;i<Qtd; i++)
        {
            Car=Buffer[i];
            if (Car ==’<’) Cap=1;
            if (Cap) *(PontBuf++)=Car;
            if (Car ==’>’) Cap=0;
        }
        Buffer[0]=0;
    }
    *PontBuf = ’\0’;
    return(PontBuf-Str);
}


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

com_serial.h:  include do módulo do protocolo serial
data:  01/09/2004
*/

int SERIAL_Inicializa(char *Device,char *Resposta);
int SERIAL_Finaliza();
int SERIAL_EnviaPrimitiva(char *Primitiva, char *Resposta);


/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

interface.c:  módulo de interface com o usuário
data:  01/09/2004
*/

//#include <varargs.h>
#include <stdio.h>
#include <stdarg.h>

#include "interface.h"
#include "main.h"

int PegaComando(char *Comando)
{
    int Tam;

    write(1,">",1);
    Tam=read(0,Comando,TAM_MAX_CMD);
    if (Tam>0) Comando[-Tam]=0;

//  Tam=fgets(Comando,TAM_MAX_CMD,STDIN);

    return(Tam);
}
```

```
void Help(char *NomeProg)
{
    printf("Execute:  %s [ling <Linguagem>] [prot <Protocolo>]\n",NomeProg);
    printf("\nLinguagem:  logo\n");
    printf("Protocolo:  serial:<device da porta serial>\n\n");
    printf("Exemplo:  %s ling logo prot serial:/dev/ttyS1\n",NomeProg);
}

void Mensagem(int Tipo,char *Mens,...)
{
    va_list PontArg;
    char MensExp[250];
    va_start( PontArg, Mens );
//      va_start( PontArg);
    vsprintf( MensExp, Mens, PontArg );
    va_end( PontArg);

    switch(Tipo)
    {
        case MSG_ERRO:
            printf("Erro:  %s\n",MensExp);
        break;
        case MSG_ADV:
            printf("Atenção:  %s\n",MensExp);
        break;
        case MSG_AVISO:
            printf("%s\n",MensExp);
        break;
    }

}

/*
Projeto:  robo - programa de controle do robô Monty
Autor:  Aléxis Rodrigues de Almeida

interface.h:  include do módulo de interface
data:  01/09/2004
*/
#define MSG_ERRO 0
#define MSG_ADV 1
#define MSG_AVISO 2

int PegaComando(char *Comando);
void Help(char *);
void Mensagem(int Tipo,char *Mens,...);
```