UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

DESENVOLVIMENTO DE MICRORROBÔ MÓVEL AUTÔNOMO

MATHEUS MAURÍCIO PEREIRA CASTRO

SÃO PAULO
2007

MATHEUS MAURÍCIO PEREIRA CASTRO

DESENVOLVIMENTO DE MICRORROBÔ MÓVEL AUTÔNOMO

Projeto de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro

Área de Concentração:
Engenharia Mecatrônica

São Paulo
2007

MATHEUS MAURÍCIO PEREIRA CASTRO

DESENVOLVIMENTO DE MICRORROBÔ MÓVEL AUTÔNOMO

Relatório Parcial do Projeto de Conclusão de Curso II apresentado à Escola Politécnica da Universidade de São Paulo

Área de Concentração:
Engenharia Mecatrônica

Orientador: Prof. Dr. Jun Okamoto Júnior

São Paulo
2007

FICHA CATALOGRÁFICA

Castro, Matheus Maurício Pereira
Desenvolvimento de microrrobô móvel autônomo − São Paulo, 2007.

Trabalho de Formatura − Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Robôs 2.Hardware (Controle) 3.Software (Controle)

AGRADECIMENTOS

Agradeço ao meu orientador, Prof. Dr. Jun Okamoto Jr., por suas valiosas orientações, inspirando grande motivação e transmitindo muita confiança durante toda a condução deste trabalho.

Agradeço ao Engenheiro José Carlos dos Santos pelo constante incentivo e apoio técnico, imprescindíveis para a conclusão deste projeto.

Agradeço ao Laboratório de Percepção Avançada do Departamento de Engenharia Mecatrônica da Escola Politécnica da USP por fornecer toda infra-estrutura técnica e apoio financeiro.

Agradeço à minha família, que indiretamente contribuiu, com seu apoio e incentivo, para o bom andamento deste projeto de conclusão de curso.

Os que se encantam com a prática sem a ciência são como os timoneiros que entram no navio sem timão nem bússola, nunca tendo certeza do seu destino.

(Leonardo da Vinci)

RESUMO

O presente projeto diz respeito ao desenvolvimento de um robô móvel autônomo de pequeno porte, sendo que o foco das atividades é dado ao desenvolvimento de hardware e software de controle. Este robô é composto de uma pequena plataforma móvel com duas rodas acionadas independentemente por dois motores de passo, sendo que o controle de posicionamento é feito em malha aberta. O robô possui sensores infravermelhos de distância, os quais são responsáveis pela percepção do ambiente no qual o robô está inserido. Há ainda um sistema de interface com o usuário, composto por botões e um display de cristal líquido. O software do robô é dividido hierarquicamente em dois níveis: baixo nível e alto nível. A camada de baixo nível diz respeito ao software de controle, o qual trata da execução das tarefas essenciais do robô, como, por exemplo, o acionamento dos motores e a leitura dos sensores. A camada de alto nível refere-se ao software de aplicação do robô, isto é, o software específico para que o robô cumpra uma determinada tarefa, que, neste caso, consiste em vagar pelo ambiente desviando de obstáculos.

ABSTRACT

This project concerns about the development of a small autonomous mobile robot, being the activities focused on the development of controlling hardware and software. The robot has a small two-wheeled mobile platform driven by two stepper motors, which are under open loop positional control. Infrared distance sensors are responsible for the environmental perception. In addition, pushbuttons and a liquid crystal display allow a user interface system. The robot software is divided in two levels: low level and high level. The low-level layer is related to the controlling software, which executes the robot essential tasks, like driving the stepper motors and acquiring the sensor signals. The high-level layer runs the algorithm responsible for the accomplishment of some task, which is, in this project, the mission of exploring the environment, avoiding obstacles.

SUMÁRIO

1 INTRODUÇÃO

A robótica móvel é atualmente uma das áreas de vanguarda nas instituições de ensino e de pesquisa em todo o mundo, sendo que há um interesse cada vez maior nas aplicações comerciais que vêm surgindo nesta área. A robótica móvel possibilita a atuação não-supervisionada de máquinas em tarefas complexas que requerem interação com o meio físico. Busca e salvamento de sobreviventes em situações de catástrofe, detecção de fogos em florestas, transporte de objetos, vigilância e limpeza de grandes áreas, exploração subaquática ou planetária, aplicações em agricultura (colheita autônoma, tratamento da terra, semeadura, etc.) constituem alguns exemplos de robôs móveis. A adoção comercial de robôs móveis autônomos tem ganhado recentemente um novo ímpeto através do aumento da sua taxa de crescimento e de desenvolvimento, graças à disponibilidade de simuladores e sistemas de hardwares a preços acessíveis (COSTA, 2003).

Dentro deste contexto promissor surgiu o interesse em se desenvolver o presente trabalho, o qual trata da concepção de um robô móvel autônomo de pequeno porte. Este robô é batizado neste projeto como robô *Jerry*, sendo que o mesmo foi inspirado em robôs de uma competição para resolver labirintos - a competição *Micromouse*. Esta competição é um evento que acontece em várias partes do mundo, onde microrrobôs competem com a finalidade de resolver um labirinto no menor tempo possível.

O projeto e a construção do robô *Jerry* é uma atividade que efetivamente envolve a combinação integrada de mecânica, eletrônica e computação. Desta forma, o desenvolvimento deste projeto de conclusão de curso provê uma oportunidade ímpar para se trabalhar todos os aspectos de um projeto genuinamente mecatrônico.

2 OBJETIVO

Este trabalho tem o objetivo de integrar e consolidar os conhecimentos obtidos durante o curso através do projeto completo de um sistema mecatrônico. Para isto, este projeto pretende o desenvolvimento de um robô móvel autônomo de pequeno porte (robô *Jerry)*.

O desenvolvimento do robô *Jerry* envolve duas etapas distintas. Na primeira etapa tem-se como meta o desenvolvimento do protótipo físico do robô, o qual será composto de uma pequena plataforma móvel e um hardware de controle. Já na segunda etapa trata-se da programação do robô *Jerry*, o que envolve o desenvolvimento de toda uma estrutura de software de controle do robô, bem como o desenvolvimento de um software de aplicação para que o robô cumpra uma determinada tarefa.

3 HARDWARE

O hardware do robô *Jerry* diz respeito ao protótipo físico do robô, sendo este composto de uma plataforma móvel (estrutura eletro-mecânica) e de um hardware de controle (circuito eletrônico). Este protótipo apresenta duas rodas acionadas independentemente por dois motores de passo, sendo que o controle de posicionamento será feito em malha aberta. Sensores infravermelhos de distância são responsáveis pela percepção do ambiente no qual o robô está inserido. O robô conta ainda com um sistema de interface com usuários, composto por botões, *leds* sinalizadores, e um display de cristal líquido.

3.1 PLATAFORMA MÓVEL

Decidiu-se por comprar de um fornecedor especializado uma plataforma móvel adequada ao projeto, resolvendo-se, desta forma, praticamente toda a questão da estrutura eletro-mecânica.

A plataforma móvel utilizada no projeto é a *AIRAT2* (Figura 3.1) do fornecedor *Active Robots Ltd.* Esta plataforma é composta por um chassi de alumínio, duas rodas de alumínio revestidas com borracha, dois apoios esféricos (*ball-casters*), e dois motores de passo acoplados às rodas.

As rodas são acionadas por dois motores Sanyo H546 (Figura 3.2). Estes são motores de passo híbridos com 1,8º de ângulo de passo. Eles operam com tensão nominal de 3,15 V e corrente elétrica de 1 A, fornecendo um torque estático de 0,147 N.m.

A locomoção do robô através desta plataforma se dá pelo acionamento diferencial das duas rodas, conferindo ao robô a possibilidade translação retilínea (para frente ou para trás), translação curvilínea e rotação em torno do próprio eixo. A estabilidade do movimento é garantida pelos apoios esféricos.

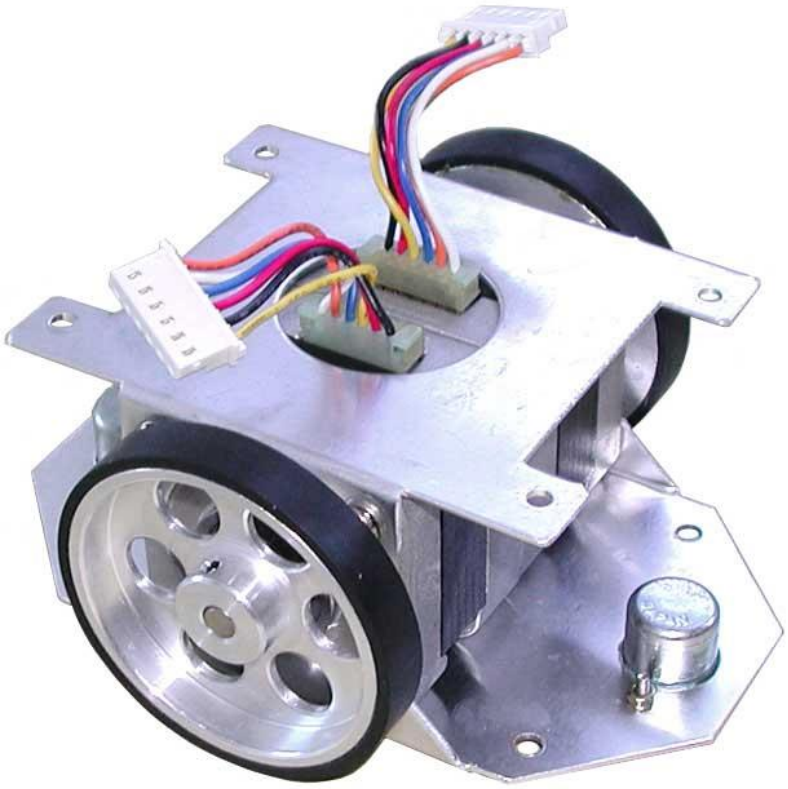


Figura 3.1 - Plataforma eletro-mecânica *AIRAT2*

Figura 3.2 - Motores de passo *Sanyo H546*

A Tabela 3.1 apresenta as especificações gerais da plataforma móvel AIRAT2.

Tabela 3.1 - Especificações da plataforma móvel AIRAT2

Item	Descrição
Tamanho da Plataforma	88 mm x 114 mm (*largura x comprimento*)
Frame do Chassi	144 mm x 67 mm (alumínio)
Roda	Roda de alumínio (ø51.3 mm com borracha) x 2, *ball caster* tamanho pequeno x 2
Motor	Motor de passo híbrido (Sanyo H546) x 2

3.2 HARDWARE DE CONTROLE

Este hardware foi projetado para desempenhar quatro funções básicas:

- Acionar e controlar os motores de passo;
- Fazer o sensoriamento do ambiente externo;
- Permitir uma interface com o usuário;
- Fornecer alimentação elétrica adequada para o circuito elétrico.

As três primeiras funcionalidades citadas são gerenciadas pelo módulo de processamento do hardware (microcontrolador).Na Figura 3.3, é mostrado o esquema geral do hardware de controle e monitoração do robô.

O projeto do circuito eletrônico e o layout da placa de circuito impresso foram elaborados com o auxílio do pacote de softwares *OrCAD*® da empresa *Cadence*®. Duas ferramentas deste pacote de software foram utilizadas: o *OrCAD Capture*, o qual foi usado para criar o projeto do circuito eletrônico em sua forma esquemática (referencie o Apêndice A para consultar o esquema elétrico completo do hardware); e o *OrCAD Layout*, o qual foi usado para projetar a disposição física dos componentes e circuitos na placa de circuito impresso (referencie o Apêndice B para consultar o layout da placa).

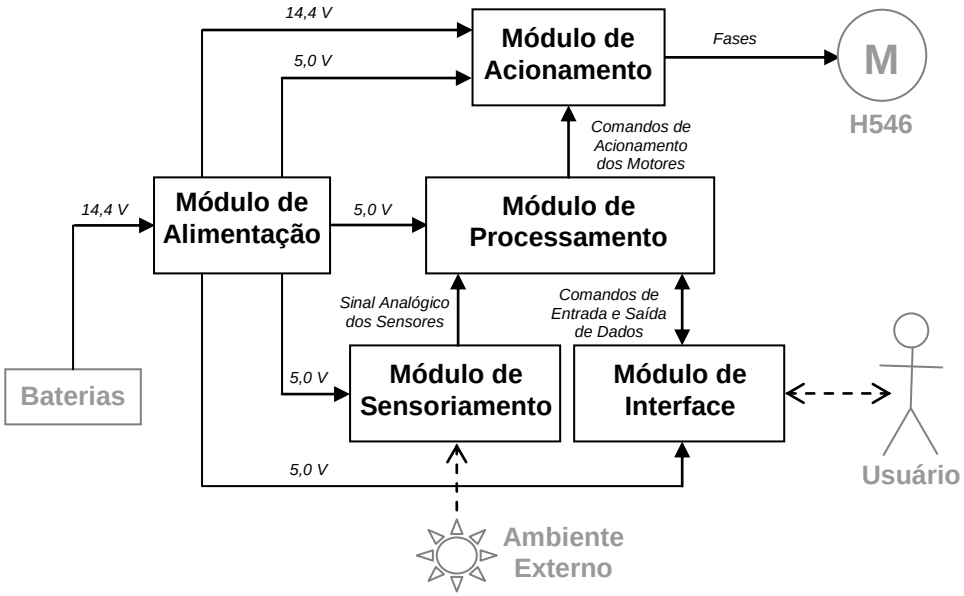


Figura 3.3 - Diagrama Esquemático do Hardware

3.2.1 Módulo de Processamento

O módulo de processamento é composto por um microcontrolador de 28 pinos, o *PIC16F873* do fabricante *Microchip Technology Inc*. Este microcontrolador possui 8K x 14 *words* de memória *FLASH*, 368 x 8 bytes de memória *RAM*, 256 x 8 bytes de memória *EEPROM*, 5 canais conversores analógico-digital, e será operado com uma freqüência de *clock* de 20MHz.

Este módulo é responsável por controlar o acionamento dos motores, fazer a aquisição dos sinais dos sensores, gerenciar a interface com o usuário, e ainda executar o software de aplicativo da tarefa a ser cumprida pelo robô. A Figura 3.4 mostra o diagrama simplificado deste sistema.

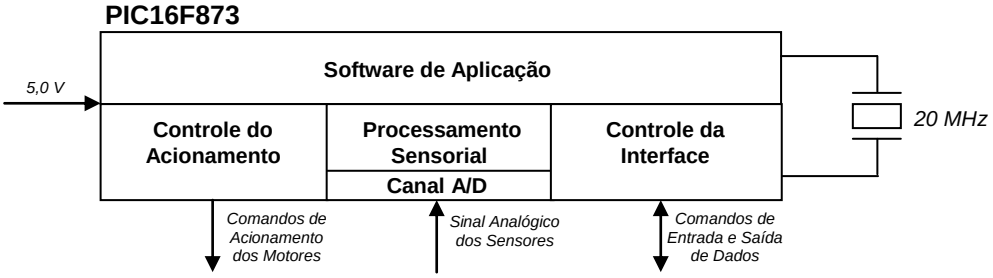


Figura 3.4 - Representação Simplificada do Módulo de Processamento

3.2.2 Módulo de Acionamento

O módulo de acionamento é responsável por acionar independentemente as duas rodas do robô através de cada um dos motores de passo (motor H546). Este sistema está representado simplificadamente pela Figura 3.5, e é composto por duas partes principais: o seqüenciador lógico (circuito integrado L297) e o *driver* de potência (circuito integrado SLA7024). É necessário ainda que o módulo de processamento (PIC16F873) envie os sinais de comando de passo (*clock*), de sentido de rotação, e de habilitação para o seqüenciador.

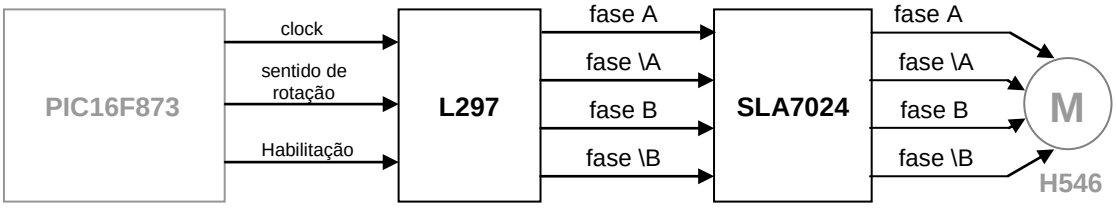


Figura 3.5 - Representação Simplificada do Módulo de Acionamento

O seqüenciador lógico é um circuito lógico que controla a excitação das bobinas do motor seqüencialmente, em resposta a um pulso de *clock* dado pelo microcontrolador. No caso do robô *Jerry*, o seqüenciador L297 faz a excitação do tipo *two-phase-on* para motores de quatro fases, isto é, duas das quatro fases do motor H546, estão energizadas ao mesmo tempo em cada estado de excitação. A Figura 3.6 mostra este tipo de seqüenciamento, onde os sinais *A, B, C e D* representam cada uma das quatro fases do motor.

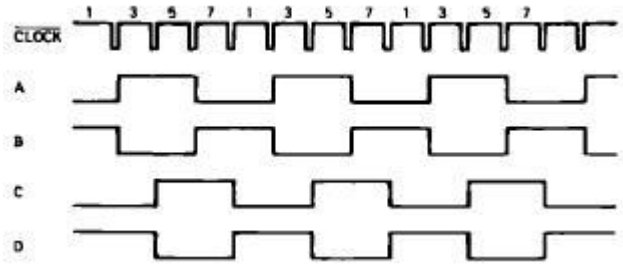


Figura 3.6 - Seqüenciamento de Fases

Os sinais de saída do seqüenciador lógico são transmitidos aos terminais de entrada do *driver* de potência, através do qual é comandado o chaveamento de corrente elétrica nas bobinas do motor. A função do *driver* é servir como um *buffer* de amplificação de corrente entre o seqüenciador lógico e o motor. O *driver* SLA7024 faz o acionamento por PWM (*Pulse Width Modulation*). As vantagens deste tipo de acionamento são: o uso de uma única fonte de tensão, a perda de potência é baixa, e a voltagem aplicada ao motor é automaticamente ajustada para que o acionamento seja feito a uma corrente elétrica pré-estabelecida. Resumindo, quando um pulso de *clock* é aplicado ao seqüenciador, os seus terminais de saída mudam para controlar o *driver*, o qual, por sua vez, faz o motor girar de um ângulo de passo.

3.2.3 Módulo de Sensoriamento

Sensores infravermelhos de distância (sensor Sharp GP2D120, Figura 3.7) compõem o módulo de sensoriamento, o qual é responsável pela percepção do ambiente no qual o robô está inserido. O sensor Sharp GP2D120 gera uma saída analógica inversamente proporcional à distância entre o sensor e um anteparo (Figura 3.8). Esta saída pode ser diretamente enviada ao microcontrolador PIC16F873, pois este dispositivo possui internamente um módulo conversor analógico-digital, o que simplifica bastante a montagem do circuito para este sistema (Figura 3.9).

Figura 3.7 - Sensor de Distância Sharp GP2D120

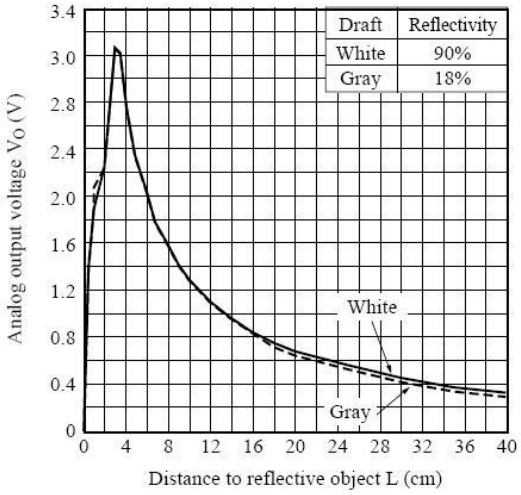


Figura 3.8 - Saída (Analógica) do Sensor Sharp GP2D120

O conversor analógico-digital do microcontrolador possui cinco canais de entrada, mas a placa do circuito possibilita doze posições de fixação para os sensores de distância (Figura 3.10), permitindo, assim, diferentes possibilidades de configurações de montagem para estes sensores. A Tabela 3.2 mostra as posições de fixação dos sensores que compartilham o mesmo canal de entrada do conversor. É importante observar que não se deve conectar mais de um sensor num mesmo canal de entrada, pois isto provocaria um curto-circuito entre os sensores ligados em comum (consulte o esquema elétrico no Apêndice A).

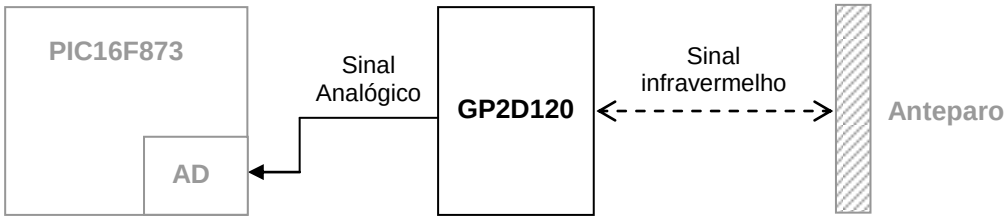


Figura 3.9 - Representação Simplificada do Módulo de Sensoriamento

A configuração do robô *Jerry* utilizará 4 sensores, fixados nas posições: J1, J2, J3 e J4. Esta configuração distribui os sensores uniformemente ao redor do robô, permitindo uma monitoração uniforme do ambiente.

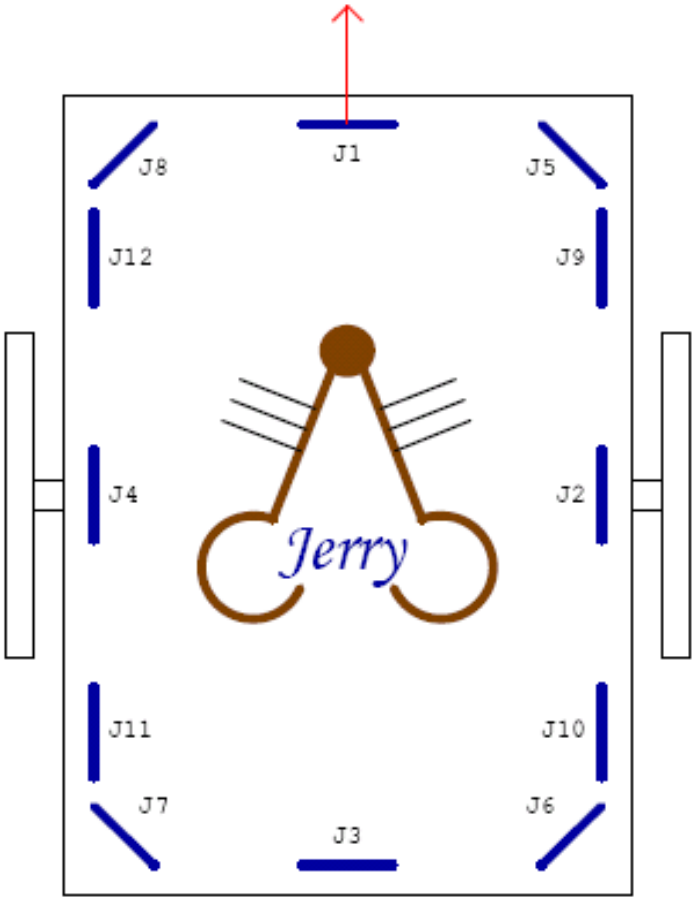


Figura 3.10 - Posições Possíveis para Fixação dos Sensores

Tabela 3.2 - Ligação Elétrica entre Canal A/D e Posição do Sensor

Canal A/D	Posição do Sensor
AN0	J1
AN1	J2, J5, J9
AN2	J3, J6, J10
AN3	J4, J7, J11
AN4	J8, J12

3.2.4 Módulo de Interface com o Usuário

O módulo de interface com o usuário conta com um display de cristal líquido de caracteres (*LCD H0802A*, Figura 3.11), quatro LEDs de sinalização, e cinco chaves do tipo *pushbutton*. Há ainda a possibilidade de um sinalizador sonoro (*buzzer*). A Figura 3.12 mostra o esquema simplificado deste módulo.

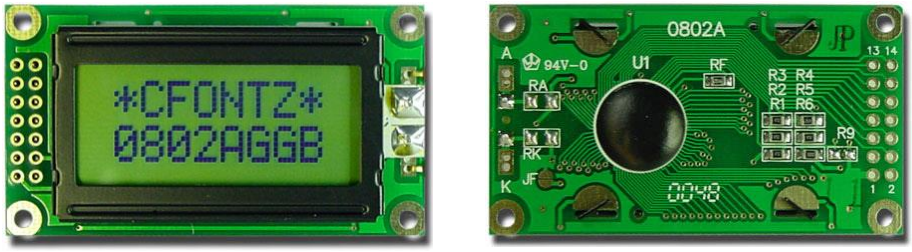


Figura 3.11 - Vistas Frontal e Traseira do *LCD*

O *LCD* é o principal dispositivo de saída para o usuário. Este componente possui oito colunas e duas linhas no mostrador de caracteres, sendo que cada caractere é formado por uma matriz de 5x10 pontos. Neste display pode-se imprimir, por exemplo, mensagens de status do robô. A luz de fundo do mostrador não será ligada neste projeto, a fim de se economizar energia elétrica da bateria.

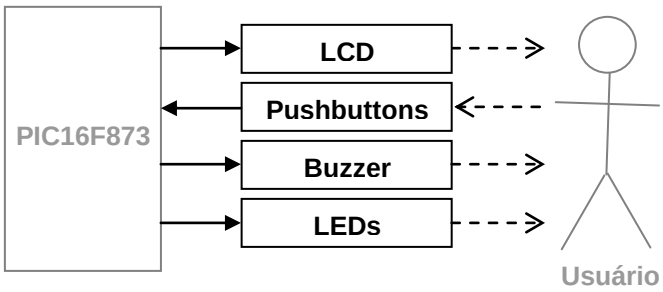


Figura 3.12 – Representação Simplificada do Módulo de Interface com o Usuário

Um dos cinco *pushbuttons* (botão *reset*) tem a função de reinicializar o sistema operacional do robô *Jerry*. Os outros quatro podem ter funcionalidades que variaram de acordo com a tarefa programada no robô.

3.2.5 Módulo de Alimentação

O módulo de alimentação é responsável por fornecer energia elétrica para o motor e para toda a parte eletrônica do circuito do hardware. Basicamente um regulador de tensão chaveado (circuito integrado LM2575) compõe este sistema (Figura 3.13). Um conjunto de baterias de voltagem nominal de 14,4V é utilizado como fonte de alimentação para o *driver* do motor (SLA7024) e como tensão de entrada para o regulador, o qual fornece tensão de saída de 5V para os demais componentes do hardware. Há ainda uma chave de duas posições, atuando diretamente no fornecimento de energia para o hardware, para ligar e desligar o robô.

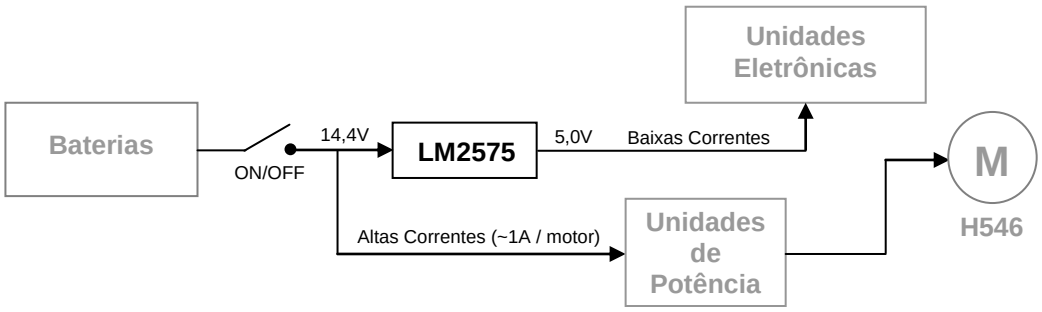


Figura 3.13 – Representação Simplificada do Módulo de Alimentação

3.3 MONTAGEM

As Figuras 3.14 e 3.15 mostram a placa de circuito impresso antes e depois da montagem dos componentes eletrônicos.

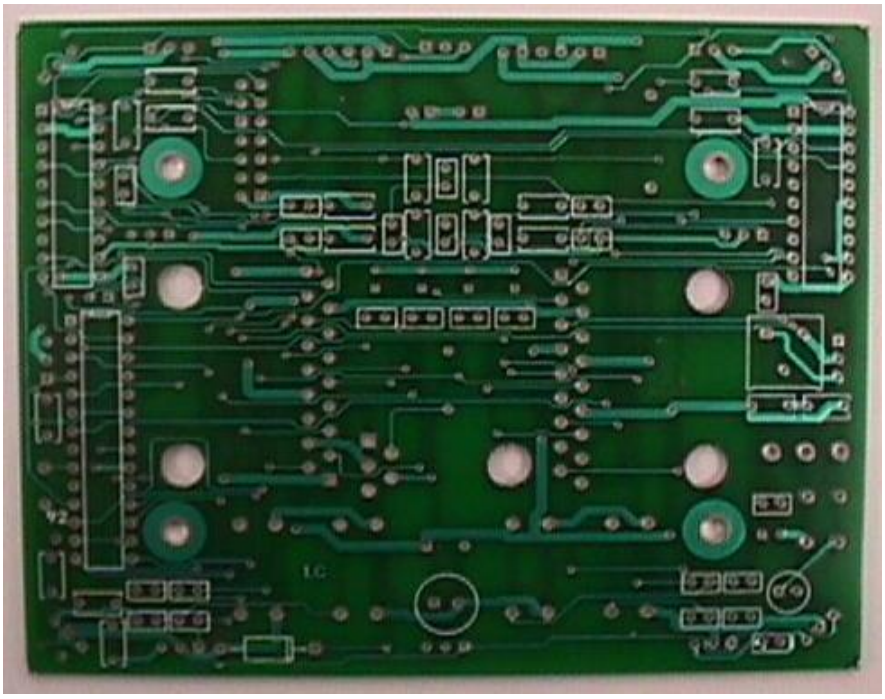


Figura 3.14 – Placa de Circuito Impresso do Hardware de Controle

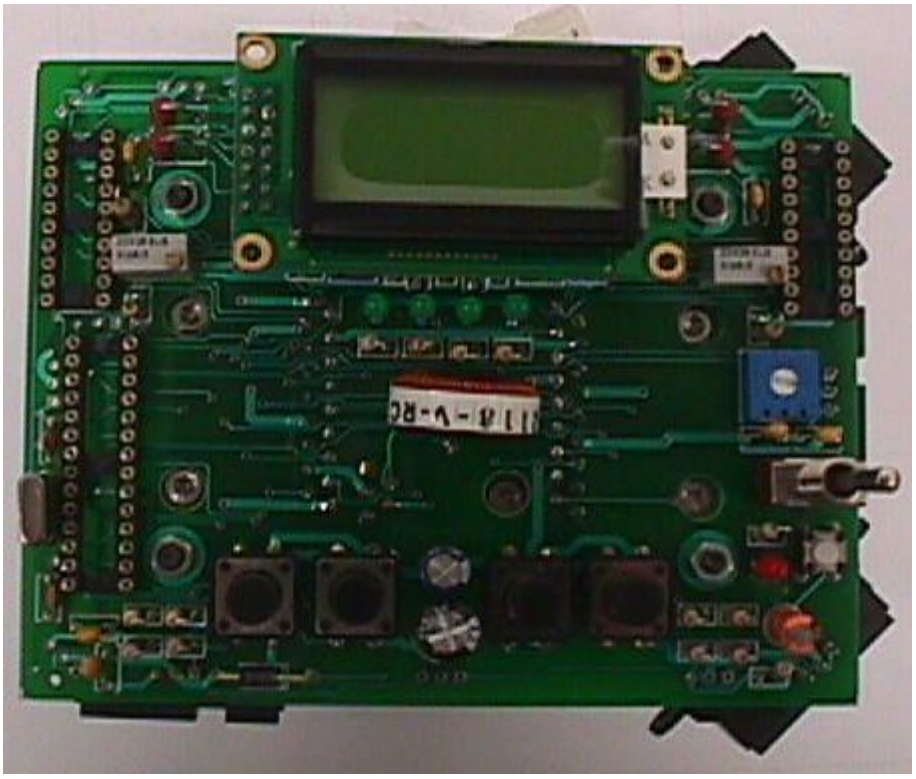


Figura 3.15 – Placa do Circuito Montada

Para a fixação do hardware de controle à plataforma móvel, utilizou-se espaçadores para empilhar, sobre a plataforma, o dissipador de calor e a placa

de circuito impresso, sendo a fixação feita por parafusos e porcas. A Figura 3.16 associada à Tabela 3.3 mostra os elementos utilizados na montagem.

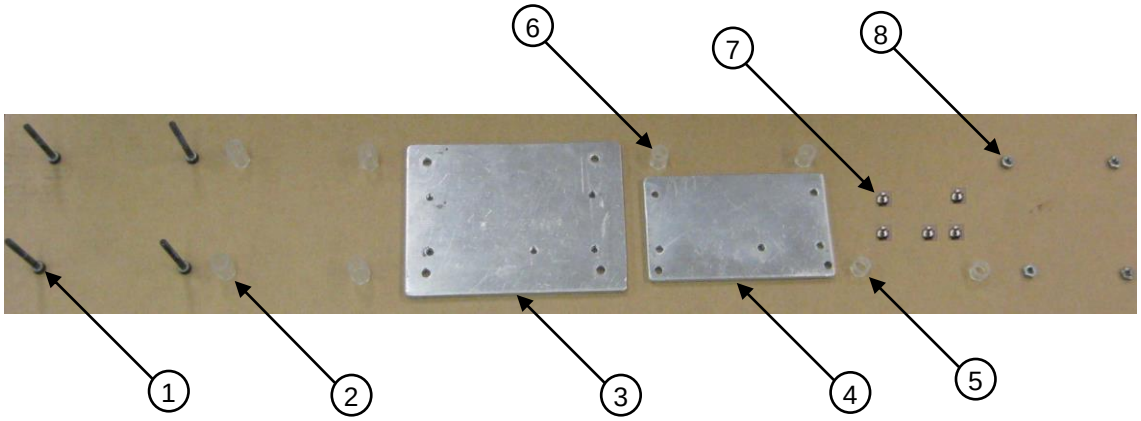


Figura 3.16 – Componentes de Montagem

Tabela 3.3 – Lista dos Componentes de Montagem

Item	Descrição	Quantidade
1	Parafuso Allen (M3x30mm)	4
2	Espaçador de poliestireno (ø7,5mm x 12,5mm)	4
3	Placa inferior de alumínio dissipadora de calor (3mm x 675mm x 925mm)	1
4	Placa inferior de alumínio dissipadora de calor (3mm x 470mm x 790mm)	1
5	Espaçador de poliestireno (ø7,5mm x 5,0mm)	2
6	Espaçador de poliestireno (ø7,5mm x 8,0mm)	2
7	Parafuso Allen (M3x7mm)	5
8	Porca (M3)	4

Os espaçadores de poliestireno foram feitos a partir do tambor transparente de uma caneta *BIC*®. Este material é interessante pelo fato de se

obter, através de lixação manual, uma boa precisão no comprimento do espaçador (precisão por volta de 0,1mm). Além disto, o custo e a disponibilidade do material são outras vantagens importantes. A função dos espaçadores é sustentar o empilhamento das placas, de modo a se obter dois vãos livres para acomodar os componentes da placa de circuito impresso e toda a fiação do robô.

Os parafusos Allen M3 x 7mm são usados para fixar os *drivers* SLA7024 e o regulador chaveado LM2575 no dissipador de calor (placas de alumínio). Usa-se ainda uma pasta térmica entre os elementos condutores para melhorar a condução de calor. As placas de alumínio possuem furos com rosca para os parafusos M3 x 7mm do dissipador, e furos lisos para os parafusos M3 x 30mm de sustentação (Figura 3.17). A placa superior é utilizada para permitir o contato entre o dissipador e os componentes a serem dissipados sem que haja interferência mecânica entre o dissipador e outros componentes da placa, com por exemplo, os conectores do motor de passo.

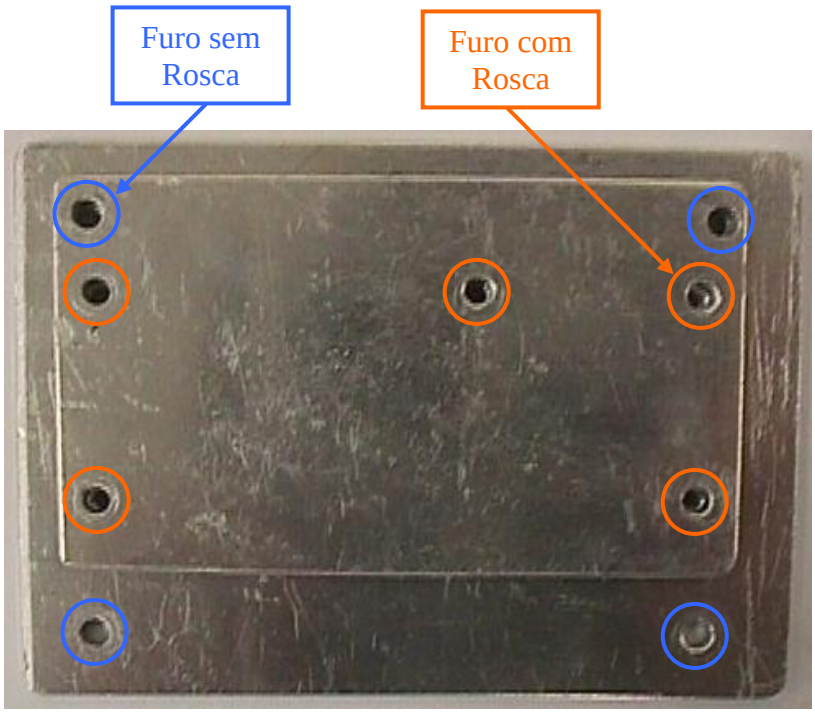


Figura 3.17 – Dissipador de Calor

As Figuras 3.18 e 3.19 mostram o protótipo parcialmente montado. A fim de se ter uma melhor clareza na visualização dos detalhes, mostra-se nestas figuras apenas a placa de circuito impresso (sem os componentes eletrônicos, bem como sem os parafusos M3 x 7mm do dissipador).

Figura 3.18 – Montagem do Hardware de Controle sobre a Plataforma (Vista Lateral)

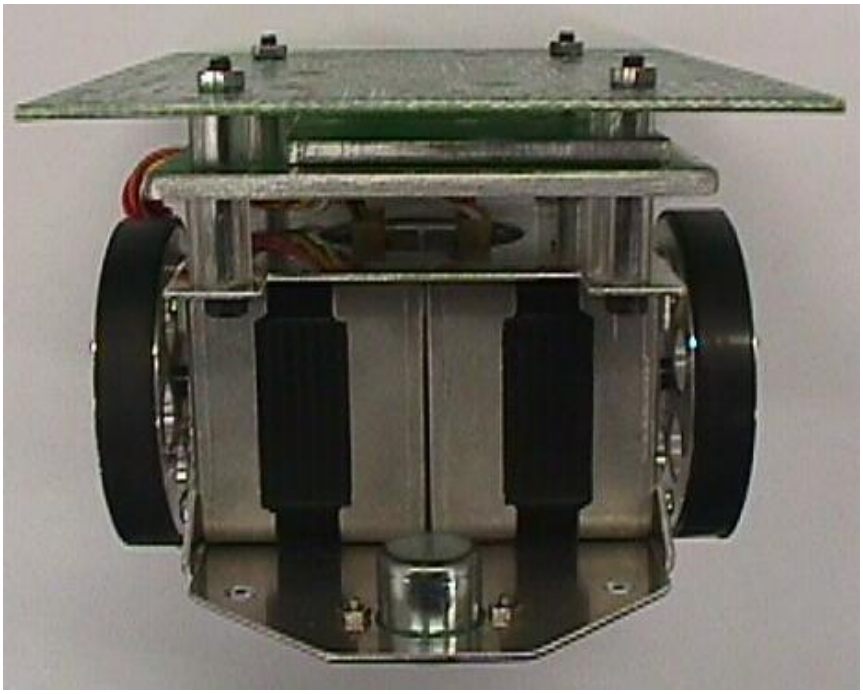


Figura 3.19 – Montagem do Hardware de Controle sobre a Plataforma (Vista Frontal)

Finalmente pode-se ver na Figura 3.20 o hardware de controle montado sobre a plataforma móvel. As baterias são acomodadas nos compartimentos frontal e traseiro da plataforma móvel, sendo as mesmas presas por fitas adesivas.

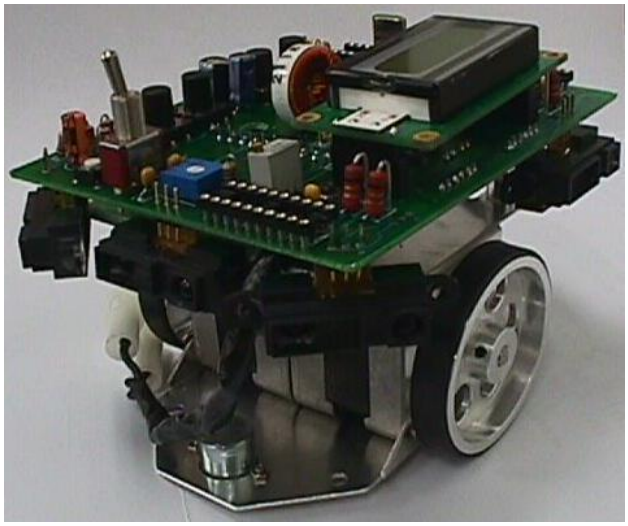


Figura 3.20 – Hardware de Controle Montado

4. SOFTWARE

O software do robô está dividido hierarquicamente em dois níveis: baixo nível e alto nível. A camada de baixo nível diz respeito ao software de controle, o qual trata da execução das tarefas essenciais do robô, como, por exemplo, o acionamento dos motores e a leitura dos sensores. A camada de alto nível refere-se ao software de aplicação do robô, isto é, o software específico para que o robô execute uma determinada tarefa.

Todo o programa está escrito em linguagem *C*, e o compilador utilizado é o *PCM C Compiler* do fabricante *CCS, Inc.* Este é um compilador específico para a família de microcontroladores *PIC* de 14 bits. O programa é gravado no microcontrolador *PIC16F873* do robô *Jerry* através do programador universal *TOPMAX* do fabricante *EETools, Corp.*

4.1 ACIONAMENTO DOS MOTORES DE PASSO

Uma vez que o robô utiliza motores de passo sem uma configuração de realimentação de posição, deve-se evitar a todo custo que o motor emperre ou perca passo. A solução para este tipo de problema é o acionamento com um perfil de velocidades e acelerações adequados.

Este robô está projetado para movimentar-se a baixas velocidades, e, assim, podemos admitir que o motor fornece um torque (T_{roda}) de cerca 0,05Nm. Portanto, em cada roda do robô, tem-se uma força (F_{roda}) de:

$$F_{roda} = \frac{T_{roda}}{\frac{\phi_{roda}}{2}} = \frac{0{,}05}{\frac{0{,}051}{2}} \cong 2N \qquad (4.1)$$

Como são duas rodas, a força de impulsão (F) do robô é:

$$F = F_{roda} \times 2 = 4N \qquad (4.2)$$

O robô possui uma massa (m) de cerca de 2 kg. Assim o robô deve ser acionado com uma aceleração (a) de:

$$a = \frac{F}{m} = \frac{4}{2} = 2m/s^2 \qquad (4.3)$$

Então, sabendo que o motor tem uma resolução de N_p = 200 passos/revolução, a aceleração em termos de números de passo (a_p) é dada por:

$$a_p = \frac{a}{\frac{\pi \phi_{roda}}{N_p}} = \frac{2}{\frac{\pi \times 0{,}051}{200}} \cong 2482 passos/s^2 \qquad (4.4)$$

Por segurança, seja a_p = 2000 passos/s^2.

Para acelerar o motor, deve-se emitir pulsos em intervalos de tempo decrescentes até que se atinja uma taxa máxima. Da mesma forma, para desacelerar o motor, deve-se aumentar o intervalo de tempo entre um pulso e outro até que o motor pare. Estes intervalos de tempo podem ser calculados a partir da equação de movimento com aceleração constante:

$$s = \frac{1}{2} a_p t^2 \qquad (4.5)$$

Sendo que aqui, s é a distância percorrida em número passos, a_p é a aceleração em passos/s^2, e t é o tempo decorrido em segundos.

Rearranjando a equação (4.5), tem-se:

$$t = \sqrt{\frac{2 \times s}{a_p}} \qquad (4.6)$$

Assim, finalmente, pode-se calcular a tabela de velocidades utilizada para o acionamento dos motores de passo:

Tabela 4.1 – Aceleração do Motor de Passo

s (passos)	t (segundos)	$\Delta t = t_i - t_{i-1}$ (segundos)	Freqüência (hertz)
0	0,000000	–	–
1	0,031623	0,031623	31,623
2	0,044721	0,013099	76,344
3	0,054772	0,010051	99,494
4	0,063246	0,008473	118,018
5	0,070711	0,007465	133,956
6	0,077460	0,006749	148,170
7	0,083666	0,006206	161,126
8	0,089443	0,005777	173,109
9	0,094868	0,005426	184,311
10	0,100000	0,005132	194,868
11	0,104881	0,004881	204,881
12	0,109545	0,004664	214,425
13	0,114018	0,004473	223,562

4.2 BIBLIOTECA DO SOFTWARE DE CONTROLE

O software de controle consiste de um conjunto de funções e rotinas que tratam da execução das tarefas essenciais do módulo de processamento. Estas tarefas essenciais podem ser enquadradas em três categorias, de acordo com os módulos do hardware de controle com os quais estas tarefas estão relacionadas. A primeira categoria compreende um conjunto de funções e rotinas para se fazer o acionamento dos motores; a segunda cuida dos

procedimentos de aquisição dos sinais provenientes dos sensores de distância; enquanto que a terceira categoria disponibiliza uma biblioteca para controlar os dispositivos de interface com o usuário. Estes três grupos de bibliotecas compõem o software de controle, o qual fornece uma plataforma para que o algoritmo de aplicação do robô possa ser executado pela camada de software de alto nível.

4.2.1 Acionamento dos Motores

A seguir são nomeadas e descritas as funções e rotinas que compõem esta biblioteca do software de controle. Esta categoria conta com três rotinas principais:

MOTORENABLE

Sintaxe: motorEnable (***enable***)

Parâmetros: ***enable*** é uma variável booleana.

Retorno: Indefinido.

Descrição: Habilita ou desabilita o funcionamento dos motores de passo (***enable***=*FALSE*: motores desligados; ***enable***=*TRUE*: motores ligados).

MOTORLEFT

Sintaxe: motorLeft (***step***, ***dir***)

Parâmetros: ***step*** é uma variável inteira de 16 bits não-negativa.
dir é uma variável booleana.

Retorno: Indefinido.

Descrição: Esta rotina aciona o motor do lado *esquerdo* do robô. A variável ***step*** informa o número de passos a ser girado pelo motor; e ***dir*** se refere ao sentido de rotação do motor (***dir***=*FALSE*: *sentido anti-horário*; ***dir***=*TRUE*: sentido *horário*).

MOTORRIGHT

Sintaxe: motorRight (***step***, ***dir***)

Parâmetros: ***step*** é uma variável inteira de 16 bits não-negativa. ***dir*** é uma variável booleana.

Retorno: Indefinido.

Descrição: Esta rotina aciona o motor do lado *direito* do robô. A variável ***step*** informa o número de passos a ser girado pelo motor; e ***dir*** se refere ao sentido de rotação do motor (***dir***=*FALSE*: *sentido horário*; ***dir***=*TRUE*: *sentido anti-horário*).

A partir destas três rotinas básicas, outras rotinas podem ser obtidas:

MOVEFWD

Sintaxe: moveFwd (***dist***)

Parâmetros: ***dist*** é uma variável inteira de 16 bits não-negativa.

Retorno: Indefinido.

Descrição: Rotina para movimentar o robô para *frente* em ***dist*** milímetros.

MOVEBWD

Sintaxe: moveBwd (***dist***)

Parâmetros: ***dist*** é uma variável inteira de 16 bits não-negativa

Retorno: Indefinido.

Descrição: Rotina para movimentar o robô para *trás* em ***dist*** milímetros.

TURNCCW

Sintaxe: turnCcw (***degree***)

Parâmetros: ***degree*** é uma variável inteira de 16 bits não-negativa.

Retorno: Indefinido.

Descrição: Rotina para girar o robô em torno do próprio eixo no *sentido anti-horário* de ***degree*** graus.

TURNCW

Sintaxe: turnCw (***degree***)

Parâmetros: ***degree*** é uma variável inteira de 16 bits não-negativa.

Retorno: Indefinido.

Descrição: Rotina para girar o robô em torno do próprio eixo no *sentido horário* de ***degree*** graus.

STOP

Sintaxe: stop ()

Parâmetros: Nenhum.

Retorno: Indefinido.

Descrição: Faz com que o robô pare imediatamente.

Tem-se ainda uma função para verificar se o robô está se movendo ou não. Isto é particularmente útil para gerenciar o uso de vários comandos de movimentação em seqüência.

ISITMOVING

Sintaxe: ***status*** = isItMoving ()

Parâmetros: Nenhum.

Retorno: ***status*** é uma variável booleana.

Descrição: Verifica se o robô esta em movimento (***status***=*FALSE*: o robô está parado; ***status***=*TRUE*: o robô está em movimento).

Há ainda uma rotina referente aos parâmetros de aceleração do motor de passo, discutidos anteriormente.

SETUPACCELERATION

Sintaxe: setupAcceleration ()

Parâmetros: Nenhum.

Retorno: Indefinido.

Descrição: Escreve na memória do microcontrolador os valores dos parâmetros de aceleração do motor de passo. Esta rotina deve ser executada antes do uso das demais rotinas de movimentação.

4.2.2 Sensoriamento do Ambiente

Esta biblioteca diz respeito ao uso dos sensores Sharp GP2D120. Este sensor gera uma saída analógica inversamente proporcional à distância entre o sensor e um anteparo (veja a Figura 3.8 no Capítulo 3). Esta saída analógica é lida pelo módulo conversor analógico-digital do microcontrolador PIC16F873, transformando-a em um número inteiro. A seguir são mostradas as funções e rotinas presentes nesta biblioteca.

SETUPSENSOR

Sintaxe: setupSensor ()

Parâmetros: Nenhum.

Retorno: Indefinido.

Descrição: Inicializa o módulo de sensoriamento.

GETSENSOR

Sintaxe: ***value*** = getSensor (***sens_id***)

Parâmetros: ***sens_id*** é uma variável inteira de 8 bits não-negativa. Valores válidos de ***sens_id*** devem pertencer ao intervalo [0, 4].

Retorno: ***value*** é uma variável inteira não-negativa.

Descrição: ***value*** recebe o valor gerado pelo sensor de distância identificado por ***sens_id***, sendo que ***sens_id*** corresponde a um dos cinco canais do módulo conversor analógico-digital do microcontrolador.

GETSENSORMEAN

Sintaxe: ***value*** = getSensorMean (***sens_id***)

Parâmetros: ***sens_id*** é uma variável inteira de 8 bits não-negativa. Valores válidos de ***sens_id*** devem pertencer ao intervalo [0, 4].

Retorno: ***value*** é uma variável inteira não-negativa.

Descrição: ***value*** recebe média aritmética de oito leituras consecutivas do sensor de distância identificado por ***sens_id***, sendo que ***sens_id*** corresponde a um dos

cinco canais do módulo conversor analógico-digital do microcontrolador.

4.2.3 Interface com o Usuário

As principais rotinas e funções da biblioteca de interface com o usuário são apresentadas a seguir.

LCDINITIALIZATION

Sintaxe: lcdInitialization ()

Parâmetros: Nenhum.

Retorno: Indefinido.

Descrição: Executa os procedimentos de inicialização do display de cristal líquido.

LCDPUTC

Sintaxe: lcdPutc (***c***)

Parâmetros: ***c*** é uma variável do tipo *caractere ASCII*.

Retorno: Indefinido.

Descrição: Envia um caractere ao display de cristal líquido. Existem três caracteres especiais:
'\f': limpa o display, e o cursor vai para o começo da primeira linha;

'\n': move o cursor para o começo da segunda linha;
'\b': corresponde ao comando de *backspace*.
Observação: Para o envio de um *string* de caracteres ao display, pode-se fazer uso da rotina *printf* do compilador, com a seguinte sintaxe:
printf (lcdPutc, "string").

LCDGETC

Sintaxe: ***c*** = lcdGetc (***x***, ***y***)

Parâmetros: ***x*** e ***y*** são variáveis inteiras de 8 bits não negativas.
Retorno: ***c*** é uma variável do tipo *caractere ASCII*.

Descrição: ***c*** recebe o caractere atual da posição (***x***, ***y***) do display.

WAITFORBTN

Sintaxe: ***button*** = waitForBtn ()

Parâmetros: Nenhum.

Retorno: ***button*** é uma variável inteira de 8 bits.

Descrição: Aguarda até que algum botão do robô seja pressionado, armazenando o valor correspondente na variável ***button***.

4.3 SOFTWARE DE APLICAÇÃO

O software de aplicação desenvolvido neste projeto tem o intuito de ilustrar o uso de todas as funcionalidades presentes neste protótipo. Para isto, desenvolveu-se um aplicativo onde o usuário tem a opção de executar testes para verificar o funcionamento dos motores e sensores do robô, bem como a opção de executar um algoritmo de exploração, no qual o robô vaga aleatoriamente pelo ambiente, desviando de obstáculos encontrados ao longo de sua trajetória.

4.3.1 Menu de Opções

A interface com o usuário é feita através de um menu de opções, o qual é exibido no display de cristal líquido e acessado pelos seguintes botões apresentados na Tabela 4.2:

Tabela 4.2 – Botões para Navegar no Menu de Opções

Botão	Descrição	Símbolo
NEXT	Navega para a direita ciclicamente entre as opções de um mesmo grupo do menu.	→
PREV	Navega para a esquerda ciclicamente entre as opções de um mesmo grupo do menu.	←
OK	Confirma a opção exibida no menu; Vai para o subgrupo de opções inferior ao atual.	✓
CANCEL	Cancela a opção exibida no menu; Vai para o subgrupo de opções superior ao atual.	✘

Ao se navegar pelo menu do robô, o display de cristal líquido exibe as opções mostradas na Figura 4.1:

4.3.2 Algoritmo de Exploração do Ambiente

A opção *Explore* do menu de opções executa a tarefa propriamente dita para a qual o robô foi programando. Esta tarefa consiste em explorar o ambiente, desviando de obstáculos. Para cumprir tal propósito, um algoritmo simples foi desenvolvido (Figura 4.2).

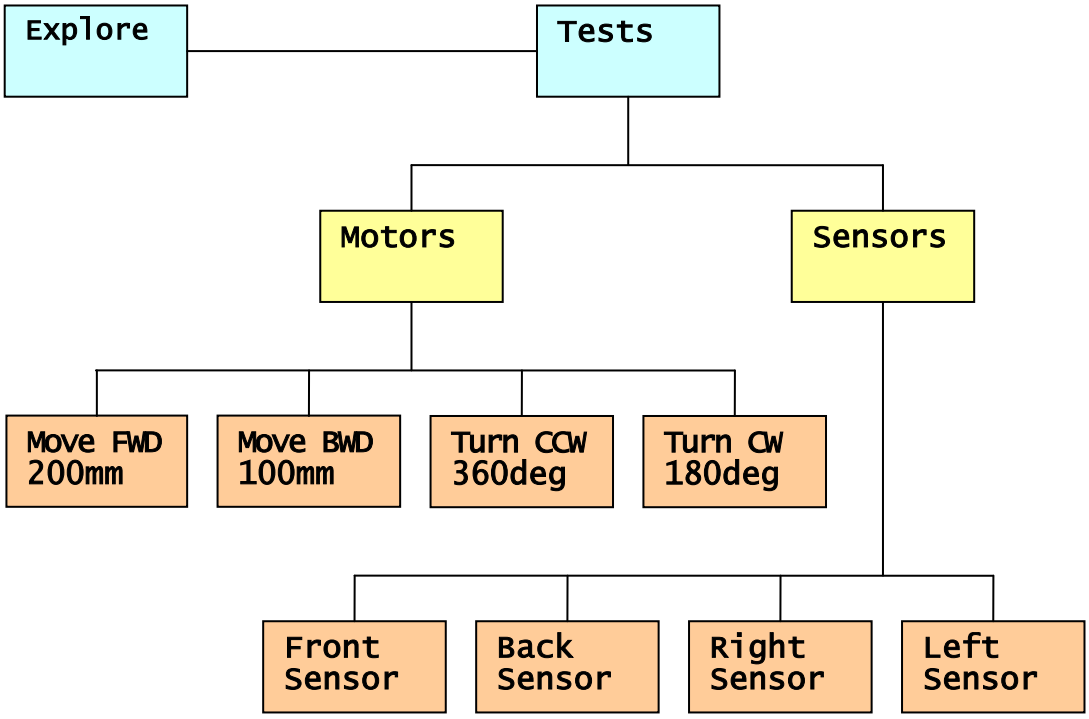


Figura 4.1 – Estrutura do Menu de Opções Exibido no Display

Este algoritmo consiste dos seguintes passos. Primeiramente é feita a leitura dos quatro sensores, os quais estão posicionados na frente, na traseira, e nas laterais do robô (posições J1, J2, J3 e J4, vide Figura 3.10). A seguir um número aleatório é gerado através de uma manipulação do valor do *timer* interno do microcontrolador. Este valor obtido é usado para que o algoritmo decida por alguma das quatro opções: o robô

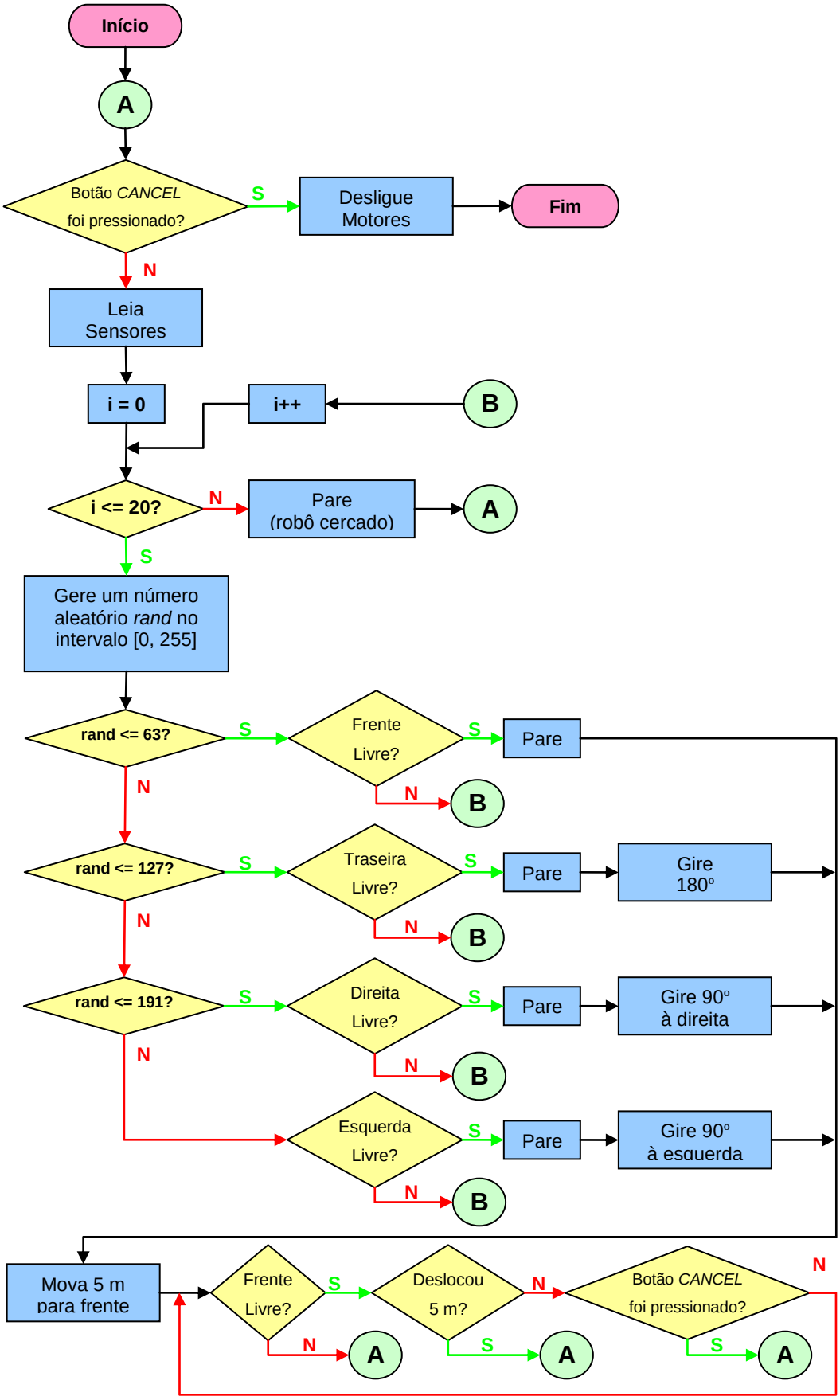


Figura 4.2 – Fluxograma do Algoritmo para Exploração do Ambiente

segue em frente, o robô segue na mesma direção em sentido oposto, o robô segue na direção perpendicular à direita, ou o robô segue na direção perpendicular à esquerda. Porém, antes que a movimentação na direção e sentido escolhidos comece, o algoritmo verifica se a opção escolhida está livre de obstáculos (isto é feito através da checagem do valor lido anteriormente pelo sensor correspondente). Se o caminho escolhido estiver bloqueado, volta-se para o passo em que um número aleatório é gerado, e, então, sorteia-se novamente uma das quatro opções de movimentação. Caso tenham sido feitas 20 tentativas de se encontrar um caminho livre, e em todas elas detectou-se a presença de obstáculos, o robô pára e considera que está cercado em todas as direções, voltando ao início do algoritmo. Caso o botão *CANCEL* tenha sido pressionado em algum momento, o algoritmo é finalizado. Por outro lado, voltando ao passo anterior, se o caminho aleatório escolhido estiver livre de obstáculos, o robô segue por este caminho. A partir deste ponto, monitora-se três situações para concluir se a execução deve voltar para o início do algoritmo. Estas situações são: o robô encontra um obstáculo pela frente, o robô desloca-se por 5 metros pelo caminho escolhido, o botão *CANCEL* é pressionado em algum momento. Na ocorrência de qualquer uma destas três situações, volta-se para o início do algoritmo. No caso da última situação (botão *CANCEL* pressionado), a execução do algoritmo é finalizada.

5 RESULTADOS

Ao longo do desenvolvimento deste trabalho, conduziu-se alguns testes e experimentos com a finalidade de definir e verificar os requisitos do projeto. Os resultados mais significativos destes testes e experimentos são apresentados e discutidos a seguir.

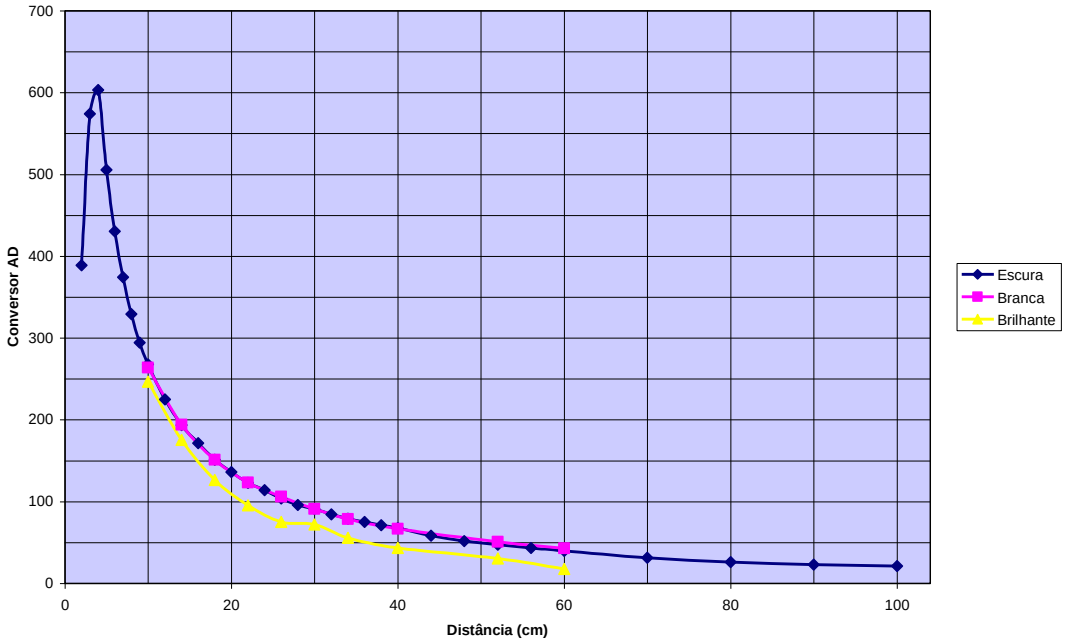
Primeiramente testou-se o circuito do módulo de acionamento. Nos testes, observou-se que, sem carga aplicada, o motor responde bem até uma freqüência de acionamento de cerca de 1kHz. Este valor corresponde a uma rotação de 300rpm, e uma velocidade linear do robô de cerca de 80cm/s, o que é um valor alto para os propósitos deste robô. A velocidade máxima especificada neste projeto é de até 18cm/s (223Hz). Nesta freqüência de acionamento, verificou-se que o motor fornece torque suficiente para a movimentação plena do robô.

Testes realizados no circuito do módulo de alimentação também mostram resultados satisfatórios. O componente LM2575 (regulador chaveado) regula corretamente a tensão em 5,0V para valores de entrada maiores que 7,0V e menores que 40V. A tensão nominal fornecida pelas baterias é de 14,4V, e, portanto, as variações observadas neste valor nominal de tensão, na faixa de 13V a 18V, estão dentro das especificações. Foi também observado que este conjunto de baterias provê uma autonomia de funcionamento ininterrupto ao robô de cerca de trinta minutos, o que é suficiente para os propósitos deste protótipo.

Acreditava-se que sensor de distância apresentaria restrições quanto ao tipo de superfície do anteparo (obstáculo). No entanto, testes realizados com um sensor *SHARP GP2D120* mostraram que este tipo de sensor é pouco sensível ao tipo de superfície. Nas especificações do sensor consta que anteparo deve ser um objeto de superfície opaca. A fim de se verificar a influência do tipo superfície do objeto na resposta do sensor, realizou-se ensaios cuidadosos para três tipos superfícies: opaca escura (papel pardo), opaca branca (papel sulfite) e brilhante (papel alumínio). Os resultados são mostrados na Figura 5.1. A faixa de valores ensaiada comum para os três tipos

de superfície foi de 10cm a 60cm. Nesta faixa praticamente não se observa diferença na resposta entre as superfícies branca e escura; e se vê uma pequena discrepância entre estas duas superfícies e a superfície brilhante. Isto mostra que, adotando uma certa tolerância para a medição, o sensor se mostra pouco sensível à influência do tipo de superfície. Desta forma, confirmou-se que os dados adquiridos pelo sistema de sensoriamento são confiáveis o suficiente para serem usados pelo robô em tomadas de decisão.

Figura 5.1: Influência do Tipo de Superfície na Resposta do Sensor *SHARP GP2D120*



A Tabela 5.1 resume os parâmetros de projeto analisados nos testes e experimentos apresentados acima.

Finalmente, para validar tanto o protótipo construído e como o software desenvolvido, foram realizados testes com o robô em duas situações particulares. Na primeira situação, o robô foi colocado para explorar uma pequena área delimitada (Figura 5.1). Na segunda situação, permitiu-se ao robô explorar livremente um amplo corredor (Figura 5.2).

Tabela 5.1 – Parâmetros de Projeto Analisados

Parâmetros	Valores	Comentários
Freqüência Máxima de Acionamento	223 Hz	A esta freqüência, há torque suficiente para a movimentação do robô
Velocidade Máxima do Robô	18 cm/s	Velocidade adequada para os propósitos do projeto
Tensão Nominal das Baterias	14,4 V	Variação no Valor Nominal na Faixa de 13 V a 18 V
Autonomia do Robô	Aproximadamente 30 minutos	Considerando Funcionamento Pleno e Ininterrupto
Resposta do Sensor de Distância	4 cm a 30 cm	Pouco Sensível ao Tipo de Superfície do Anteparo

Figura 5.1 – Exploração de uma Pequena Área Delimitada

Figura 5.2 – Exploração Livremente de um Amplo Corredor

Em ambas as situações de exploração, foi possível constatar que o robô cumpriu satisfatoriamente a tarefa para o qual foi programado, ou seja, vagou aleatoriamente pelo ambiente sem colidir com nenhum obstáculo.

6 CONCLUSÕES

Neste projeto de formatura pôde-se chegar a um protótipo de robô móvel autônomo plenamente funcional, bem como desenvolver um aplicativo de software para que o robô cumprisse uma determinada tarefa satisfatoriamente.

O desenvolvimento deste microrrobô passou por todas as etapas de projeto de hardware e software. Primeiramente fez-se o projeto do hardware de controle, passando em seguida para concepção do layout da placa de circuito impresso. Com o hardware de controle finalizado, partiu-se para o desenvolvimento do software de controle. Ao término desta etapa, obteve-se uma plataforma de desenvolvimento, através da qual pôde-se finalmente desenvolver um aplicativo para exploração de ambiente.

Deve-se notar que esta plataforma de desenvolvimento, composta pelo hardware e software de controle, possibilita o desenvolvimento de aplicativos com um grau bem maior de complexidade do que o apresentado. Desta forma, há um grande potencial para que trabalhos futuros venham ser desenvolvidos tendo como base este robô. Uma sugestão promissora, com fins acadêmicos, é o uso deste protótipo para resolver labirintos, já que este projeto fornece um robô com uma estrutura adequada para este propósito. Também é sugerido o uso deste trabalho como base para a construção de novas versões do robô *Jerry*.

REFERÊNCIAS

ALLEGRO MICROSYSTEMS, INC. **SLA7024M, SLA7026M, and SMA7029M.** Disponível em: <http://www.allegromicro.com/en/Products/Part_Numbers/97024/97024.pdf>. Acessado em: 24 nov 2007.

CCS, INC. **C Compile Reference Manual.** Disponível em: <http://www.ccsinfo.com/ downloads/ccs_c_manual.pdf>. Acessado em: 15 abr. 2007.

COSTA, A. H. R. **Robótica Móvel Inteligente:** Progressos e Desafios. 2003. 1v. Tese (Livre Docência) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2003.

CRYSTALFONTZ AMERICA, INC**. CFAH0802A-GYH-JP:** 8x2 STN, Positive Transflective Yellow-Green LCD with Green LED Backlight. Disponível em: <http://www.datasheetarchive.com/CFAH0802A-YYE-JP-datasheet.html>. Acessado em: 24 nov 2007.

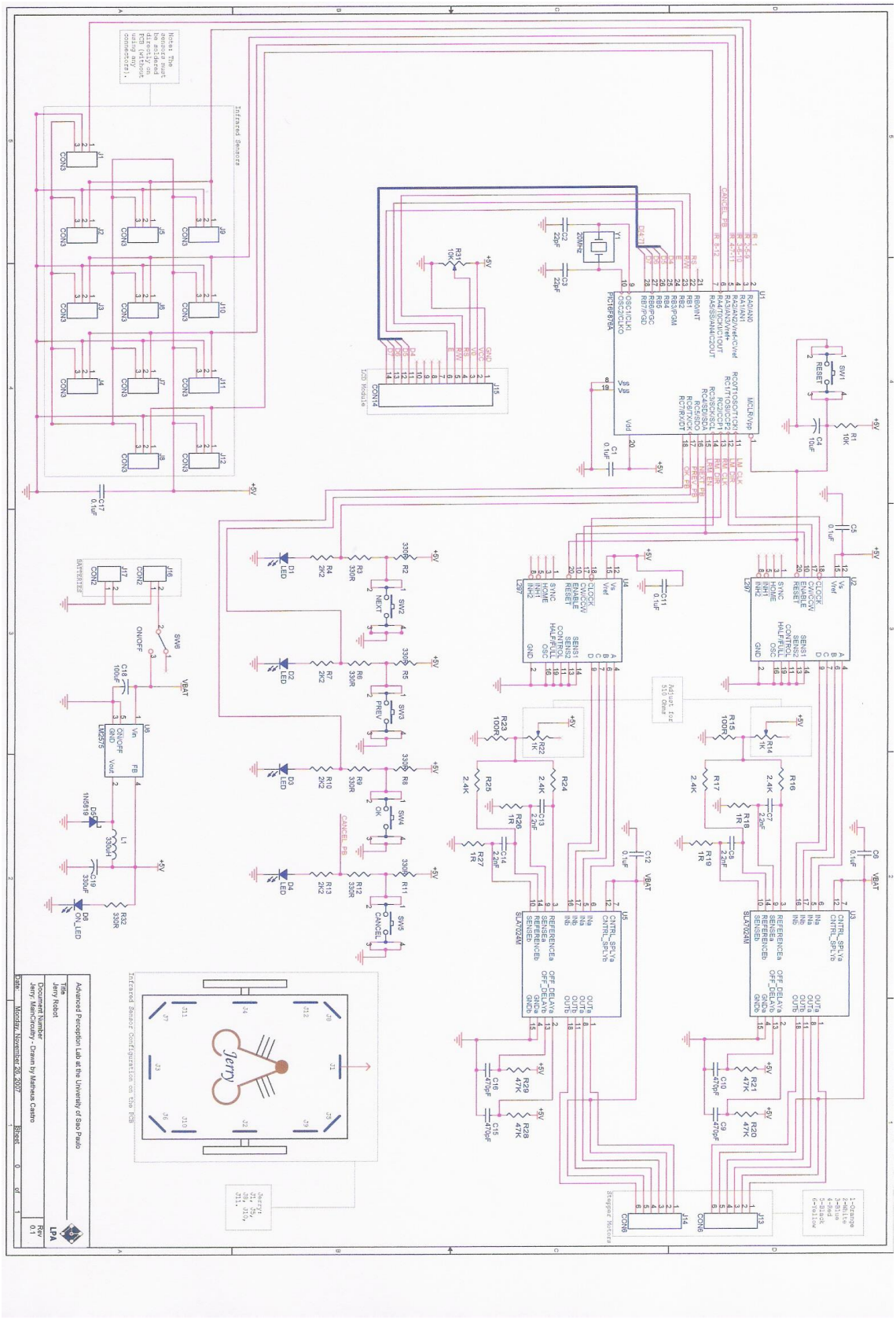
KENJO, T. **Stepping Motors and their Microprocessor Controls.** 1ª edição. Oxford University Press, 1984.

MICROCHIP TECHNOLOGY INC. **PIC16F87XA Data Sheet:** 28/40/44-Pin Enhanced Flash Microcontrollers. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>. Acessado em: 15 abr. 2007.

NATIONAL SEMICONDUCTOR CORPORATION. **LM1575/LM2575/ LM2575HV**. Disponível em: <http://cache.national.com/ds/LM/LM1575.pdf>. Acessado em: 24 nov 2007.

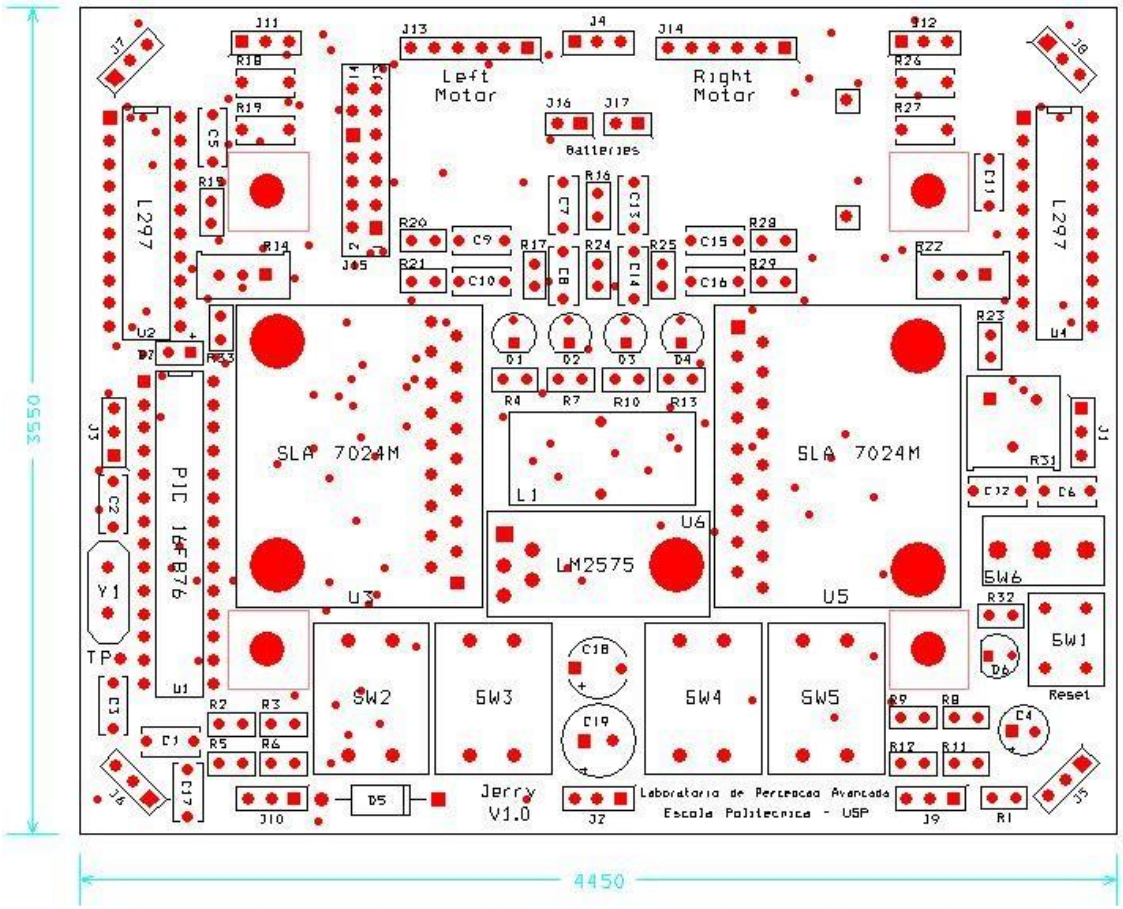
STMICROELETRONICS COMPANY. **L297:** Stepper Motor Controllers. Disponível em: <http://www.st.com/stonline/products/literature/ds/1334/l297.pdf>. Acessado em: 24 nov 2007.

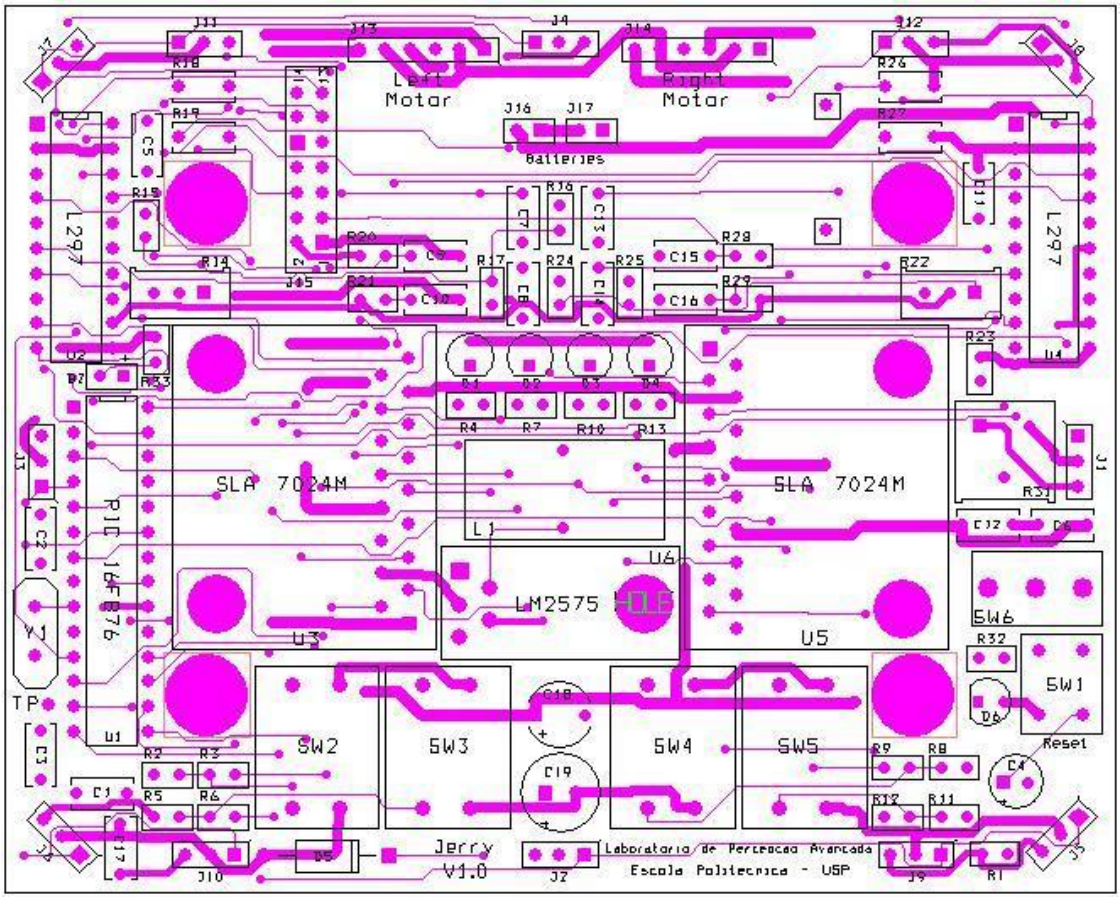
APÊNDICE A - Esquema do Circuito Elétrico

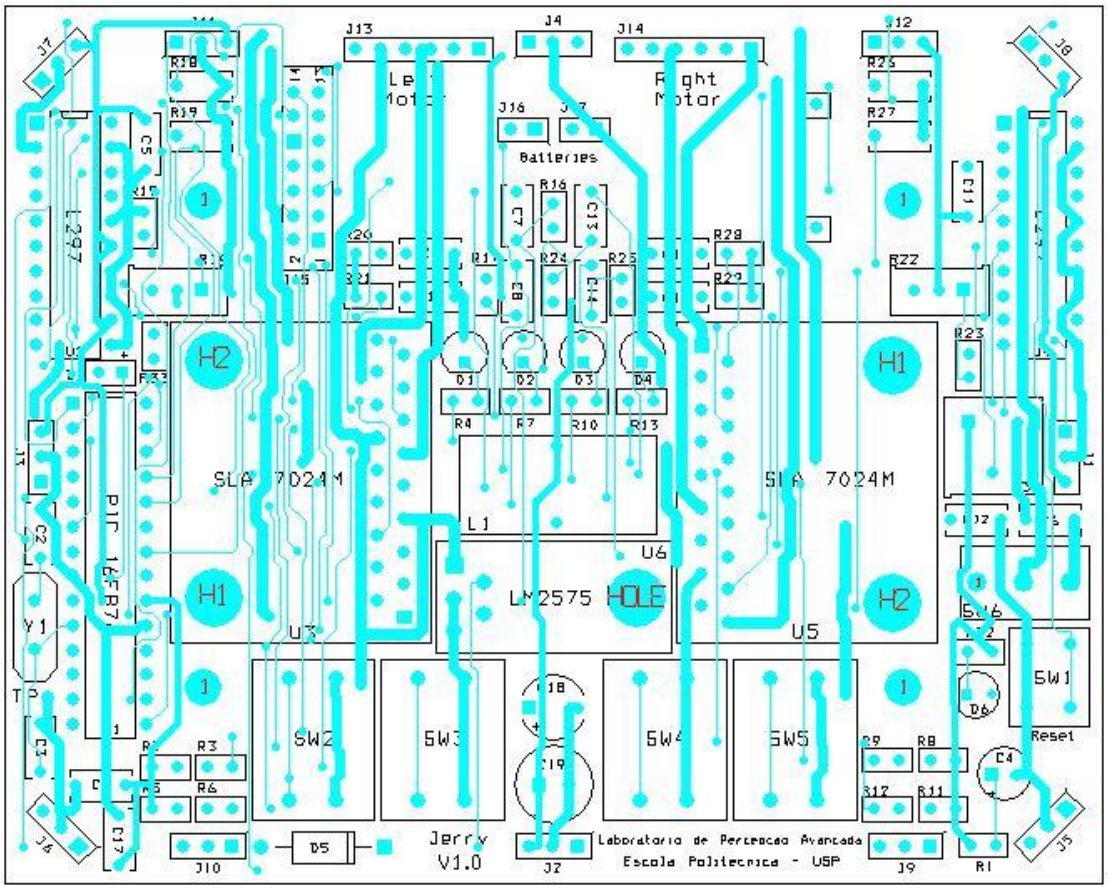


APÊNDICE B - Layout da Placa de Circuito Impresso

A seguir seguem desenhos referentes ao layout da placa de circuito impresso do hardware de controle. As regiões em vermelho representam furos passantes na placa. As linhas em magenta representam as trilhas de circuito na face superior da placa, enquanto que as linhas em ciano mostram as trilhas na face inferior. A placa possui dimensões de 4,450 pol x 3,550 pol (113,0 mm x 90,2 mm).




Left
Motor
Right
Motor
Batteries
L297
SLA 7024M
SLA 7024M
LM2575 HOLE
SW2
SW3
SW4
SW5
SW1
Reset
SW6
Jerry
V1.0
Laboratorio de Percepcao Avancada
Escola Politecnica - USP


Batteries
L297
SLA 7024M
SLA 7024M
LM2575 HOLE
SW2
SW3
SW1
Reset
Jerry
V1.0
Laboratorio de Percepcao Avancada
Escola Politecnica - USP

APÊNDICE C - Código Fonte do Programa

```
/****************************************
Explorer.c
Este programa, em linguagem C e compilado
atraves do PCM C Compiler do fabricante
CCS, Inc., eh um aplicativo para Robo
Jerry versao 1.0, ilustrando o uso de
todas as funcionalidades presentes neste
prototipo. A interface com o usuario eh
feita atraves de um menu de opcoes,
exibido no display de cristal liquido e
acessado pelos botoes: NEXT(=proxima
opcao), PREV(=opcao anterior), OK
(=confirma opcao), e CANCEL (=cancela
opcao). Neste aplicativo tem-se a opcao
de executar testes para verificar o
funcionamento dos motores e sensores do
robo, bem como a opcao de executar uma
rotina de exploracao do ambiente, onde o
robo vaga indefinidamente pelo ambiente
desviando de obstaculos.
****************************************/

//inclui a biblioteca do PIC16F873
#include <16F873.h>
//Configuration words para o PIC16F873
#fuses HS,NOPROTECT,NOWDT,NOPUT,NOBROWNOUT
//Habilita o uso da funcao delay
#use delay (clock=20000000)

#define STEP_LEFT PIN_C0 //pulsos de clock p/ o motor de passo
//esquerdo
#define DIR_LEFT PIN_C1 //direcao de rotacao p/ o motor de passo
//esquerdo; 0->CCW
#define STEP_RIGHT PIN_C2 //pulsos de clock p/ o motor de passo
//esquerdo
#define DIR_RIGHT PIN_C3 //direcao de rotacao p/ o motor de passo
//direito; 0->CCW
#define MT_ENABLE PIN_C4 //sinal de habilitacao do funcionamento
//dos motores; 1->habilita
#define LED1 PIN_C5 //pinos correspodentes aos 4 leds de
//sinalizacao
#define LED2 PIN_C6
#define LED3 PIN_C7
#define LED4 PIN_A4
#define SW2 PIN_C5 //pinos correspondentes aos 4
//pushbuttons
#define SW3 PIN_C6
#define SW4 PIN_C7
#define SW5 PIN_A4
#define OPCYCLE_FREQ 65036 //define a frequencia de interrupcao;
//65036->200us->5kHz
#define NEXT 0x07 //mascara p/ leitura do botao SW2
#define PREV 0x0B //mascara p/ leitura do botao SW3
#define OK 0x0D //mascara p/ leitura do botao SW4
#define CANCEL 0x0E //mascara p/ leitura do botao SW5
#define OFF 0 //usados como parametros para
//habilitar/desabilitar os motores
#define ON 1
```

```
#define FWD 0 //usados como parametros para definir o
//sentido do movimento do robo (frente/tras)
#define BWD 1
#define SPEED_MAX 13 //nivel de velocidade maximo para os
//motores
#define LCD_TYPE 2 //configuracao do display: 0=caracter
//5x7 pontos, 1=5x10, 2=2 linhas
#define LCD_LINE_TWO 0x40 //endereco no LCD RAM para a segunda
//linha do display
#define SENS_FRONT 0 //canal AD para o sensor frontal
#define SENS_BACK 2 //canal AD para o sensor traseiro
#define SENS_RIGHT 1 //canal AD para o sensor lateral direito
#define SENS_LEFT 3 //canal AD para o sensor lateral
//esquerdo
#define LIMIT_DIST 0x2A //valor referente a distancia limite que
//o robo considera como obstaculo
//Variaveis globais
int led;//corresponde ao output dos leds:
//led=0b'XXXX<LED1><LED2><LED3><LED4>', active high
int btn;//corresponde ao input dos botoes:
//btn=0b'XXXX<SW2><SW3><SW4><SW5>', active low
//Variaveis usadas no algoritmo de leitura dos pushbuttons
int btn_prev;
int btn_current;
//Variaveis usadas no acionamento dos motores de passo
unsigned int mtl_speed; unsigned int mtr_speed;
unsigned long mtl_step_current;
unsigned long mtr_step_current;
unsigned long mtl_step_target;
unsigned long mtr_step_target;
unsigned int mtl_counter;
unsigned int mtr_counter;


//-----------------------------------------------//
//------------INTERFACE COM USUARIO------------//
//-----------------------------------------------//
//Esta estrutura eh referenciada a pinos de I/O do microcontrolador
//para ter acesso aos pinos do LCD. Os bits estao em ordem crescente.
//Por exemplo, ENABLE eh o pino B3
struct lcd_pin_map {
  boolean unused;
  boolean rs;
  boolean rw;
  boolean en;
  int data: 4;
} lcd;
#byte lcd=6//coloca a estrutura inteira no PORTB (no endereco 6)
//Este byte deve ser enviado ao LCD para fazer sua inicializacao
byte CONST LCD_INIT_STRING[4] = {0x20 | (LCD_TYPE << 2), 0x08, 0x01,
0x06};
//Usados para configurar a direcao do I/O port
STRUCT lcd_pin_map const LCD_WRITE = {0,0,0,0,0};//para o modo de
//escrita, todos os pinos sao saidas
STRUCT lcd_pin_map const LCD_READ = {0,0,0,0,15};//para o modo de
//leitura, os pinos de dados sao entradas
//Leitura do byte enviando pelo LCD
byte lcdReadByte();
//Envio de um nibble para o LCD
void lcdSendNibble(byte n);
```

```
//Envio de um byte para o LCD
void lcdSendByte(byte address,byte n);
//Procedimento de inicializacao do LCD
void lcdInitialization();
//Vai para a posicao (x,y) no display; Ex: (2,1)->inicio da segunda
//linha
void lcdGoToXY(byte x,byte y);
//Envia um caractere ao LCD
//\f->Limpa display, cursor no comeco da primeira linha
//\n->Move para o comeco da segunda linha
//\b->Backspace
void lcdPutc(char c);
//Obtem o caractere da posicao (x,y) no display
char lcdGetc(byte x,byte y);
//Funcao que detecta que algum botao foi pressionado
int waitForBtn();

//-------------------------------------//
//------------MOVIMENTACAO-------------//
//-------------------------------------//
//Rotina para habilitar ou desabilitar o acionamento do motor
void motorEnable(short enable);
//Rotina para acionar a roda esquerda do robo
void motorLeft(unsigned long step, short dir);
//Rotina para acionar a roda direta do robo
void motorRight(unsigned long step, short dir);
//Rotina para movimentar o robo para frente em 'dist' mm
void moveFwd(unsigned long dist);
//Rotina para movimentar o robo para tras em 'dist' mm
void moveBwd(unsigned long dist);
//Rotina para girar o robo em torno do proprio eixo no sentido
//anti-horario de 'degree' graus
void turnCcw(unsigned long degree);
//Rotina para gira o robo em torno do proprio eixo no sentido horario
//de 'degree' graus
void turnCw(unsigned long degree);
//Faz com que o robo pare imediatamente
void stop();
//Verifica se o robo esta em movimento
short isItMoving();
//Gera o proximo valor de mtx_speed
unsigned int nextSpeed(unsigned long remaining, unsigned int speed);
//Escreve na eeprom os valores dos parametros de aceleracao do motor
//de passo
void setupAcceleration();

//-------------------------------------//
//------------SENSORIAMENTO------------//
//-------------------------------------//
//Inicializacao do modulo de sensoriamento
void setupSensor();
//Le o valor gerado pelo sensor de distancia 'sens_id'
unsigned int getSensor(unsigned int sens_id);
//Obtem a media de 8 leituras do sensor de distancia 'sens_id'
unsigned int getSensorMean(unsigned int sens_id);
//-------------------------------------//
//------------APLICATIVO---------------//
//-------------------------------------//
//o robo explora o ambiente desviando de obstaculos
void explore();
```

```
//-----------------------------------------//
//------------PROGRAMA PRINCIPAL-----------//
//-----------------------------------------//
Main(){
  int buttons;
  unsigned int sensor;
  //Inicializa o timer1: recebe clock
  //interno (20MHz) e é divido por 2.Ou seja,
  //incrementa a cada 0,4us[=inv((fosc/4)/2)]
  setup_timer_1(T1_INTERNAL | T1_DIV_BY_2);
  //Inicializacao do modulo de sensoriamento
  setupSensor();
  //Escreve na eeprom os valores dos parametros da aceleracao do motor
  //de passo
  setupAcceleration();
  enable_interrupts(global);//habilita as interrupcoes
  enable_interrupts(int_timer1);//habilita a interrupção int_timer1
  mtl_speed=0;
  mtr_speed=0;
  mtl_step_current=0;
  mtr_step_current=0;
  mtl_step_target=0;
  mtr_step_target=0;
  mtl_counter=0;
  mtr_counter=0;
  motorEnable(OFF);//motores desabilitaods
  led=0x0F;
  btn=0x0F;
  btn_prev=0x0F;
  btn_current=0x0F;

  //Verificacao do funcionamento dos LEDs de sinalizacao
  led=0x0F;
  delay_ms(500);
  led=0;
  delay_ms(500);
  led=0x0F;
  delay_ms(500);
  led=0x0A;
  delay_ms(500);
  led=0x05;
  delay_ms(500);
  led=0;
  lcdInitialization();//inicializacao do display
  printf(lcdPutc, "\fJerry\nv1.0");
  delay_ms(500);

  //Menu de opcoes para controlar as funcionalidades do robo
  //As opcoes sao exibidas no LCD, e o usuario as acessa atraves do
  //botoes
  //botao NEXT: proxima opcao do menu
  //botao PREV: opcao anterior do menu
  //botao OK: ativa a opcao exibida no menu, ou entra no subdiretorio
  //correspondente
  //botao CANCEL: cancela o aplicativo em execucao, ou retorna ao
  //diretorio um nivel acima do diretorio atual
MENU1:
  printf(lcdPutc, "\fExplore ");
  buttons=waitForBtn();
```

```
  switch(buttons){
    case NEXT:
      goto MENU2;
    break;
    case PREV:
      goto MENU2;
    break;
    case OK:
      explore();
    break;
    case CANCEL:
      goto MENU1;
    break;
  }//endswitch
  goto MENU1;

MENU2:
  printf(lcdPutc, "\fTests ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU1;
    break;
    case PREV:
      goto MENU1;
    break;
    case OK:
      goto MENU2_1;
    break;
    case CANCEL:
      goto MENU2;
    break;
}//endswitch
goto MENU2;

MENU2_1:
  printf(lcdPutc, "\fMotors ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_2;
    break;
    case PREV:
      goto MENU2_2;
    break;
    case OK:
      goto MENU2_1_1;
    break;
    case CANCEL:
      goto MENU2;
    break;
  }//endswitch
  goto MENU2_1;

MENU2_2:
  printf(lcdPutc, "\fSensors ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_1;
```

```
    break;
    case PREV:
      goto MENU2_1;
    break;
    case OK:
      goto MENU2_2_1;
    break;
    case CANCEL:
      goto MENU2;
    break;
  }//endswitch
  goto MENU2_2;

MENU2_1_1:
  printf(lcdPutc, "\fMove FWD\n200mm ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_1_2;
    break;
    case PREV:
      goto MENU2_1_4;
    break;
    case OK:
      motorEnable(ON);
      moveFwd(200);
      while(isItMoving());//espera o robo terminar o movimento
      motorEnable(OFF);
    break;
    case CANCEL:
      goto MENU2_1;
    break;
  }//endswitch
  goto MENU2_1_1;

MENU2_1_2:
  printf(lcdPutc, "\fMove BWD\n100mm ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_1_3;
    break;
    case PREV:
      goto MENU2_1_1;
    break;
    case OK:
      motorEnable(ON);
      moveBwd(100);
      while(isItMoving());//espera o robo terminar o movimento
      motorEnable(OFF);
    break;
    case CANCEL:
      goto MENU2_1;
    break;
  }//endswitch
  goto MENU2_1_2;

MENU2_1_3:
  printf(lcdPutc, "\fTurn Ccw\n360deg ");
  buttons=waitForBtn();
```

```
  switch(buttons){
    case NEXT:
      goto MENU2_1_4;
    break;
    case PREV:
      goto MENU2_1_2;
    break;
    case OK:
      motorEnable(ON);
      TurnCcw(360);
      while(isItMoving());//espera o robo terminar o movimento
      motorEnable(OFF);
    break;
    case CANCEL:
      goto MENU2_1;
    break;
  }//endswitch
  goto MENU2_1_3;

MENU2_1_4:
  printf(lcdPutc, "\fTurn Cw\n180deg ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_1_1;
    break;
    case PREV:
      goto MENU2_1_3;
    break;
    case OK:
      motorEnable(ON);
      TurnCw(180);
      while(isItMoving());//espera o robo terminar o movimento
      motorEnable(ON);
    break;
    case CANCEL:
      goto MENU2_1;
    break;
  }//endswitch
  goto MENU2_1_4;

MENU2_2_1:
  printf(lcdPutc, "\fFront \nSensor ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_2_2;
    break;
    case PREV:
      goto MENU2_2_4;
    break;
    case OK:
      printf(lcdPutc, "\fSF = %02x ", getSensorMean(SENS_FRONT));
      delay_ms(1000);
    break;
    case CANCEL:
      goto MENU2_2;
    break;
  }//endswitch
  goto MENU2_2_1;
```

```
MENU2_2_2:
  printf(lcdPutc, "\fBack \nSensor ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_2_3;
    break;
    case PREV:
      goto MENU2_2_1;
    break;
    case OK:
      printf(lcdPutc, "\fSB = %02x ", getSensorMean(SENS_BACK));
      delay_ms(1000);
    break;
    case CANCEL:
      goto MENU2_2;
    break;
  }//endswitch
  goto MENU2_2_2;

MENU2_2_3:
  printf(lcdPutc, "\fRight \nSensor ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_2_4;
    break;
    case PREV:
      goto MENU2_2_2;
    break;
    case OK:
      printf(lcdPutc, "\fSR = %02x ", getSensorMean(SENS_RIGHT));
      delay_ms(1000);
    break;
    case CANCEL:
      goto MENU2_2;
    break;
  }//endswitch
  goto MENU2_2_3;

MENU2_2_4:
  printf(lcdPutc, "\fLeft \nSensor ");
  buttons=waitForBtn();
  switch(buttons){
    case NEXT:
      goto MENU2_2_1;
    break;
    case PREV:
      goto MENU2_2_3;
    break;
    case OK:
      printf(lcdPutc, "\fSL = %02x ", getSensorMean(SENS_LEFT));
      delay_ms(1000);
    break;
    case CANCEL:
      goto MENU2_2;
    break;
  }//endswitch
  goto MENU2_2_4;
```

```
}//end main


//-----------------------------------------------//
//------------INTERRUPCAO PRINCIPAL------------//
//-----------------------------------------------//
//Rotina de interrupcao (interrompe c/ overflow do timer1) para a
execucao
//das tarefas iterativas do microcontrolador: acionamento dos motores,
//acionamento dos leds, e leitura dos botoes.
#int_timer1
OpCycle(){
  unsigned long mtl_remaining;
  unsigned long mtr_remaining;

  //Sempre que a rotina de interrupcao eh chamada
  //o timer1 eh "setado" para OPCYCLE_FREQ, e conta de
  //(OPCYCLE_FREQ) até 65535(16-bit timer), perfazendo
  //(65536 - OPCYCLE_FREQ) x 0,4us; neste instante a
  //interrupcao eh acionada novamente
  set_timer1(OPCYCLE_FREQ);
  //MOTOR DRIVE
  if(mtl_step_target!=0){
    if(mtl_counter<read_eeprom(mtl_speed))//temporizador para gerar a
    //frequencia de acionamento
      mtl_counter++;
    else{
      output_low(STEP_LEFT);//pulso de comando de passo
      delay_us(4);
      output_high(STEP_LEFT);
      mtl_counter=0;
      mtl_step_current++;
      mtl_remaining=mtl_step_target-mtl_step_current;
      mtl_speed=nextSpeed(mtl_remaining, mtl_speed);
      if(mtl_remaining==0){//verifica se o motor atingiu a posicao
        mtl_step_target=0;
        mtl_step_current=0;
        mtl_speed=0;
      }//endif mtl_remaining
    }//endelse
  }//endif mtl_step_target

  if(mtr_step_target!=0){
    if(mtr_counter<read_eeprom(mtr_speed))//temporizador para gerar a
    //frequencia de acionamento
      mtr_counter++;
    else{
      output_low(STEP_RIGHT);//pulso de comando de passo
      delay_us(4);
      output_high(STEP_RIGHT);
      mtr_counter=0;
      mtr_step_current++;
      mtr_remaining=mtr_step_target-mtr_step_current;
      mtr_speed=nextSpeed(mtr_remaining, mtr_speed);
      if(mtr_remaining==0){//verifica se o motor atingiu a posicao
        mtr_step_target=0;
        mtr_step_current=0;
        mtr_speed=0;
      }//endif mtr_remaining
```

```
    }//endelse
  }//endif mtr_step_target

  //BTNs
  btn_prev=btn_current;
  btn_current=input(SW2);
  btn_current=(btn_current<<1)+input(SW3);
  btn_current=(btn_current<<1)+input(SW4);
  btn_current=(btn_current<<1)+input(SW5);
  if(btn_prev==btn_current)
    btn=btn_current;
  //LEDs
  if((led&0x01)==0x01)
    output_high(LED4);
  else
    output_low(LED4);
  if((led&0x02)==0x02)
    output_high(LED3);
  else
    output_low(LED3);
  if((led&0x04)==0x04)
    output_high(LED2);
  else
    output_low(LED2);
  if((led&0x08)==0x08)
    output_high(LED1);
  else
    output_low(LED1);
  }//end interrupcao


//-----------------------------------------------//
//------------INTERFACE COM USUARIO------------//
//-----------------------------------------------//
//Leitura do byte enviando pelo LCD
byte lcdReadByte(){
  byte low,high;

  set_tris_b(LCD_READ);
  lcd.rw = 1;
  delay_cycles(1);
  lcd.en = 1;
  delay_cycles(1);
  high = lcd.data;
  lcd.en = 0;
  delay_cycles(1);
  lcd.en = 1;
  delay_us(1);
  low = lcd.data;
  lcd.en = 0;
  set_tris_b(LCD_WRITE);

  return((high<<4)|low);
}

//Envio de um nibble para o LCD
void lcdSendNibble(byte n){
  lcd.data=n;
  delay_cycles(1);
  lcd.en=1;
```

```
  delay_us(2);
  lcd.en=0;
}

//Envio de um byte para o LCD
void lcdSendByte(byte address, byte n){
  lcd.rs=0;
  while(bit_test(lcdReadByte(), 7));
  lcd.rs=address;
  delay_cycles(1);
  lcd.rw=0;
  delay_cycles(1);
  lcd.en=0;
  lcdSendNibble(n>>4);
  lcdSendNibble(n&0xf);
}

//Procedimento de inicializacao do LCD
void lcdInitialization(){
  byte i;

  set_tris_b(LCD_WRITE);
  lcd.rs = 0;
  lcd.rw = 0;
  lcd.en = 0;
  delay_ms(15);
  for(i=1;i<=3;i++){
    lcdSendNibble(3);
    delay_ms(5);
  }
  lcdSendNibble(2);
  for(i=0;i<=3;i++)
    lcdSendByte(0,LCD_INIT_STRING[i]);
  lcdSendByte(0,0x0E);
  lcdSendByte(0,0x01);
}

//Vai para a posicao (x,y) no display; Ex: (2,1)->inicio da segunda
//linha
void lcdGoToXY(byte x, byte y){
  byte address;
  if(y!=1)
    address=LCD_LINE_TWO;
  else
    address=0;
  address+=x-1;
  lcdSendByte(0,0x80|address);
}

//Envia um caractere ao LCD
//\f->Limpa display, cursor no comeco da primeira linha
//\n->Move para o comeco da segunda linha
//\b->Backspace
void lcdPutc(char c){
  switch(c){
    case '\f':
      lcdSendByte(0,1);
      delay_ms(2);
    break;
    case '\n' :
```

```
      lcdGoToXY(1,2);
    break;
    case '\b':
      lcdSendByte(0,0x10);
      lcdSendByte(1,0x20);
    break;
    default:
      lcdSendByte(1,c);
    break;
  }//endswitch
}//end lcdPutc

//Obtem o caractere da posicao (x,y) no display
char lcdGetc(byte x, byte y){
  char value;

  lcdGoToXY(x,y);
  lcd.rs=1;
  value=lcdReadByte();
  lcd.rs=0;

  return(value);
}

//Funcao que detecta que algum botao foi pressionado
int waitForBtn(){
  int temp;

  do{
    temp=btn;
  }while((temp&0x0F)==0x0F);
  while((btn&0x0F)!=0x0F);

  return((temp&0x0F));
}//end waitForBtn


//-------------------------------------//
//------------MOVIMENTACAO-------------//
//-------------------------------------//
//Rotina para habilitar ou desabilitar o acionamento do motor
void motorEnable(short enable){
  if(enable==ON)
    output_high(MT_ENABLE);
  else
    output_low(MT_ENABLE);
}

//Rotina para acionar a roda esquerda
void motorLeft(unsigned long step, short dir){
  //espera o motor completar o numero de passos do comando anterior
  while(mtl_step_target!=0);
  //sentido de movimento para a roda esquerda do robo
  if(dir==FWD)
    output_low(DIR_LEFT);//motor esquerdo gira no sentido CCW
  else//if(dir==BWD)
    output_high(DIR_LEFT);//motor esquerdo gira no sentido CW
    mtl_step_target=step;//numero de passos a ser dado pelo motor
  }
```

```
//Rotina para acionar a roda direita
void motorRight(unsigned long step, short dir){
  //espera o motor completar o numero de passos do comando anterior
  while(mtr_step_target!=0);
  //sentido de movimento para a roda direita do robo
  if(dir==FWD)
    output_high(DIR_RIGHT);//motor direito gira no sentido CW
  else//if(dir==BWD)
    output_low(DIR_RIGHT);//motor direito gira no sentido CCW
  mtr_step_target=step;//numero de passos a ser dado pelo motor
}

//Rotina para movimentar o robo para frente em 'dist' mm
void moveFwd(unsigned long dist){
  unsigned long step;

  //1 volta = 200 passos = 161,16mm; 200/(161,16)=~1,24=31/25
  step=(dist*31)/25;//'dist' nao deve ser maior que 52851 para ocorrer
  //overflow em 'step'
  motorLeft(step, FWD);
  motorRight(step, FWD);
}

//Rotina para movimentar o robo para tras em 'dist' mm
void moveBwd(unsigned long dist){
  unsigned long step;

  //1 volta = 200 passos = 161,16mm; 200/(161,16)=~1,24=31/25
  step=(dist*31)/25;//'dist' nao deve ser maior que 52851 para ocorrer
  //overflow em 'step'
  motorLeft(step, BWD);
  motorRight(step, BWD);
}

//Rotina para girar o robo em torno do proprio eixo no sentido
//anti-horario de 'degree' graus
void turnCcw(unsigned long degree){
  unsigned long step;

  //Para girar de 360 graus em torno do proprio eixo, cada roda deve
  //percorrer um circulo de
  //diametro 'd', onde 'd' eh distancia entre as rodas (d=82,3mm),
  //portanto
  //step = degree*[(pi*d)*(31/25)]/360=~(320/360)*degree=(8/9)*degree
  step=(8*degree)/9;
  motorLeft(step, BWD);
  motorRight(step, FWD);
}

//Rotina para gira o robo em torno do proprio eixo no sentido horario
//de 'degree' graus
void turnCw(unsigned long degree){
  unsigned long step;

  //Para girar de 360 graus em torno do proprio eixo, cada roda deve
  //percorrer um circulo de
  //diametro 'd', onde 'd' eh distancia entre as rodas (d=82,3mm),
  //portanto
  //step = degree*[(pi*d)*(31/25)]/360=~(320/360)*degree=(8/9)*degree
  step=(8*degree)/9;
```

```
  motorLeft(step, FWD);
  motorRight(step, BWD);
}

//Faz com que o robo pare imediatamente
void stop(){
  mtl_speed=0;
  mtr_speed=0;
  mtl_step_current=0;
  mtr_step_current=0;
  mtl_step_target=0;
  mtr_step_target=0;
  mtl_counter=0;
  mtr_counter=0;
}

//Verifica se o robo esta em movimento
short isItMoving(){
  if((mtl_step_target!=0)||(mtr_step_target!=0))
  return(TRUE);//robo em movimento
  else
  return(FALSE);//robo parado
}

//Gera o proximo valor de mtx_speed
unsigned int nextSpeed(unsigned long remaining, unsigned int speed){
  //Controla a velocidade para (des)acelerar o motor
  if(remaining>(long)speed)
    speed++;
  else
    speed--;
  //Limites de velocidade
  if(speed>SPEED_MAX)
    speed=SPEED_MAX;
  if(speed<0)
    speed=0;
  return(speed);
}

//escreve os valores das constantes para aceleracao do motor
void setupAcceleration(){
  write_eeprom(0,0xFF);//cte p/ vel=0
  write_eeprom(1,0x9E);//cte p/ vel=1, etc.
  write_eeprom(2,0x41);
  write_eeprom(3,0x32);
  write_eeprom(4,0x2A);
  write_eeprom(5,0x25);
  write_eeprom(6,0x22);
  write_eeprom(7,0x1F);
  write_eeprom(8,0x1D);
  write_eeprom(9,0x1B);
  write_eeprom(10,0x1A);
  write_eeprom(11,0x18);
  write_eeprom(12,0x17);
  write_eeprom(13,0x16);
}


//--------------------------------------//
//------------SENSORIAMENTO------------//
```

```
//-------------------------------------//
//Inicializacao do modulo de sensoriamento
void setupSensor(){
  setup_port_a(A_ANALOG);//pinos AN0, AN1, AN2, AN3, AN4 analogicos;
  //Vref+ = Vdd, Vref- = Vss
  setup_adc(ADC_CLOCK_DIV_32);//Tad = 32*Tosc = 1,6us (Tad = tempo de
  //conversao de cada bit)
}

//Le o valor gerado pelo sensor de distancia 'sens_id'
unsigned int getSensor(unsigned int sens_id){
  unsigned int value;

  if(sens_id<=4){
    set_adc_channel(sens_id);// sens_id: canal de entrada
    //analogica->AN0,...,AN4
    delay_us(50);//tempo de conversao AD
    value=read_ADC()>>2;//read_ADC retorna um valor de 10 bit;
    //value->8 bits
    return(value);
  }
  else//sens_id invalido
    return(0xff);
}

//Obtem a media de 8 leituras do sensor de distancia 'sens_id'
unsigned int getSensorMean(unsigned int sens_id){
  int i;
  long sensor;

  sensor=0;
  for(i=1; i<=8; i++)//somatorio de 8 leituras do sensor
    sensor=sensor+(long)getSensor(sens_id);
  sensor=sensor>>3;//equivale a dividir o somatorio por 8

  return((unsigned int)sensor);
}


//-------------------------------------//
//------------APLICATIVOS------------//
//-------------------------------------//
//O robo explora o ambiente desviando de obstaculos
void explore(){
  short exp_canceled;
  unsigned int i, rand;
  //sensores frontal, traseiro, lateral direito, e lateral esquerdo,
  respectivamente
  long sensF, sensB, sensR, sensL;

  printf(lcdPutc,"\fExplor- \n-ing... ");
  delay_ms(1000);
  printf(lcdPutc,"\fPress \nCancel ");
  delay_ms(1000);
  printf(lcdPutc,"\fto abort\nmission ");
  delay_ms(1000);
  motorEnable(ON);
  exp_canceled=FALSE;
EXP_AGAIN:
  if(exp_canceled==FALSE){
```

```
    sensF=getSensorMean(SENS_FRONT);
    sensB=getSensorMean(SENS_BACK);
    sensR=getSensorMean(SENS_RIGHT);
    sensL=getSensorMean(SENS_LEFT);
    //Imprime no display os valores lidos dos sensores
    printf(lcdPutc, "\f %02x %02x\n %02x %02x ", sensF, sensB,
    sensR, sensL);
    //Faz 20 tentativas para verificar se ha caminho livre
    for(i=1; i<=20; i++){
      //Gera um numero "aleatorio"
      rand=(unsigned int)(get_timer1()&0x00FF);
      //Move para frente
      if(rand<=63){
        if(sensF<=LIMIT_DIST)//o sensor gera um valor inversamente
        //proporcional a distancia
          goto EXP_FWD;
      }//endif FWD
      //Move para tras
      else if(rand<=127){
        if(sensB<=LIMIT_DIST)
          goto EXP_BWD;
      }//endif BWD
      //Move para a direita
      else if(rand<=191){
        if(sensR<=LIMIT_DIST)
          goto EXP_RIGHT;
      }//endif RIGHT
      //Move para a esquerda
      else{
        if(sensL<=LIMIT_DIST)
          goto EXP_LEFT;
      }//endelse LEFT
    }//endfor
    //O robo detectou que esta cercado de obstaculos, e para!
    stop();
    delay_ms(500);
    printf(lcdPutc, "\f*JERRY**\n*LOCKED*");
    delay_ms(1000);
    exp_canceled=!((short)(btn&0x01));//verifica se o botao CANCEL foi
    //pressionado
    goto EXP_AGAIN;
EXP_FWD:
    stop();
    moveFwd(5000);//move 5000mm=5m
    goto EXP_OBSTACLE;
EXP_BWD:
    stop();
    turnCcw(180);//gira 180 graus
    moveFwd(5000);//move 5000mm=5m
    goto EXP_OBSTACLE;
EXP_LEFT:
    stop();
    turnCcw(90);//gira (90 graus) para a esquerda
    moveFwd(5000);//move 5000mm=5m
    goto EXP_OBSTACLE;
EXP_RIGHT:
    stop();
    turnCw(90);//gira 90 graus para a direita
    moveFwd(5000);//move 5000mm=5m
    goto EXP_OBSTACLE;
```

```
EXP_OBSTACLE:
    //Verifica obstaculos na direcao do movimento ou se o robo
    //terminou o trajeto ou se o botao CANCEL foi pressionado
    do{
      sensF=getSensorMean(SENS_FRONT);
      exp_canceled=!((short)(btn&0x01));
    }while((sensF<=LIMIT_DIST)&&(isItMoving())&&(!exp_canceled));
    goto EXP_AGAIN;
  }//endif exp_canceled

  motorEnable(OFF);
  printf(lcdPutc, "\fMISSION*\n*ABORTED");
  delay_ms(2000);
}//end explore
```