Felipe Augusto Weilemann Belo

Controle de Velocidade de um Sistema Robótico Rádio-Controlado Sentinela

Projeto de Fim de Curso

Projeto de Fim de Curso apresentado como requisito parcial para obtenção do título de Bacharel pelo Programa de Graduação em Engenharia de Controle e Automação da PUC-Rio.

Orientador: Marco Antônio Meggiolaro

PUC-Rio, novembro de 2004

À minha mãe e meu orientador Marco Antonio Meggiolaro.

Resumo

Felipe Augusto Weilemann Belo. **Controle de Velocidade de um Sistema Robótico Rádio-Controlado Sentinela.** PUC-Rio, 2004. Projeto de Fim de Curso - Engenharia de Controle e Automação, Pontifícia Universidade Católica do Rio de Janeiro.

Foi construído um robô com onde foi feito o estudo do controle de velocidade em motores DC. Este robô foi pensado para uso em missões de reconhecimento, monitoramento e resgate. Trata-se de um rover do tipo differential-drive constituído de duas rodas e ponto de apoio. Seu controle foi feito por rádio-controle com possibilidade de controlar a velocidade independente das rodas ou controlar a velocidade e curva. O controle de velocidade foi feito utilizando-se um microcontrolador e a velocidade de cada roda foi obtida através de encoders incrementais.

Palavras-chave

Robôs Móveis; Teoria de Controle; Microcontroladores; Controle de Velocidade; Motores DC; Rádio-Controle;

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

1.1.
Motivação:

Existem diversas tarefas não apropriadas para o ser humano, seja por limitações físicas, periculosidade ou baixa eficiência . Podemos ver na industria diversas tarefas, antes inviáveis, sendo hoje executadas por robôs especialistas. Um exemplo comum é o uso de manipuladores robóticos na industria automotiva, utilizados na fabricação e montagem de automóveis, estes suportam altas temperaturas, toneladas de pesos e ainda trabalham em alta velocidade, tarefas impraticáveis para o ser humano.

O uso de robôs na automação industrial, exploração espacial, em locais de difícil acesso e em tarefas penosas e perigosas tem se difundido muito nos últimos anos, principalmente devido à queda de custos associados à produção dos mesmos. O crescimento e popularização dos microcontroladores em chips únicos associados à redução de tamanho dos componentes eletrônicos e enorme queda de preços dos mesmos são fatores críticos na expansão da robótica vista nos últimos anos. Neste contexto, aqueles que não estiverem capacitados a dominar e desenvolver dentro dos muitos campos associados à automação e robótica estarão defasados tecnologicamente no mercado global.

A robótica é um campo multidisciplinar. O desenvolvimento de um robô requer conhecimentos de diversas áreas, tais como as engenharias mecânica e elétrica, ciência da computação, inteligência artificial e sistemas de controle. Os desafios encontrados no desenvolvimento de um robô agregam ao profissional da área de controle e automação habilidades básicas nestas diversas áreas, possibilitando seu crescimento como profissional generalista e de ampla visão.

Sistemas de controle são utilizados em diversas áreas, do controle de processos industriais (controle de pressão, vazão, temperatura, umidade, entre

outros) ao uso em veículos espaciais, associados à robótica, os sistemas de controle tem papel vital no funcionamento automático e eficiente de robôs. A execução de tarefas simples como posicionamento, controle de velocidade e controle de força necessita do amparo de sistemas de controle para que se obtenham bons resultados.

O robô aqui apresentado pretende abordar o uso de diversas tecnologias freqüentemente utilizadas em robótica e propor algumas possibilidades de controle de velocidade em motores DC. Acoplado de uma câmera, pode ser tele-operado e aplicado em missões de monitoramento, reconhecimento e segurança.

1.2. O robô sentinela

O robô desenvolvido neste projeto (Figura 1-1) foi projetado para uso em terrenos desestruturados, possuindo a característica de poder trabalhar de cabeça para baixo. Além disso, é um robô versátil podendo ser aprimorado para experiências de navegação inercial, inteligência artificial, sensoriamento, reconhecimento de áreas e planejamento de trajetórias. Finalmente, é um robô de design simples, fácil construção, e ainda de baixo custo.

O robô é chamado de Sentinela por ter sido pensado para uso em missões de reconhecimento, monitoramento e resgate. O Sentinela é um *rover* constituído de 2 rodas acopladas às extremidades do seu corpo, possuindo uma calda com função de apoio ao chão. Seu controle é feito por rádio-controle com possibilidade de controle de velocidade independente das rodas ou controle de velocidade e curva. O controle de velocidade é feito por um controlador discreto do tipo PID (proporcional, integral, derivativo) e a velocidade de cada roda é obtida através de encoders incrementais. Sua estrutura permite o acoplamento de uma câmera de vídeo com transmissor para que ele possa ser tele-operado, porém a instalação da câmera não foi efetuada.

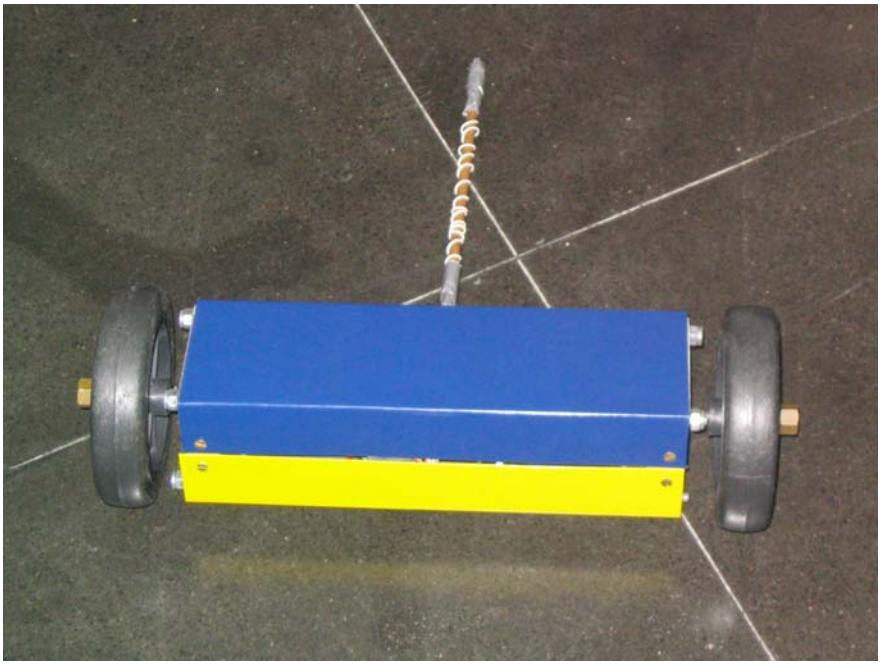

Figura 1-1: Robô Sentinela montado

O sentinela foi inspirado no robô Lacraia (Figura 1-2) desenvolvido na Puc-Rio para ser utilizado durante a "3ª Guerra de Robôs", evento realizado em outubro de 2003 na Universidade Federal de Itajubá. O Lacraia é um robô de estrutura similar ao Sentinela, porém de porte superior, e também é controlado por rádio controle. A sua estrutura possibilita completa liberdade de movimentos em qualquer direção ou orientação (inclusive de cabeça para baixo), pontos fortes para a agilidade do robô durante o evento ao qual foi proposto. O Lacraia porém, não possuía controle avançado de velocidade, seu controle era baseado em ligar e desligar os seus motores. O Sentinela surgiu da idéia de se fazer um robô em menor escala com a agilidade do Lacraia, que pudesse ser tele-operado (tornando possível seu uso em diversas aplicações), e que fosse viável de ser usado, caso aprimorado, em experiências de navegação inercial, inteligência artificial, sensoriamento, reconhecimento de áreas e planejamento de trajetórias. O Sentinela ainda possui o diferencial de trabalhar com PWM (*Pulse Width Modulation*) e *encoders* para se ter controle de velocidade visando mantê-lo em linha reta em terrenos planos e possibilitando um controle mais refinado dos movimentos do robô.

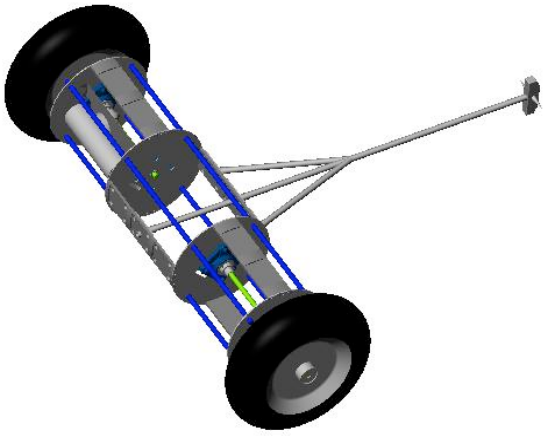

Figura 1-2: Projeto do robô Lacraia montado.

Algumas das possíveis aplicações para o robô sentinela acoplado de uma câmera e tele-operado são em situações que envolvem perigo como:

- Segurança e monitoramento de áreas;
- Busca de minas ou materiais químicos (acoplado dos sensores necessários);
- Reconhecimento de áreas perigosas ou de difícil acesso. Como exemplo têm-se o uso pela polícia em missões de resgate de reféns e o uso por bombeiros em busca de pessoas em um incêndio;

Como para se desenvolver qualquer robô, muitos conhecimentos foram abordados neste projeto, estando eles situados dentro das engenharias elétrica, mecânica e ciência da computação.

Da engenharia mecânica temos:

- Projeto mecânico do robô;
- Desenho técnico do robô feito em *Solid Works*;
- Confecção das peças em alumínio;
- Montagem do robô;

Da engenharia elétrica temos:

- Uso do microcontrolador PIC;
- Interface entre o microcontrolador e o motor através de ponte H;
- Variação da potência aplicada aos motores através de PWM;
- Uso de *encoders* incrementais;
- Uso de rádio transmissor e receptor utilizado em aeromodelismo;
- Controle em malha aberta;
- Controle em malha fechada PID independente;

- Controle em malha fechada PID conjunto;

Da ciência da computação temos:

- Programação de microcontrolador PIC em C;
- Processamento do sinal do rádio controle;
- Leitura dos encoders;
- Geração do sinal de PWM;
- Implementação das malhas de controle discretizadas;

2
Apresentação teórica do problema:

2.1.
Controle de velocidade

Um dos objetivos do projeto é o estudo de diferentes malhas de controle para controlar a velocidade dos motores. O diagrama geral de controle pode ser visto abaixo (Figura 2-1):

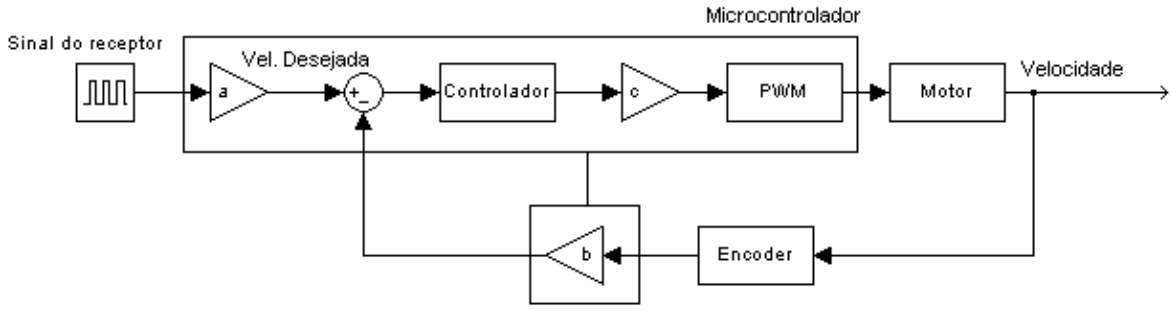

Figura 2-1: Diagrama geral de controle

O processamento do sinal do receptor, a aquisição da velocidade do motor e a interface entre microcontrolador e motores serão detalhados adiante.

Neste trabalho serão considerados 3 tipos de controle:

- Controle em malha aberta;
- Controle em malha fechada independente para cada motor;
- Controle em malha fechada para os dois motores;

Para os controles em malha fechada será utilizado controle tipo PID (proporcional, integral, derivativo).

2.1.1.
Controle em malha aberta

O controle em malha aberta possui o seguinte diagrama (Figura 2-2):

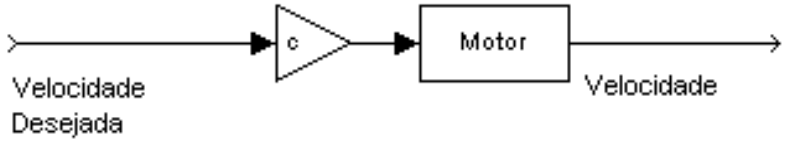

Figura 2-2: Controle em malha aberta

No controle em malha aberta, não existe feedback de velocidade do motor, portanto, o robô não sabe o quão rápido as rodas estão, ou, se o robô está fazendo curva, não há ajuste quando as velocidades forem diferentes das velocidades desejadas.

Para este tipo de controle, o microcontrolador simplesmente processa o sinal do receptor e gera uma saída para o motor relativa a velocidade desejada. Dependendo do terreno, obstáculos, atritos ou carga, a velocidade gerada não será necessariamente a velocidade desejada.

2.1.2.
Controle em malha fechada independente

No controle independente para cada motor, entramos com a velocidade desejada para cada um separadamente e eles serão tratados com controlador próprio.

Temos abaixo (Figura 2-3) o diagrama de blocos para este tipo de controle:

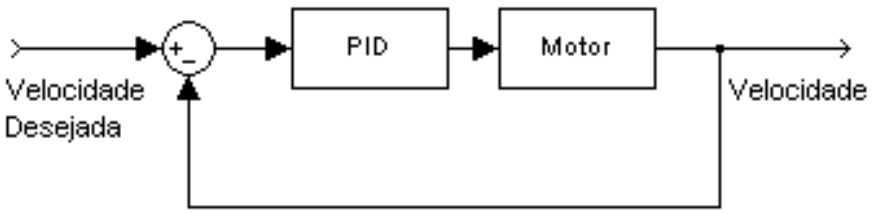

Figura 2-3: Controle em malha fechada independente

A idéia básica deste controle é verificar a diferença entre a velocidade desejada e a velocidade real (chamada de erro) e passando pelo controle PID calcular a nova saída para o motor.

A desvantagem do controle independente é que quando desejamos a mesma velocidade para os dois motores e fatores externos causam um erro na velocidade de um dos motores, este erro não é visto pelo outro motor que não poderá se corrigir para que os dois fiquem com a mesma velocidade. Este problema não acontece com o próximo controlador.

2.1.3.
Controle em malha fechada para os dois motores

O controle em malha fechada para dois motores (Figura 2-4) permite sincronia entre as duas velocidades garantindo com que o robô ande em linha reta. A sincronia é dada pela linha central do diagrama acima.

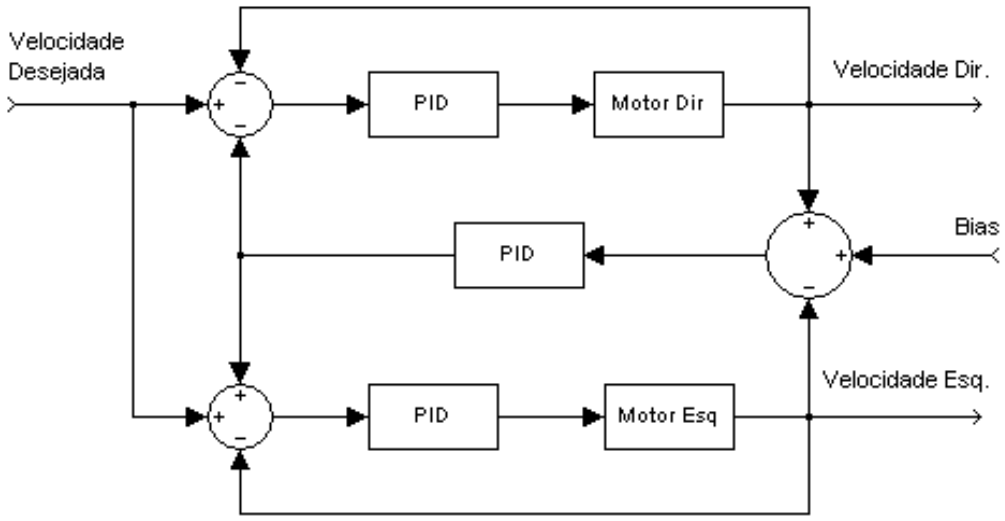

Figura 2-4: Controle em malha fechada para dois motores

O controlador no centro verifica a velocidade dos dois motores e os compara. A diferença entre as velocidades somada ao *Bias* passa pelo PID central, o resultado é somado a malha de controle do motor direito e subtraído da malha de controle para o motor esquerdo. Desta maneira, quando a velocidade do motor direito for maior que o esquerdo, será subtraído um erro da velocidade

desejada direita e somado um erro à velocidade desejada esquerda. Quando a velocidade do motor esquerdo for maior que a do motor direito, obtemos o resultado inverso.

O *Bias* tem função para permitir que o robô faça curva. Quando seu valor for positivo, o robô fará curva para a esquerda, e quando negativo para a direita. Isto acontece pois o feedback do erro entre velocidades será a soma do Bias à diferença entre as velocidades, portanto, quando a diferença entre velocidades for igual ao *Bias*, este erro será nulo.

Esta malha de controle é apresentada em [1].

2.2. Controle PID

2.2.1. Discretização do controle PID

A ação de controle PID nos controladores analógicos é dada por:

$$c(t) = K\left[e(t) + \frac{1}{Ti}\int_0^t e(t)dt + Td\,\frac{de(t)}{dt}\right] \qquad 1$$

E pode ser representada em Laplace como:

$$c(s) = K\left[1 + \frac{1}{Ti \cdot s} + Td \cdot s\right] \qquad 2$$

Para discretizarmos, vamos aproximar a integral pelo somatório trapezoidal e a derivada pela diferença entre dois pontos.

$$c(kT) = K\left\{\begin{array}{l} e(kT) + \dfrac{T}{Ti}\left[\begin{array}{l} \dfrac{e(0)+e(T)}{2} + \\ \dfrac{e(T)+e(2T)}{2} + \\ \cdots + \\ \dfrac{e((k-1)T)+e(kT)}{2} \end{array}\right] \\ +Td\dfrac{e(kT)-e((k-1)T)}{T} \end{array}\right\} \quad 3$$

$$c(kT) = K\left\{\begin{array}{l} e(kT) + \dfrac{T}{Ti}\displaystyle\sum_{h=1}^{k}\dfrac{e((h-1)T)+e(hT))}{2} \\ +Td\dfrac{e(kT)-e((k-1)T)}{T} \end{array}\right\} \quad 4$$

Se define:

$$\frac{e((h-1)T)+e(hT))}{2} = f(hT), f(0) = 0 \quad 5$$

E portanto:

$$c(kT) = K\left\{e(kT) + \frac{T}{Ti}\sum_{h=1}^{k} f(ht) + Td\frac{e(kT)-e((k-1)T)}{T}\right\} \quad 6$$

Vamos agora discretizar por partes:

$$Z\left[\sum_{h=1}^{k}\frac{e((h-1)T)+e(hT))}{2}\right] = Z\left[\sum_{h=1}^{k} f(hT)\right] =$$

$$\frac{1}{1-z^{-1}}\left[F(z)-f(0)\right] \quad 7$$

$$= \frac{1}{1-z^{-1}}F(z)$$

Pode-se notar que:

$$F(z) = Z\left[f(hT)\right] = \frac{1+z^{-1}}{2}E(z) \quad 8$$

A transformada de c(kT) fica:

$$C(z) = K\left[1 + \frac{T}{2Ti}\frac{1+z^{-1}}{1-z^{-1}} + \frac{Td}{T}(1-z^{-1})\right]E(z) \quad 9$$

Podemos reescrever como:

$$C(z) = K\left[1 - \frac{T}{2Ti} + \frac{T}{Ti}\frac{1}{1-z^{-1}} + \frac{Td}{T}\left(1-z^{-1}\right)\right]E(z)$$
$$= \left[Kp + Ki\frac{1}{1-z^{-1}} + Kd\left(1-z^{-1}\right)\right]E(z) \quad 10$$

Onde:

$$Kp = K - \frac{KT}{2Ti} = K - \frac{Ki}{2} \quad 11$$

É a constante Proporcional.

$$Ki = \frac{KT}{Ti} \quad 12$$

É a constante Integral.

$$Kd = \frac{KTd}{T} \quad 13$$

É a constante Derivativa.

Reescreve-se como:

$$(z^{-1}-1)C(z) = ((Kp + 2Kd)z^{-1} - Kd\cdot z^{-2} - Kp - Ki - Kd)E(z) \quad 14$$

Fazendo a inversa temos:

$$c(n) = Kd\cdot e(n-2) - (Kp + 2Kd)e(n-1)$$
$$+(Kp + Kd + Ki)e(n) \quad 15$$
$$+c(n-1)$$

2.2.2.
Discretização do controlador PI

Uma simplificação do controle PID é o controle do tipo PI (proporcional, integral), que pode ser discretizado como:

$$c(n) = Kp\cdot e(n) + Ki\cdot c(n-1) \quad 16$$

Onde:

- c(n) = Saída no momento n;
- e(n) = Erro no momento n;

- Kp = Constante proporcional;
- Ki = Constante integral;

Esta discretização para o controle PI é utilizada em [1].

2.3. Rádio-Controle

O microcontrolador receberá um sinal digital do rádio-controle que será utilizado para se definir a velocidade de cada motor. Serão dois sinais recebidos, um para cada motor.

O rádio-controle utilizado foi o *6X computer radio super* da *Futaba*. Este é um rádio-controle de 6 canais utilizado em aeromodelos. O receptor usado foi o *FP-R138DP* também da *Futaba*.

O sinal de cada canal do receptor possui período entre 18ms e 25ms, com o sinal alto variando entre 1ms e 2ms.

2.4. Interface entre microcontrolador e motores

O microcontrolador não é capaz de acessar os motores diretamente por não poder suprir a corrente necessária para o funcionamento dos mesmos. Portanto, é utilizado um circuito como interface entre os sinais de controle e a tensão fornecida para o motor por uma bateria externa.

O circuito de interface pode ser implementado por diversas tecnologias, tais como relés, transistores bipolares, transistores de potência mosfets ou circuitos integrados conhecidos como “motor-driver-power ICs”. Porém, independente da tecnologia implementada, a topologia básica dos circuitos costuma manter-se a mesma, quatro chaves conectadas na topologia de um H, onde os terminais do motor formam a barra horizontal do H. Esta topologia é conhecida como ponte H.

2.4.1. Ponte H

A topologia de uma ponte H pode ser vista na figura abaixo (Figura 2-5).

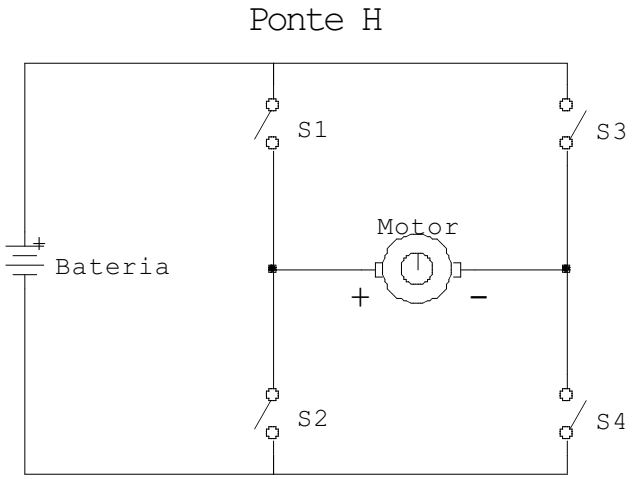

Figura 2-5: Topologia de uma ponte H

Na ponte H, as chaves são implementadas por relés ou transistores, a potência é fornecida pela bateria e os sinais de controle pelo microcontrolador.

Na ponte H, as chaves são abertas ou fechadas de maneira a colocar a voltagem em polaridade que faça a corrente circular pelo motor em dada direção. Ou, colocar a voltagem em polaridade inversa fazendo a corrente circular na direção oposta e o motor girar em rotação reversa.

Na figura apresentada, vamos ter a corrente circulando da esquerda para a direita quando as chaves S1 e S4 estiverem fechadas e S2 e S3 abertas. A corrente estará circulando da direita para a esquerda quando S2 e S3 estiverem fechadas e S1 e S4 abertas.

Para controlar a velocidade do motor, abrimos e fechamos as chaves em diferentes proporções de tempo utilizando a técnica PWM (Pulse Width Modulation).

2.4.2. PWM

PWM significa modulação da largura de pulso e é uma técnica muito utilizada quando se deseja variar digitalmente uma potência aplicada. Duas aplicações muito comuns do uso de PWM são o controle de velocidade de um motor DC e a dimmerização digital de lâmpadas.

No controle de velocidade feito, o sinal PWM será utilizado para chavear a ponte H. O sinal PWM está ilustrado na figura abaixo (Figura 2-6):

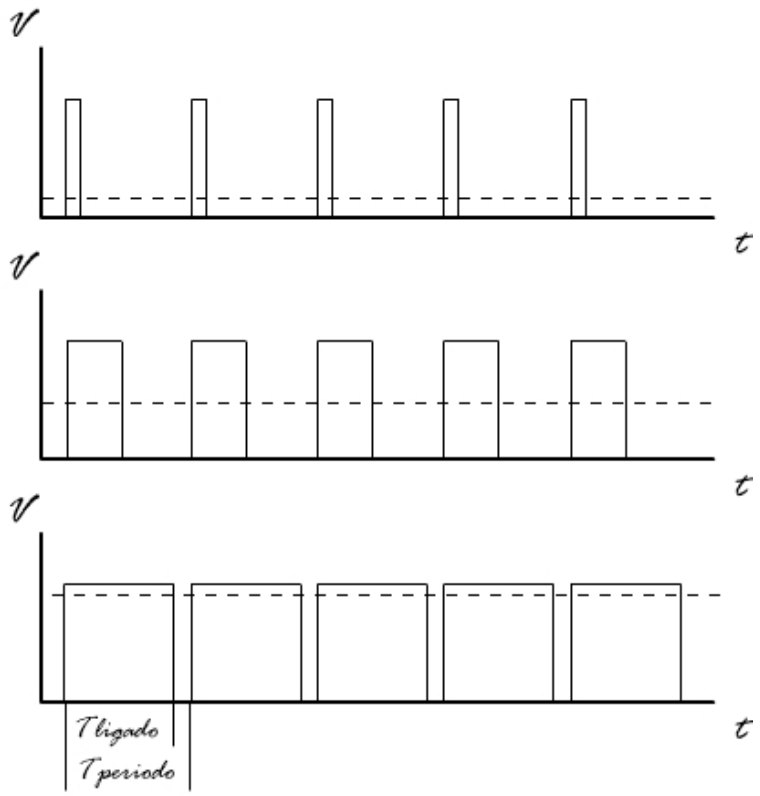

Figura 2-6: Sinal PWM

Como pode ser visto na figura, a técnica PWM constitui da modulação da largura de um pulso digital repetido em intervalos regulares. Quando este sinal chaveia uma potência, conseguimos controlar a potência média liberada digitalmente. No caso do motor DC, a velocidade do motor pode ser ajustada mudando-se a proporção $T_{ligado}/T_{período.}$

$$Pulse - Width - Ratio = \frac{Tligado}{Tperíodo} \qquad 17$$

Como já foi mencionado, a ponte H pode ser implementada através de diversas tecnologias, entre elas, relés. Porém, o uso de relés não é aconselhado com PWM pois os relés não conseguem chavear rapidamente, além de se esgotarem rapidamente quando intensamente chaveados. Para uso de PWM, transistores são mais aconselhados.

2.5.
Leitura de velocidade

Para ler a velocidade em cada motor, precisamos de um transdutor de velocidade. Para o robô Sentinela foi escolhido utilizar um encoder incremental, que é um transdutor de posição e pode ser utilizado para calcular a velocidade dos motores.

2.5.1.
Encoders

Encoders são transdutores de movimento, que medem rotação ou posição, muito utilizados em automação e robótica.

Existem dois tipos de encoders, encoders absolutos e encoders incrementais. O encoder absoluto tem como sinal de saída um código relativo à posição. Já o encoder incremental, tem como saída um trem de pulso que especifica o quanto a posição variou. O encoder utilizado será o encoder incremental.

Vamos ver como o encoder funciona. O encoder possui um disco perfurado entre transmissores e receptores IR. Quando o sinal de infravermelho do transmissor atravessa um furo do encoder e encontra o receptor, o receptor gera sinal 1, e, quando o receptor não recebe o sinal de infravermelho, ele gera sinal 0. Com o disco girando, a saída do receptor gera um trem de pulsos. Se trabalhamos com dois pares de transmissores e receptores, com seus sinais defasados, podemos além de obter um trem de pulso relativo ao movimento, saber a direção da rotação. Este tipo de encoder é chamado de incremental bidirecional.

Temos ilustrado na figura abaixo (Figura 2-7) o funcionamento do encoder incremental. Na figura só está apresentado um par transmissor-receptor, porém, a ilustração referente ao sinal apresenta o trem de pulso de 2 encoders defasados. Perceba que se o disco estiver em rotação gerando os sinais como na seta da esquerda para direita, quando o sinal um sobe, o sinal 2 está em 0. Agora, se o disco estiver girando no sentido contrário, o sinal será gerado como na seta da direita para esquerda, e quando o sinal 1 sobe, o sinal 2 está em 1. Desta maneira, conseguimos saber a direção da rotação do disco, e contanto o número de pulsos,

sabemos o quanto foi girado. Se também tivermos o valor do tempo, podemos calcular a velocidade de rotação, o que é feito dentro do microcontrolador para saber a velocidade de cada motor.

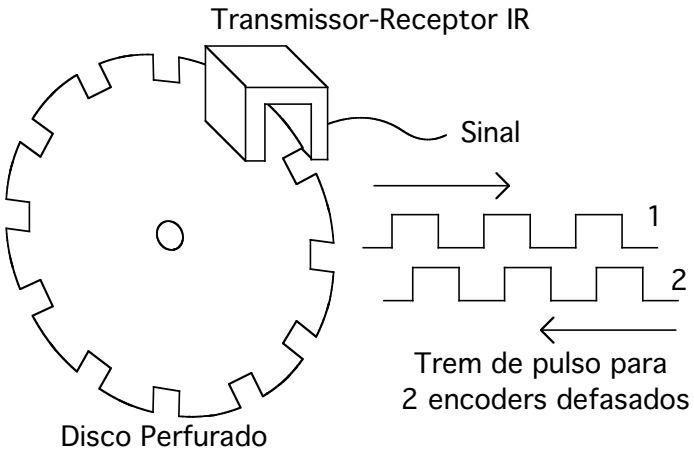

Figura 2-7: Transmissor-Receptor IR

Alguns encoders possuem ainda uma terceira saída, que gera um pulso quando o encoder atinge sua posição absoluta zero.

2.6. Microcontrolador PIC

Microcontroladores são componentes eletrônicos que combinam um microprocessador com outros periféricos. Enquanto a maioria dos microprocessadores possuem apenas CPU (Unidade de Processamento Central), ALU (Unidade Lógica Aritmética), linhas para dados, linhas para endereços e acesso de memória externa, os microcontroladores costumam possuir tudo isso e mais memória interna, periféricos para comunicação serial, timers, osciladores, portas de input ou output, PWM, entre outros.

Na construção do robô Sentinela, se utiliza um microcontrolador como cérebro. É ele que recebe e decodifica os sinais do receptor de rádio e dos encoders, faz os cálculos do controlador PID e gera os sinais PWM para acionamento dos robôs.

Os microcontroladores utilizados no estudo e implementação do robô sentinela foram todos microcontroladores tipo PIC fabricados pela Microchip. Foram escolhidos estes microcontroladores ao invés de outros disponíveis no

mercado devido à sua fácil disponibilidade, abrangente documentação, por serem os microcontroladores mais utilizados no Brasil e por possuírem determinados periféricos que substituiriam algumas soluções por software, como o PWM.

O PIC é normalmente programado em assembler, porém encontram-se compiladores em C para PIC. Pela linguagem C ser mais amplamente difundida e de mais fácil uso, optou-se por trabalhar com ela e não com a linguagem assembler.

3
Experiência e Procedimento Experimental:

3.1.
Projeto

O design de um *Rover* é de grande importância, tanto como fonte de locomoção, tanto como plataforma para seus sensores, eletrônica e equipamento experimental. Seis desenhos base são sugeridos (Figura 3-1 à Figura 3-6):

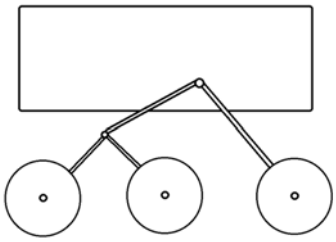

Figura 3-1: Rover de seis rodas, similar ao desenvolvido pelo Jet Propulsion Lab da Nasa, integrante das missões Spirit e Opportunity enviadas a Marte

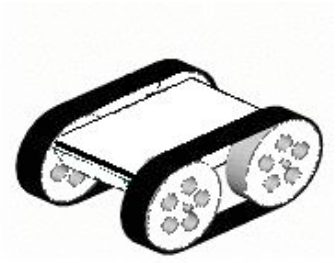

Figura 3-2: Rover com desenho similar a um tanque

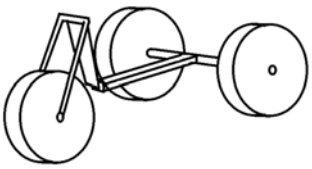

Figura 3-3: Rover simila a um tricíclo

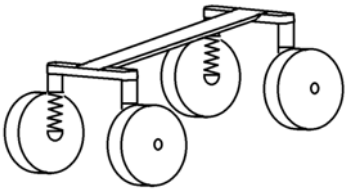

Figura 3-4: Rover de 4 rodas com tração e suspensão independentes

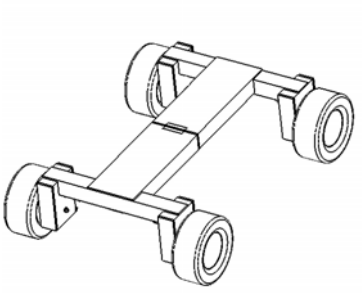

Figura 3-5: Rover de 4 rodas com centro articulado

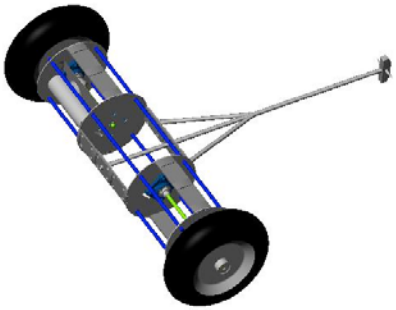

Figura 3-6: Rover de 2 rodas com ponto de apoio (calda)

Cada um dos desenhos propostos possui suas vantagens e desvantagens apresentadas na Tabela 3-1.

Tabela 3-1: Vantagens e desvantagens para diferentes possibilidades de construção

	Estabilidade	Construção	Robustez	Simplicidade	Aprimoramento	Desvio de obstáculos
1	Alta	Muito difícil	Pouca	Nenhuma	Possível	Excelente
2	Média	Mediana	Muita	Mediana	Difícil	Bom
3	Baixa	Fácil	Pouca	Muita	Difícil	Ruim
4	Média	Difícil	Mediana	Mediana	Difícil	Médio
5	Média	Mediana	Mediana	Mediana	Difícil	Médio
6	Média	Muito fácil	Muita	Muita	Possível	Bom

O robô Lacraia no qual o Sentinela fora inspirado, havia demonstrado grande agilidade e liberdade de movimentos. O desenho de *Rover* com duas rodas apresenta capacidade de navegação sem problemas de estabilidade e com facilidade de se comportar em ambientes inóspitos por funcionar de cabeça para baixo. Além disto, é um *Rover* de relativa fácil construção por não ser demasiado complexo e ser de aspecto modular. O aspecto modular é devido ao fato do seu corpo ser composto de 4 eixos nos quais são encaixados módulos separados por espaçadores. Desta maneira, conseguimos redefinir os módulos alterando o tamanho dos espaçadores ou até incluir novos módulos aumentando os eixos. Como visto, com pequenas alterações, podemos modificar este *Rover* para incluir novos acessórios ou modificar os existentes.

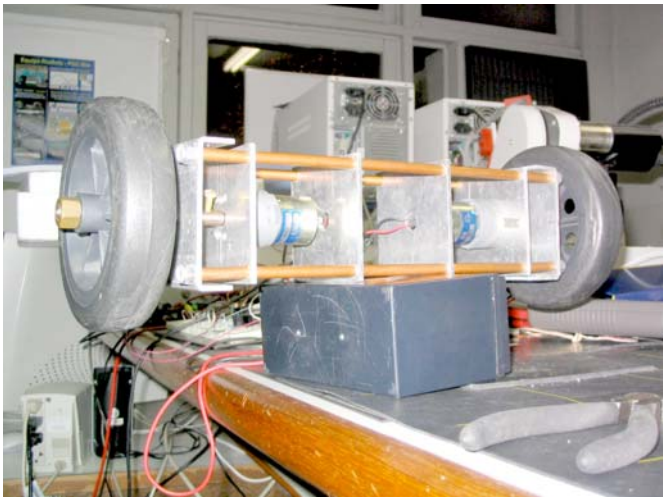

Figura 3-7: Sentinela aberto em processo de montagem

Em sua concepção final, o Sentinela ficou com cinco módulos, dois para os motores, dois para os *encoders* e um para a eletrônica. O receptor do sinal de rádio ficou no módulo da eletrônica e as baterias nos módulos dos motores.

O tamanho do robô foi escalado no menor tamanho possível de se comportar todas as partes do robô.

Dois projetos foram feitos para o robô. Inicialmente pretendia-se trabalhar com acrílico, buscando uma aparência mais profissional, usar motores *MicroMo*, tacômetros *MicroMo*, desenvolver as caixas de redução, usar rolamentos e ter as peças confeccionadas por um torneiro profissional. Devido a diversas dificuldades encontradas o projeto foi completamente modificado.

Confeccionar as peças através de um torneiro profissional se mostrou uma opção inviável pelos custos envolvidos em se fazer as peças em curto prazo.

O uso de acrílico é muito utilizado em robôs experimentais por ser um material leve, resistente, de boa aparência e certa facilidade de uso. Porém, seu uso implicava em um custo superior, dificuldade de aquisição do material em baixa quantidade e necessidade de se trabalhar com um torneiro profissional (aumentando o custo do projeto) devido a certa dificuldade em se trabalhar com o acrílico na confecção de determinadas peças.

Os motores *MicroMo* possuídos não possuíam caixa de redução, nem tacômetros ou *encoders*. A compra destes componentes da *MicroMo* implicariam em altos custos.

O desenvolvimento de uma caixa de redução foi estudado mas a idéia foi abandonada por alguns motivos. Não existem no mercado engrenagens avulsas para a confecção de uma caixa de redução em pequena escala. A confecção de engrenagens para uma redução específica é de altíssimo custo. Mesmo que se confeccionasse as engrenagens ou tentasse adaptar engrenagens sucateadas, a construção da caixa de redução teria de ser feita com enorme precisão.

Não foram encontrados no mercado mancais pequenos prontos para uso no robô. A confecção destes mancais teria de ser feita por um torneiro, inviabilizando a construção dos mesmos.

Não foram encontrados no mercado muitos tacômetros, sendo os mesmos além de caros, grandes, analógicos (fazendo-se necessária conversão analógico-digital) e difíceis de se adaptar ao robô exigindo uso de engrenagens. E ainda, o uso de tacômetros implicaria em perda de eficiência dos motores por eles trabalharem como freio elétrico.

No fim, optou-se por trabalhar com placas de alumínio e confeccionar todas as peças, usar motores comprados em sucata com redução, construir os *encoders* utilizando peças e componentes retirados de motores que já possuíam *encoders* e eliminou-se o uso de rolamentos, o que acabou prejudicando o alinhamento das rodas.

A eletrônica do robô cuida do funcionamento do mesmo. O diagrama abaixo (Figura 3-8) resume o funcionamento do robô:

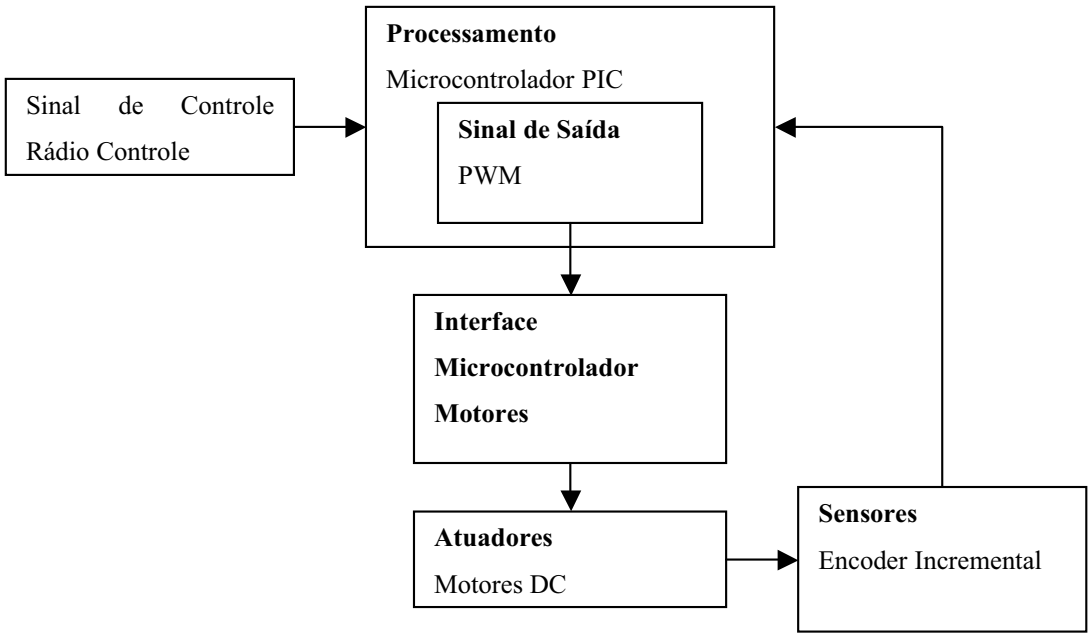

Figura 3-8: Funcionamento do robô

O sinal recebido pelo receptor do rádio controle *Futaba* é decodificado pelo microcontrolador PIC para extrairmos as informações de velocidade desejada para cada um dos motores. O sinal dos *encoders* também é processado pelo microcontrolador se saber a velocidade real dos motores. As informações relativas às velocidades desejadas e às velocidades reais são passadas à malha de controle, executada pelo microcontrolador, e então é gerado o sinal de PWM de cada motor. O microcontrolador, então, controla a ponte H, e emite para esta os sinais de PWM. A ponte H por fim, aciona os motores como desejado.

3.2. Projeto Mecânico

O Sentinela foi concebido em SolidWorks como pode ser visto abaixo (Figura 3-9 e Figura 3-10):

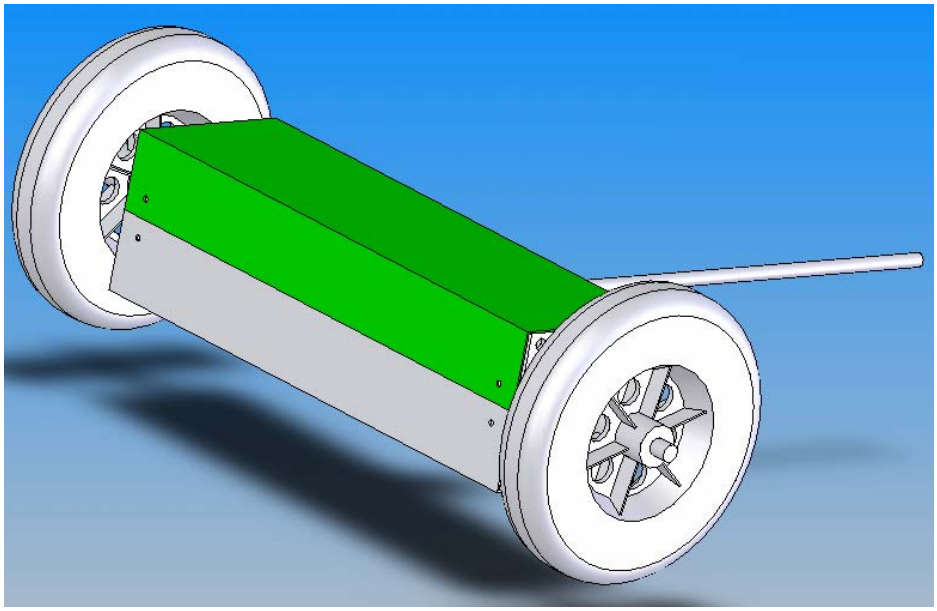

Figura 3-9: Robô Sentinela com capa

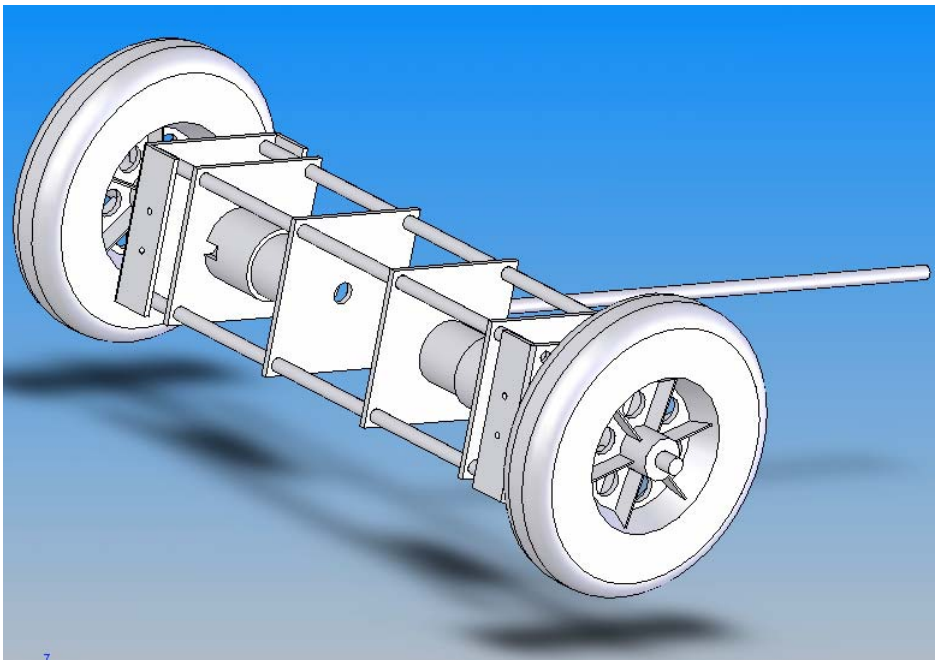

Figura 3-10: Robô Sentinela sem capa

Cada uma de suas peças foi projetada separadamente e por fim unidas para montar o Sentinela como visto. Cada uma de suas peças mais importantes será apresentada a seguir separadamente. Não são apresentados porcas e parafusos. Também não estão apresentados os tubos que separam as placas nem o eixo rosqueado que corre por dentro destes.

As peças podem ser vistas da Figura 3-11 à Figura 3-19.

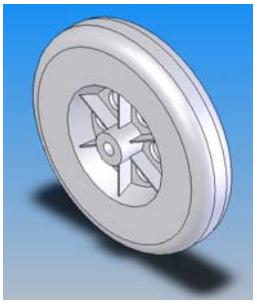

Figura 3-11: Rodas

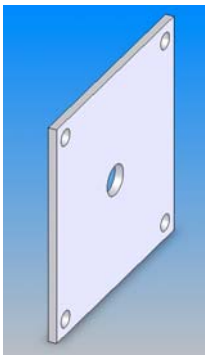

Figura 3-12: Placas Internas

Figura 3-13: Placas Externas

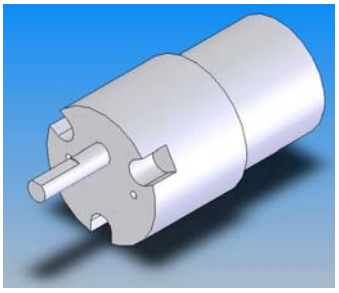

Figura 3-14: Motores

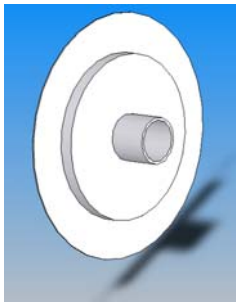

Figura 3-15: Enconders

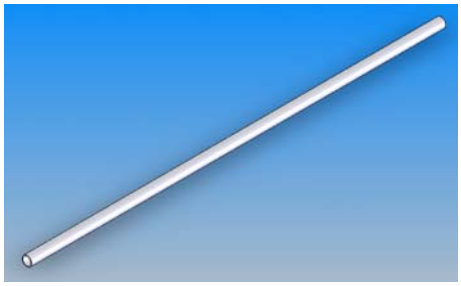

Figura 3-16: Rabos

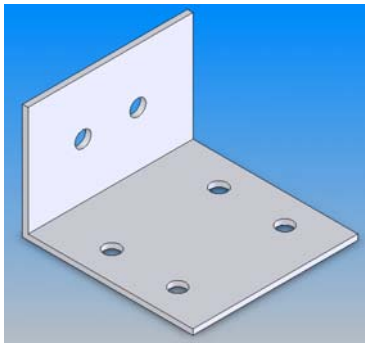

Figura 3-17: Suporte ao Rabo

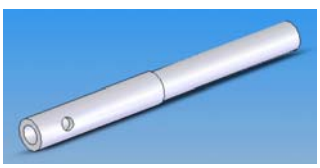

Figura 3-18: Eixos

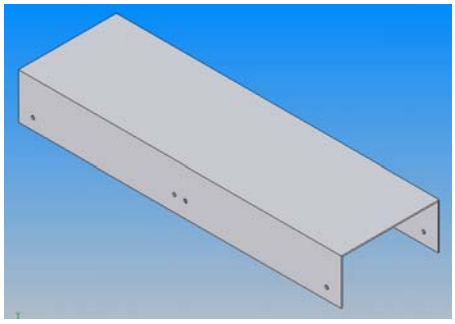

Figura 3-19: Capas

3.3. Circuito Elétrico

O circuito desenvolvido é apresentado no esquemático a seguir (Figura 3-20):

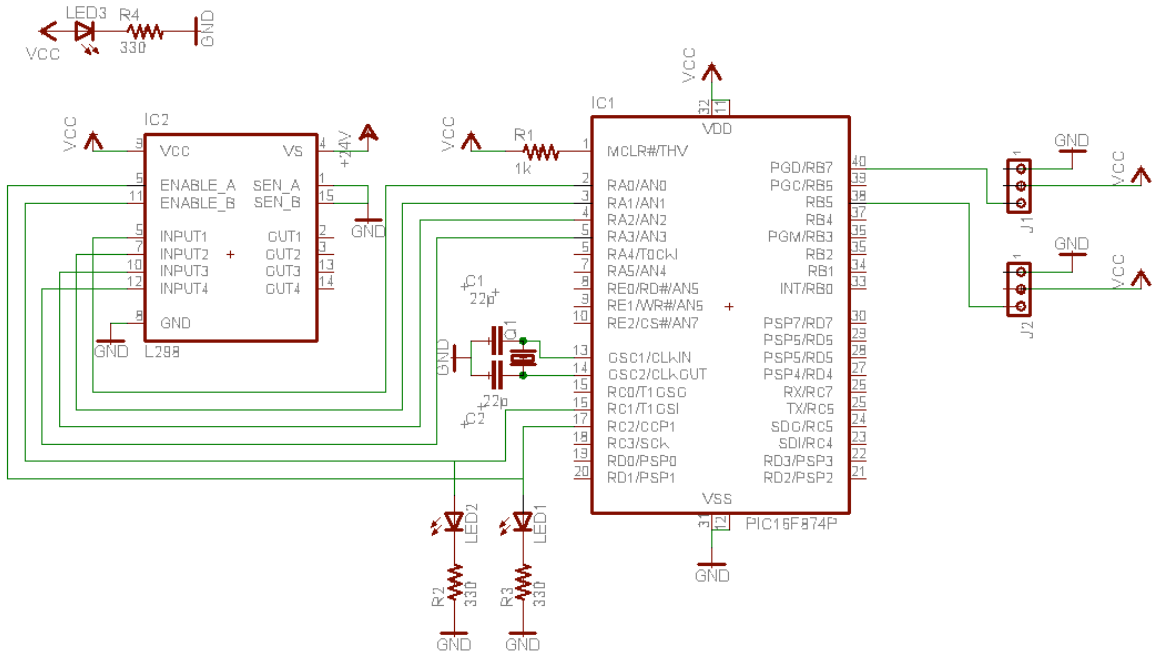

Figura 3-20: Esquemático do circuito elétrico

No circuito não estão apresentados os motores nem os encoders;

Os dois principais componentes são o microcontrolador PIC16F874 e o *Motor Driver* L298.

O integrado L298 é um componente que possui duas pontes H para alta conrrente e aceita níveis lógicos padrão TTL para controle. Há também duas

entradas utilizadas para habilitar ou desabilitar as pontes. O sinal de PWM gerado pelo microcontrolador PIC é utilizado para o controle do L298 e acionamento dos motores.

O diagrama de bloco do L298 pode ser visto abaixo (Figura 3-21):

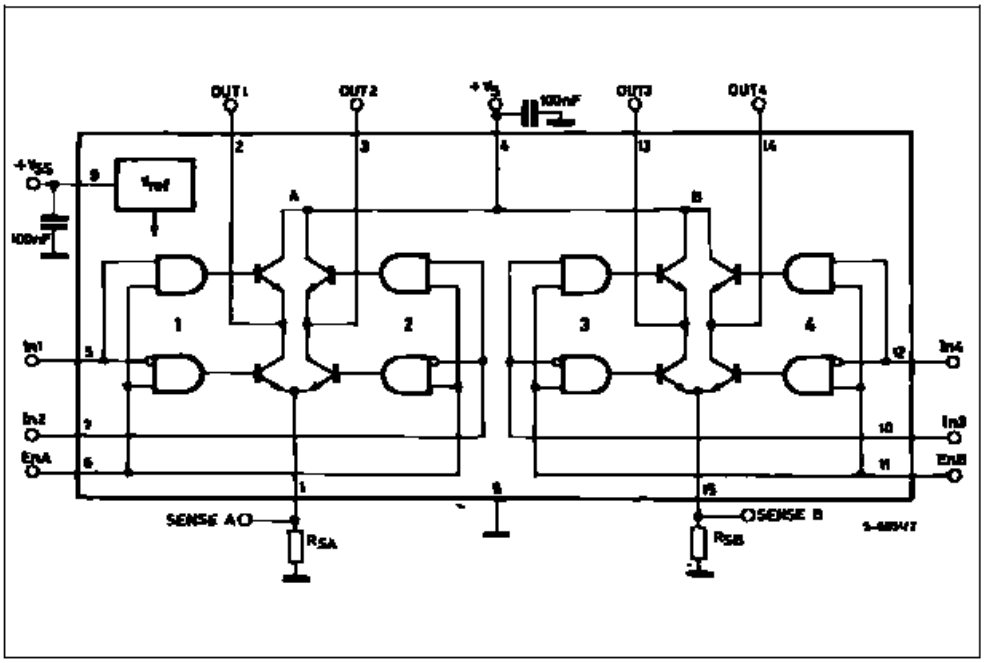

Figura 3-21: Diagrama de bloco do chip L298

As entradas utilizadas são:

- ENABLE_A e ENABLE_B: Recebem o sinal de PWM do PIC;
- INPUT1,2,3 e 4: Sinais de controle da direção de rotação dos motores. Os INPUTS 1 e 2 são de um motor e os 3 e 4 de outro;
- OUTPUT1,2,3 e 4: Saída para os motores. Os OUTPUTS 1 e 2 são de um motor e os 3 e 4 de outro;
- SENSE_A e SENSE_B: Devem estar aterrados;
- VS: Voltagem de alimentação dos motores, fornecida por baterias em 24V;
- VSS: Voltagem de alimentação do circuito integrado, é a mesma voltagem fornecida ao PIC, 5V;
- GND: Aterramento;

No microcontrolador PIC as entradas e saídas são:

- RA0 e RA1: Sinais de controle de direção do motor A;
- RA2 e RA3: Sinais de controle de direção do motor B;
- RC1: Saída PWM para habilitação do motor B;
- RC2: Saída PWM para habilitação do motor A;
- OSC1 e 2: Ligados ao cristal de 4MHz;
- RB6 e RB7: Canais do rádio;
- RB2, RB3, RB4 e RB5: Sinais do encoder;

3.4. Programação

Foram desenvolvidos uma série de programas durante o projeto. A linguagem escolhida para a programação foi C. Inicialmente o PIC utilizado foi o PIC16F628 e para a versão final foi utilizado o PIC PIC16F874A.

O PIC16F628 é um PIC de 18 pinos, popular, fácil de ser encontrado e barato. No momento inicial do projeto, ele foi utilizado para testar partes do programa final. Porém, para a versão final do projeto, este PIC não atendia a necessidade de se trabalhar com dois sinais de PWM. Portanto, numa etapa posterior, foram adquiridos alguns PICs do modelo PIC16F874A, mais difíceis de serem encontrados e também mais caros. O PIC16F874A possui 40 pinos, e entre vários periféricos que o PIC16F628 não possui, ele possui 2 saídas PWM.

Como poderá ser visto, parte dos programas aqui apresentados foram desenvolvidos para o PIC16F628 e parte para o PIC16F874.

Os programas feitos podem ser vistos abaixo (Tabela 3-2) na ordem em que foram feitos:

Tabela 3-2: Programas desenvolvidos

Nome:	Plataforma:	Breve descrição:
PWM.c	PIC16F628	Programa usado para aprender a usar o PWM do PIC
encoder.c	PIC16F628	Programa usado para aprender a trabalhar com o encoder
radio2motorv3.c	PIC16F628	Programa que recebe 2 canais do receptor e liga

		ou desliga duas saídas.
velocidade.c	PIC16F628	Programa usado para receber o sinal do rádio e decodificá-lo em liga/desliga. Similar ao anterior.
velepwm.c	PIC16F628	Variação do anterior com saída PWM
velepwm2.c	PIC16F628	Otimização do anterior.
velepwm3.c	PIC16F874	Versão do anterior para PIC18F874
Velepwm4.c	PIC16F874	Variação do anterior para dois canais e dois motores
velepwm5.c	PIC16F874	Otimização do anterior utilizando interrupção para tratar os sinais de rádio
velepwm6.c	PIC16F874	Ultima versão do velepwm mais otimizada
velpwmenc.c	PIC16F874	Agora o programa trabalha com encoder, mostrando na porta C se a velocidade do motor ultrapassa determinado ponto e a direção do motor
Velpwmenc2.c	PIC16F874	Melhorando o anterior
Velpwmenc3.c	PIC16F874	Tentando fazer o PID
Velpwmenc4.c	PIC16F874	Melhorando o anterior

Os programas estão no Apêndice A e são comentados e têm seus resultados discutidos em *6 - Resultados e Discussão.*

4
Resultados e conclusões:

Os programas desenvolvidos estão brevemente descritos no Apêndice A. Foram obtidos resultados dentro do esperado para o controle em malha aberta:

- O robô respondia corretamente aos comandos recebidos;
- O programa não apresentou problemas;

Ao aplicar-se o controle P.I.D. os resultados não foram tão bons assim.:

- Não conseguiu-se calibrar corretamente os parâmetros devido a dificuldade de calibração inerente do PIC não trabalhar com operações aritméticas;
- Alguns programas apresentaram oscilações na resposta em velocidade;

Detalhes dos resultados obtidos com cada programa podem ser verificados no apêndice.

Concluiu-se que a aplicação do P.I.D. como foi feita não levaria a resultados melhores devido à incapacidade dos PICS utilizados de trabalhar com operações aritméticas. As contas eram efetuadas através de operações lógicas para não comprometer a performance dos programas. Melhores resultados poderiam ser obtidos com microcontroladores que trabalham com operações aritméticas, principalmente se trabalharem com operações em ponto flutuante.

Pelo fato do controle P.I.D. não ter apresentado bons resultados, não foi efetuada uma configuração detalhada dos parâmetros de controle.

Para trabalhos futuros fica proposto o uso de outros microcontroladores e o acerto dos parâmetros de controle.

5
Referências Bibliográficas:

1 Jones, J.J.; Flynn. A.M. **Mobile Robots – Inspiration to Implementation.** A K Peters, ISBN 1-56881-011-3.

2 Clark, D. **Building Robot Drive Trains.** TAB Robotics.

3 Ogata K. **Sistemas de Control en Tiempo Discreto.** Pearson Educación.

Apêndices A – Programas

A.1

encoder.c

```
// Nome do programa: encoder.c
//
// Teste do encoder - Programa usado para aprender a trabalhar com
o encoder junto ao PIC.
//
// ConfiguraÁ„o:
//
// PIC: PIC16F628
// Oscillator: INTRC I/O
// Watchdog: Off
// Power Up Timer: off
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define ENC1C1 RB4                      // Define ENC1C1 como
entrada do sinal 1 do encoder
#define ENC1C1 RB5                      // Define ENC1C2 como
entrada do sinal 2 do encoder

bit IN1;                                // Guarda valor das
entradas dos encoders
bit DIR1;                               // DireÁ„o do encoder
char POS1D, POS1E;                      // PosiÁ„o do encoder
char POS1DF, POS1EF;                    // PosiÁ„o do encoder -
buffer

bit VEL1;                               // Velocidade do encoder

// Tratando as interrupÁıes:
void interrupt int_isr(void){

// InterrupÁ„o da Porta B:
    if (RBIF){                          // Verifica flag de
interrupÁ„o da Porta B
        if(ENC1C1 & !IN1){              // Verifica decida do
```

```
pulso na entrada do encoder
            DIR1 = ENC1C2;              // Armazena direÁ„o do
encoder pelo estado do sinal 2
                if(DIR1){               // Caso direÁ„o positiva
                    POS1D++;            // Conta na posiÁ„o do
encoder - direita
                }
                else{                   // Caso direÁ„o negativa
                    POS1E++;            // Conta na posiÁ„o do
encoder - esquerda
                }

        }
        IN1 = ENC1C1;                   // Atualiza estado de IN1
com estado atual do sinal 1
        RBIF=0;                         // Zera flag de
interrupÁ„o
    }

// InterrupÁ„o do Timer 1:
    if (TMR1IF) {                       // Verifica flag de
interrupÁ„o do timer 0
        TMR1IF = 0;                     // Zera flag de
interrupÁ„o
        TMR1ON =0;                      // Para timer 1
        TMR1H = 0b10110001;             // Refine perÌodo para
contagem da velocidade
        TMR1L = 0b11011111;
        TMR1ON = 1;                     // Liga timer 1
        POS1DF = POS1D;                 // Guarda a rotaÁ„o para
direita no buffer
        POS1EF = POS1E;                 // Guarda a rotaÁ„o para
esquerda no buffer
        POS1D = POS1E = 0;              // Zera posiÁ„o do encoder
    }


}


void main(void){

// ConfiguraÁ„o das portas:
    TRISA = 0x00;   // Configura Porta A como saÌda
    TRISB = 0xFF;   // Configura Porta B como entrada

// ConfiguraÁ„o das interrupÁıes:
    GIE = 1;        // Liga interrupÁ„o global
    RBIE = 1;       // Liga interrupÁ„o na Porta B (pinos 4,5,6 e
7)
    RBIF = 0;       // Desliga flag de interrupÁ„o da Porta B
    TMR1IE = 1;     // Liga interrupÁ„o do Timer 1
```

```
    TMR1IF = 0;     // Desliga flag de interrupÁ„o do Timer 1

// Acertando o Timer 1:
    TMR1H = 0b10110001;     // Define perÌodo para contagem da
velocidade
    TMR1L = 0b11011111;
    TMR1ON = 1;             // Liga Timer 1

// Inicializando as vari·veis:
    PORTA = 0x00;           // Zera todos os bits da Porta A
    IN1 = ENC1C1;           // Guarda o estado da entrada do
encoder
    VEL1 = 0;               // Zera vari·vel velocidade
    DIR1 = 0;               // Zera vari·vel direÁ„o

// Programa em loop:
    while(1){
        PORTA = DIR1 | (VEL1 << 1);             // Coloca
informaÁ„o de velocida e direÁ„o na saÌda da Porta A
        VEL1 = ((char) (POS1D-POS1E)) > 100;    // Calcula  bit
VEL1, ser· 1 se a contagem do encoder for acima de 100 pulsos no
perÌodo usado
    }

}
```

A.2

pwm.c

```
// Nome do programa: PWM.c
//
// Usando o PWM - Programa usado para aprender a usar o PWM do PIC
//
// ConfiguraÁ„o:
//
// PIC: PIC16F628
// Oscillator: INTRC I/O
// Watchdog: Off
// Power Up Timer: On
// Brown Out Detect: Enabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>

void main(void){

    PR2=0xFF;               // Define perÌodo do PWM
    CCPR1L=0xFF;            // DefÌne tamanho do primeiro pulso do
PWM
    TRISB=0b11110111;       // Configura Porta B
    T2CON=0xFF;             // Configura Timer 2 (associado com o
PWM)
    CCP1CON=0b00001100;     // Configura mÛdulo PWM

    while(1){               // MantÍm o programa rodando
    }

}
```

A.3

radio2motorv3.c

```
// Nome do programa: radio2motorv3.c
//
// Radio com 2 canais
// Programa para receber 2 canais do receptor e ligar ou desligar
duas saÌdas.
//
// ConfiguraÁ„o:
//
// PIC: PIC16F628
// Oscillator: INTRC I/O
// Watchdog: Off
// Power Up Timer: On
// Brown Out Detect: Enabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#include <pic16f6x.h>   // PIC16F628

char VEL1,VEL2;     // Guarda velocidades desejadas (somente 8
bits do timer high)
char SAIDA;         // SaÌda para port A


void main(void){
    TRISA = 0x00;   // Configura Port A como entrada
    TRISB = 0xff;   // Configura Port B como saÌda
    T1CON = 0x00;   // Configura Timer 1
    TMR1ON = 1;     // Liga Timer 1
    SAIDA = 0x00;   // Zera saÌda
    PORTA = 0x00;   // Coloca saÌda em Port A
    VEL1 = 0;       // Zera velocidades
    VEL2 = 0;


    while (1){      // Loop infinito

    while(!RB0){    // Espera RB0 ficar em 1
    }

    TMR1ON = 0;     // Rotina para zerar timer
    TMR1H = 0x00;
    TMR1L = 0x00;
    TMR1ON = 1;

    while(RB0){     // Espera RB0 ficar em 0
```

```
    }

    VEL1 = TMR1H;   // Armazena velocidade

    // Atualiza saÌda

    if(VEL1 >= 7){
        SAIDA = SAIDA | 0b00000001;
    }
    if(VEL1 < 5){
        SAIDA = SAIDA & 0b11111110;
    }

    PORTA = SAIDA;  // Atualiza Port A

    while(!RB5){    // Espera RB5 ficar em 1
    }

    TMR1ON = 0;     // Rotina para zerar timer
    TMR1H = 0x00;
    TMR1L = 0x00;
    TMR1ON = 1;

    while(RB5){     // Espera RB5 ficar em 0
    }

    VEL2 = TMR1H;   // Armazena velocidade

    // Atualiza saÌda:

    if(VEL2 >= 7){
        SAIDA = SAIDA | 0b00000010;
    }
    if(VEL2 < 5){
        SAIDA = SAIDA & 0b11111101;
    }

    PORTA = SAIDA;  // Atualiza Port A

    }
}
```

A.4

velocidade.c

```
// Nome do programa: velocidade.c
//
// RecepÁ„o do r·dio - Programa usado para receber o sinal do
r·dio e decodific·-lo
// em liga/desliga
//
// OBS: Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F628
// Oscillator: INTRC I/O
// Watchdog: Off
// Power Up Timer: off
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>

#define CANAL1 0        // DefiniÁıes para os canais
#define CANAL2 1

char SAIDA;             // Guarda estado da Porta A
char CANAIS;            // Armazena o estado dos canais para
verificar variaÁıes


// FunÁ„o que fica verificando os sinais de r·dio e retorna a
posiÁ„o dos controles de direÁ„o
int velocidade(char canal,int vel){
// Verifica mudanÁa de estado do sinal de entrada relativo ao
canal desejado:
    if ( ((PORTB & (1 << canal)) && 1) != ((CANAIS & (1 << canal))
&& 1)   ){
            if(CANAIS & (1 << canal)){              // Caso o
pulso tenha decido
                CANAIS = CANAIS & ~(1 << canal);    // Atualiza
CANAIS
                return (TMR1H<<8) + TMR1L;          // Retorna o
tempo do pulso - relativo a posiÁ„o do controle
            }
            else{                                   // Caso o
pulso tenha subido
                TMR1ON=0;                           // Para o
Timer 1
```

```
                TMR1H=0x00;                             // Zera o
Timer 1
                TMR1L=0x00;
                TMR1ON=1;                               // Liga o
Timer 1
                CANAIS = CANAIS | (1 << canal);         // Atualiza
CANAIS
                return vel;                             // Retorna a
posiÁ„o j· medida
            }
    }
    else{           // Caso n„o tenha ocorrido mudanÁa no sinal
    return vel;     // Retorna a posiÁ„o j· medida
    }
}

//
void main(void){
    int VEL1, VEL2;     // Armazenam a posiÁ„o do joystick
recebida pelo r·dio

// OBS: O nome destas vari·veis È VEL porque a velocidade desejada
para cada motor È dada pelos
// sinais recebidos pelo r·dio

    // ConfiguraÁ„o das portas
    TRISA = 0x00;       // Configura Porta A como saÌda
    TRISB = 0xff;       // Configura Porta B como entrada

    // ConfiguraÁ„o do Timer 1
    T1CON = 0x00;       // Configura Timer 1
    TMR1ON = 1;         // Liga Timer 1

    // InicializaÁ„o das vari·veis
    VEL1 = 0;           // Zera vari·vel VEL1
    PORTA = 0;          // Zera Porta A
    SAIDA = PORTA;      // Guarda estado da porta A
    CANAIS = 0x00;      // Zera Canais

    while (1){

    // Recebe as posiÁıes do R·dio-Controle:
    VEL1=velocidade(CANAL1, VEL1);
    VEL2=velocidade(CANAL2, VEL2);

    //Liga e desliga bits na saÌda para sabermos as posiÁıes do
R·dio-Controle:

    //Para o primeiro canal:
        if(VEL1 >= 1800){               // Se posiÁ„o acima de
determinado ponto
        SAIDA = SAIDA | 0b00000001;     // Liga bit 0 da vari·vel
```

```
SAIDA
        }
        if(VEL1 < 1200){                // Se abaixo de
determinado ponto
        SAIDA = SAIDA & 0b11111110;     // Desliga bit 0 da
vari·vel SAIDA
        }

    //Para o segundo canal:
        if(VEL2 >= 1800){               // Se posiÁ„o acima de
determinado ponto
        SAIDA = SAIDA | 0b00000010;     // Liga bit 1 da vari·vel
SAIDA
        }
        if(VEL2 < 1200){                // Se abaixo de
determinado ponto
        SAIDA = SAIDA & 0b11111101;     // Desliga bit 1 da
vari·vel SAIDA
        }

        PORTA = SAIDA;                  // Atualiza Porta A com o
estado da vari·vel SAÕDA
    }
}
```

A.5

velepwm.c

```
// Nome do programa: velepwm.c
//
// RecepÁ„o do r·dio e PWM - Programa usado para receber o sinal
do r·dio,decodific·-lo
// e reproduzi-lo em PWM;
//
// OBS:
// - Similar ao velocidade.c (RecepÁ„o do r·dio), porÈm este
programa gera saÌda em PWM
// relativa ao Canal 1. As diferenÁas do programa est„o somente em
main();
//
// - Este programa n„o funcionou muito bem devido a baixa
velocidade do PIC. o sinal
// recebido do r·dio n„o era medido com precis„o.
//
// OBS. Antigas:
// - Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F628
// Oscillator: INTRC I/O
// Watchdog: Off
// Power Up Timer: off
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>

#define CANAL1 0        // DefiniÁıes para os canais
#define CANAL2 1

char SAIDA;             // Guarda estado da Porta A
char CANAIS;            // Armazena o estado dos canais para
verificar variaÁıes

// FunÁ„o que fica verificando os sinais de r·dio e retorna a
posiÁ„o dos controles de direÁ„o
int velocidade(char canal,int vel){
// Verifica mudanÁa de estado do sinal de entrada relativo ao
canal desejado:
    if ( ((PORTB & (1 << canal)) && 1) != ((CANAIS & (1 << canal))
&& 1)   ){
            if(CANAIS & (1 << canal)){              // Caso o
```

```
pulso tenha decido
                CANAIS = CANAIS & ~(1 << canal);    // Atualiza
CANAIS
                return (TMR1H<<8) + TMR1L;          // Retorna o
tempo do pulso - relativo a posiÁ„o do controle
            }
            else{                                   // Caso o
pulso tenha subido
                TMR1ON=0;                           // Para o
Timer 1
                TMR1H=0x00;                         // Zera o
Timer 1
                TMR1L=0x00;
                TMR1ON=1;                           // Liga o
Timer 1
                CANAIS = CANAIS | (1 << canal);     // Atualiza
CANAIS
                return vel;                         // Retorna a
posiÁ„o j· medida
            }
    }
    else{           // Caso n„o tenha ocorrido mudanÁa no sinal
    return vel;     // Retorna a posiÁ„o j· medida
    }
}

void main(void){
    int VEL1, VEL2;     // Armazenam a posiÁ„o do joystick
recebida pelo r·dio

// OBS: O nome destas vari·veis È VEL porque a velocidade desejada
para cada motor È dada pelos
// sinais recebidos pelo r·dio

    // ConfiguraÁ„o das portas
    TRISA = 0x00;       // Configura Porta A como saÌda
    TRISB = 0xff;       // Configura Porta B como entrada

    // ConfiguraÁ„o do Timer 1
    T1CON = 0x00;       // Configura Timer 1
    TMR1ON = 1;         // Liga Timer 1

    // ConfiguraÁ„o do Timer 2 e PWM
    PR2 = 0xDF;             // Periodo do PWM
    CCPR1L = 0x00;          // Tamanho do pulso do PWM
    T2CON = 0xFF;           // Configura Timer 2
    TMR2IE = 0;             // Desliga interrupÁ„o do Timer 2
    CCP1CON = 0b00001100;   // Coloca Timer 2 funcionando para o
PWM

    // InicializaÁ„o das vari·veis
    VEL1 = 0;           // Zera vari·vel VEL1
```

```
    PORTA = 0;          // Zera Porta A
    SAIDA = PORTA;      // Guarda estado da porta A
    CANAIS = 0x00;      // Zera Canais

    while (1){

    // Recebe as posiÁıes do R·dio-Controle:
    VEL1=velocidade(CANAL1, VEL1);
    VEL2=velocidade(CANAL2, VEL2);

    //Liga e desliga bits na saÌda para sabermos as posiÁıes do
R·dio-Controle:

    //Para o primeiro canal:

        if(VEL1 >= 1800){               // Se posiÁ„o acima de
determinado ponto
        SAIDA = SAIDA | 0b00000001;     // Liga bit 0 da vari·vel
SAIDA
        }
        if(VEL1 < 1200){                // Se abaixo de
determinado ponto
        SAIDA = SAIDA & 0b11111110;     // Desliga bit 0 da
vari·vel SAIDA
        }
        if(VEL1 >= 1500){               // Se posiÁ„o acima de
determinado ponto (metade do pulso m·ximo de controle)
        CCPR1L = (VEL1 - 1500) >> 1;    // Acerta tamanho do sinal
do PWM
        }
        else{                           // Sen„o
        CCPR1L = 0x00;                  // Deixa PWM sem sinal
nenhum
        }

    //Para o segundo canal:
        if(VEL2 >= 1800){               // Se posiÁ„o acima de
determinado ponto
        SAIDA = SAIDA | 0b00000010;     // Liga bit 1 da vari·vel
SAIDA
        }
        if(VEL2 < 1200){                // Se abaixo de
determinado ponto
        SAIDA = SAIDA & 0b11111101;     // Desliga bit 1 da
vari·vel SAIDA
        }

        PORTA = SAIDA;                  // Atualiza Porta A com o
estado da vari·vel SAÕDA
    }
}
```

A.6

velepwm2.c

```
// Nome do programa: velepwm2.c
//
// RecepÁ„o do r·dio e PWM v.2 - Programa usado para receber o
sinal do r·dio,decodific·-lo
// e reproduzi-lo em PWM;
//
// OBS:
// - Tentando otimizar o programa anterior para conseguir melhor
leitura do sinal do r·dio. Para tal a funÁ„o
// que lÍ VEL foi simplificada e se utiliza um sÛ canal, o que n„o
È a situaÁ„o real para o projeto.Este programa
// apresenta melhor resultado porÈm ainda n„o muito bom. Numa
prÛxima etapa foram conseguidos
// melhores resultados trabalhando-se com interrupÁ„o.
//
// OBS. Antigas:
// - Similar ao velocidade.c (RecepÁ„o do r·dio), porÈm este
programa gera saÌda em PWM
// relativa ao Canal 1. As diferenÁas do programa est„o somente em
main();
//
// - Este programa n„o funcionou muito bem devido a baixa
velocidade do PIC. o sinal
// recebido do r·dio n„o era medido com precis„o.
//
// - Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F628
// Oscillator: INTRC I/O
// Watchdog: Off
// Power Up Timer: off
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off
#include <pic.h>


char SAIDA;             // Guarda estado da Porta A
char CANAL1,CANAL2;     // Guarda estado dos canais
int VEL1, VEL2;         // Armazenam a posiÁ„o do joystick
recebida pelo r·dio
// OBS: O nome destas vari·veis È VEL porque a velocidade desejada
para cada motor È dada pelos
// sinais recebidos pelo r·dio
```

```
// FunÁ„o que fica verificando o sinal do canal 1 do r·dio e
calcula a posiÁ„o dos controles de direÁ„o
int vel1(){
    if ( RB0 != CANAL1 ){       // Verifica mudanÁa de estado do
sinal de entrada
            if(CANAL1 ){        // Caso o pulso tenha decido
                CANAL1 = 0;     // Atualiza estado do canal
                VEL1 = (TMR1H<<8)+TMR1L;    // Calcula o tempo do
pulso - relativo a posiÁ„o do controle
            }
            else{
                TMR1ON = 0;         // Caso o pulso tenha subido
                TMR1H=0x00;         // Zera o Timer 1
                TMR1L=0x00;
                TMR1ON=1;           // Liga o Timer 1
            CANAL1 = 1;             // Atualiza estado do canal
            }
    }
}

void main(void){
    // ConfiguraÁ„o das portas
    TRISA = 0x00;           // Configura Porta A como saÌda
    TRISB = 0b11110111;     // Configura Porta B


    // ConfiguraÁ„o do Timer 2 e PWM
    PR2 = 0xDF;             // Periodo do PWM
    CCPR1L = 0x00;          // Tamanho do pulso do PWM
    T2CON = 0xFF;           // Configura Timer 2
    TMR2IE = 0;             // Desliga interrupÁ„o do Timer 2
    CCP1CON = 0b00001100;   // Coloca Timer 2 funcionando para o
PWM

    // ConfiguraÁ„o do Timer 1
    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    // InicializaÁ„o das vari·veis
    VEL1=0;                 // Zera vari·vel VEL1
    SAIDA = 0b00000001;     // Coloca estado de SAIDA

    while (1){

    // Atualiza posiÁ„o do r·dio controle:
    vel1();

    //Liga e desliga bits na saÌda para sabermos as posiÁıes do
R·dio-Controle:

        if(VEL1 >= 1800){               // Se posiÁ„o acima de
```

```
determinado ponto
        SAIDA = SAIDA | 0b00000001;     // Liga bit 0 da vari·vel
SAIDA
        }
        if(VEL1 < 1200){                // Se abaixo de
determinado ponto
        SAIDA = SAIDA & 0b11111110;     // Desliga bit 0 da
vari·vel SAIDA
        }
        if(VEL1 >= 1500){               // Se posiÁ„o acima de
determinado ponto (metade do pulso m·ximo de controle)
        CCPR1L = (VEL1 - 1500) >> 1;    // Acerta tamanho do sinal
do PWM
        }
        else{                           // Sen„o
        CCPR1L = 0x00;                  // Deixa PWM sem sinal
nenhum
        }

    }
}
```

A.7

velepwm3.c

```
// Nome do programa: velepwm3.c
//
// RecepÁ„o do r·dio e PWM v.3 - PIC16F874A - Programa usado para
receber o sinal do r·dio,decodific·-lo
// e reproduzi-lo em PWM. Trabalha com um motor;
//
// OBS:
// - Vers„o para PIC16F874A. Perceba que os bits 0 e 1 da porta A
s„o utilizados para controle de direÁ„o do motor,
// quando os dois bits est„o ligados, o C.I. que recebe estes
sinais de controle mantem o motor parado;
//
// OBS. Antigas:
// - Tentando otimizar o programa anterior para conseguir melhor
leitura do sinal do r·dio. Para tal a funÁ„o
// que lÍ VEL foi simplificada e se utiliza um sÛ canal, o que n„o
È a situaÁ„o real para o projeto.Este programa
// apresenta melhor resultado porÈm ainda n„o muito bom. Numa
prÛxima etapa foram conseguidos
// melhores resultados trabalhando-se com interrupÁ„o;
//
// - Similar ao velocidade.c (RecepÁ„o do r·dio), porÈm este
programa gera saÌda em PWM
// relativa ao Canal 1. As diferenÁas do programa est„o somente em
main();
//
// - Este programa n„o funcionou muito bem devido a baixa
velocidade do PIC. o sinal
// recebido do r·dio n„o era medido com precis„o.
//
// - Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F874A
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define RADIO1 RB7      // Define a entrada relativa ao sinal do
r·dio 1
```

```
char SAIDA;             // Guarda estado da Porta A
char CANAL1,CANAL2;     // Guarda estado dos canais
int VEL1, VEL2;         // Armazenam a posiÁ„o do joystick
recebida pelo r·dio
// OBS: O nome destas vari·veis È VEL porque a velocidade desejada
para cada motor È dada pelos
// sinais recebidos pelo r·dio

// FunÁ„o que fica verificando o sinal do canal 1 do r·dio e
calcula a posiÁ„o dos controles de direÁ„o
void vel1(){
    if ( RADIO1 != CANAL1   ){                      // Verifica
mudanÁa de estado do sinal de entrada
            if( CANAL1 ){                           // Caso o
sinal tenha decido

                VEL1 = ( TMR1H << 8 )+TMR1L;        // Calcula o
tempo do pulso - relativo a posiÁ„o do controle

                if(VEL1 >= 1550){                   // Se posiÁ„o
acima de determinado ponto
                    CCPR1L = (VEL1 - 1500) >> 1;    // Acerta
tamanho do sinal do PWM
                    SAIDA = SAIDA | 0b00000001;     // Liga bit 0
da vari·vel SAIDA
                    SAIDA = SAIDA & 0b11111101;     // Desliga bit
1 da vari·vel SAIDA
                }

                else{

                    if(VEL1 <= 1450){
                        CCPR1L = (1500 - VEL1) >> 1;// Acerta
tamanho do sinal do PWM
                        SAIDA = SAIDA | 0b00000010; // Liga bit 1
da vari·vel SAIDA
                        SAIDA = SAIDA & 0b11111110; // Desliga bit
0 da vari·vel SAIDA
                    }

                    else{
                        CCPR1L = 0x00;              // Deixa PWM
sem sinal nenhum
                        SAIDA = SAIDA | 0b00000011; // Liga bits 0
e 1 da vari·vel SAIDA
                    }

                }

                PORTA = SAIDA;                      // Atualiza
Port A com mudanÁas
```

```
            }

            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Caso o
pulso tenha subido
                TMR1H=0x00;                         // Zera o
Timer 1
                TMR1L=0x00;
                TMR1ON=1;                           // Liga o
Timer 1
            }

            CANAL1 = !CANAL1;                       // Atualiza
estado do canal
    }
}

void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda
    PR2 = 0x4F;             // PerÌodo PWM
    CCPR1L = 0x00;          // Tamanho PWM - High Bits
    T2CON = 0b11111101;     // ConfiguraÁ„o Timer 2
    TMR2IE = 0;
    CCP1CON = 0b00001100;   // Configura PWM
    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1
    VEL1 = 0;               // Zera velocidade
    SAIDA = 0b00000000;     // Zera saÌda inicial

    while (1){

    vel1();                 // Atualiza posiÁ„o do r·dio controle

    }
}
```

A.8

velepwm4.c

```
// Nome do programa: velepwm4.c
//
// RecepÁ„o do r·dio e PWM v.4 - PIC16F874A - Programa usado para
receber o sinal do r·dio,decodific·-lo
// e reproduzi-lo em PWM. Trabalha com dois motores;
//
// OBS:
//  - Fazendo o programa trabalhar com dois canais;
//
// OBS. Antigas:
// - Vers„o para PIC16F874A. Perceba que os bits 0 e 1 da porta A
s„o utilizados para controle de direÁ„o do motor,
// quando os dois bits est„o ligados, o C.I. que recebe estes
sinais de controle mantem o motor parado;
//
// - Tentando otimizar o programa anterior para conseguir melhor
leitura do sinal do r·dio. Para tal a funÁ„o
// que lÍ VEL foi simplificada e se utiliza um sÛ canal, o que n„o
È a situaÁ„o real para o projeto.Este programa
// apresenta melhor resultado porÈm ainda n„o muito bom. Numa
prÛxima etapa foram conseguidos
// melhores resultados trabalhando-se com interrupÁ„o;
//
// - Similar ao velocidade.c (RecepÁ„o do r·dio), porÈm este
programa gera saÌda em PWM
// relativa ao Canal 1. As diferenÁas do programa est„o somente em
main();
//
// - Este programa n„o funcionou muito bem devido a baixa
velocidade do PIC. o sinal
// recebido do r·dio n„o era medido com precis„o;
//
// - Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F874A
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off


#include <pic.h>
```

```
#define RADIO1 RB7      // Define a entrada relativa ao sinal do
r·dio 1
#define RADIO2 RB6      // Define a entrada relativa ao sinal do
r·dio 2

char SAIDA;             // Guarda estado da Porta A
bit CANAL1,CANAL2;      // Guarda estado dos canais
unsigned int VEL1, VEL2;            // Armazenam a posiÁ„o do
joystick recebida pelo r·dio

// FunÁ„o que fica verificando o sinal do canal 1 do r·dio e
calcula a posiÁ„o dos controles de direÁ„o
void vel1(){
    if ( RADIO1 != CANAL1   ){                          //
Verifica mudanÁa de estado do sinal de entrada
            if( CANAL1 ){                               // Caso o
sinal tenha decido

                VEL1 = ( TMR1H << 8 )+TMR1L;        // Calcula o
tempo do pulso - relativo a posiÁ„o do controle

                if(VEL1 >= 1550){                   // Se posiÁ„o
acima de determinado ponto
                    CCPR1L = (VEL1 - 1500) >> 1;    // Acerta
tamanho do sinal do PWM
                    SAIDA = SAIDA | 0b00000001;     // Liga bit 0
da vari·vel SAIDA
                    SAIDA = SAIDA & 0b11111101;     // Desliga bit
1 da vari·vel SAIDA
                }

                else{

                    if(VEL1 <= 1450){
                        CCPR1L = (1500 - VEL1) >> 1;// Acerta
tamanho do sinal do PWM
                        SAIDA = SAIDA | 0b00000010; // Liga bit 1
da vari·vel SAIDA
                        SAIDA = SAIDA & 0b11101110; // Desliga
bits 0  e 4 da vari·vel SAIDA
                    }

                    else{
                        // Deixa PWM sem sinal nenhum
                        CCPR1L = 0x00;
                    }

                }

                PORTA = SAIDA;                      // Atualiza
Port A com mudanÁas
```

```
            }

            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Caso o
pulso tenha subido
                TMR1H=0x00;                         // Zera o
Timer 1
                TMR1L=0x00;
                TMR1ON=1;                           // Liga o
Timer 1
            }

            CANAL1 = RADIO1;                        // Atualiza
estado do CANAL1
    }
}

// FunÁ„o que fica verificando o sinal do canal 2 do r·dio e
calcula a posiÁ„o dos controles de direÁ„o.
// FunÁ„o similar a vel1 e por isto n„o est· comentada.
void vel2(){
    if ( RADIO2 != CANAL2   ){
            if( CANAL2 ){

                VEL2 = ( TMR1H << 8 )+TMR1L;

                if(VEL2 >= 1550){
                    CCPR2L = (VEL2 - 1500) >> 1;
                    SAIDA = SAIDA | 0b00100100;
                    SAIDA = SAIDA & 0b11110111;
                }

                else{

                    if(VEL2 <= 1450){
                        CCPR2L = (1500 - VEL2) >> 1;
                        SAIDA = SAIDA | 0b00001000;
                        SAIDA = SAIDA & 0b11011011;
                    }

                    else{
                        CCPR2L = 0x00;
                    }

                }

                PORTA = SAIDA;

            }

            else{
```

```
                TMR1ON = 0;
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }

            CANAL2 = RADIO2;
    }
}

void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    PR2 = 0x4F;             // PerÌodo PWM
    CCPR1L = 0x00;          // Tamanho PWM - High Bits
    CCPR2L = 0x00;          // Tamanho PWM - High Bits
    CCP1CON = 0b00001100;   // Configura PWM
    CCP2CON = 0b00001100;   // Configura PWM

    T2CON = 0b11111101;     // ConfiguraÁ„o Timer 2
    TMR2IE = 0;
    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    VEL1 = 0;               // Zera velocidade
    VEL1 = 2;               // Zera velocidade
    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
    CANAL2 = RADIO2;

    while (1){

    vel1();                 // Atualiza posiÁ„o do r·dio controle
- canal 1
    vel2();                 // Atualiza posiÁ„o do r·dio controle
- canal 2

    }
}
```

A.9

velepwm5.c

```
// Nome do programa: velepwm5.c
//
// RecepÁ„o do r·dio e PWM v.4 - PIC16F874A - Programa usado para
receber o sinal do r·dio,decodific·-lo
// e reproduzi-lo em PWM. Trabalha com dois motores;
//
// OBS:
//  - Tratando agora com interrupÁ„o para otimizar o programa. Os
resultados s„o muito melhores.
//
// OBS. Antigas:
//  - Fazendo o programa trabalhar com dois canais;
//
// - Vers„o para PIC16F874A. Perceba que os bits 0 e 1 da porta A
s„o utilizados para controle de direÁ„o do motor,
// quando os dois bits est„o ligados, o C.I. que recebe estes
sinais de controle mantem o motor parado;
//
// - Tentando otimizar o programa anterior para conseguir melhor
leitura do sinal do r·dio. Para tal a funÁ„o
// que lÍ VEL foi simplificada e se utiliza um sÛ canal, o que n„o
È a situaÁ„o real para o projeto.Este programa
// apresenta melhor resultado porÈm ainda n„o muito bom. Numa
prÛxima etapa foram conseguidos
// melhores resultados trabalhando-se com interrupÁ„o;
//
// - Similar ao velocidade.c (RecepÁ„o do r·dio), porÈm este
programa gera saÌda em PWM
// relativa ao Canal 1. As diferenÁas do programa est„o somente em
main();
//
// - Este programa n„o funcionou muito bem devido a baixa
velocidade do PIC. o sinal
// recebido do r·dio n„o era medido com precis„o;
//
// - Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F874A
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off
```

```
#include <pic.h>
#define RADIO1 RB7
#define RADIO2 RB6

#include <pic.h>
#define RADIO1 RB7      // Define a entrada relativa ao sinal do
r·dio 1
#define RADIO2 RB6      // Define a entrada relativa ao sinal do
r·dio 2

char SAIDA;             // Guarda estado da Porta A
bit CANAL1,CANAL2;      // Guarda estado dos canais
unsigned int VEL1, VEL2;            // Armazenam a posiÁ„o do
joystick recebida pelo r·dio


// Tratando as interrupÁıes:
void interrupt int_isr(void){
    if(RBIF){                                       // Tratando
interrupÁ„o na porta B

        if ( RADIO1 != CANAL1   ){                  // Verifica
mudanÁa de estado do sinal de radio 1
            if( CANAL1 ){                           // Caso o
sinal tenha decido
                VEL1 = ( TMR1H << 8 ) + TMR1L;      // Calcula
velocidade em cima do timer 1
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            CANAL1 = RADIO1;                        // Atualiza
estado do CANAL1
        }

        if ( RADIO2 != CANAL2   ){                  // Verifica
mudanÁa de estado do sinal do radio 2
            if( CANAL2 ){                           // Caso o
sinal tenha decido
                VEL2 = ( TMR1H << 8 ) + TMR1L;      // Calcula
velocidade em cima do timer 1
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
```

```
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            CANAL2 = RADIO2;                        // Atualiza
estado do CANAL2
        }
        RBIF = 0;                                   // Zera flag
de interrupÁ„o da porta B
    }
}

void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    GIE = 1;                // Liga interrupÁ„o global
    RBIE = 1;               // Liga interrupÁ„o da porta B
    RBIF = 0;               // Zera flag de int. da porta B

    PR2 = 0x4F;             // PerÌodo PWM
    CCPR1L = 0x00;          // Tamanho PWM - High Bits
    CCPR2L = 0x00;          // Tamanho PWM - High Bits
    CCP1CON = 0b00001100;   // Configura PWM
    CCP2CON = 0b00001100;   // Configura PWM

    T2CON = 0b11111101;     // ConfiguraÁ„o Timer 2
    TMR2IE = 0;
    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    VEL1 = 0;               // Zera velocidade
    VEL1 = 2;               // Zera velocidade
    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
    CANAL2 = RADIO2;

    while (1){

            if(VEL1 >= 1550){                   // Se posiÁ„o
acima de determinado ponto
                CCPR1L = (VEL1 - 1500) >> 1;    // Acerta tamanho
do sinal do PWM
                SAIDA = SAIDA | 0b00000001;     // Liga bit 0 da
vari·vel SAIDA
                SAIDA = SAIDA & 0b11111101;     // Desliga bit 1
da vari·vel SAIDA
            }
            else{
```

```
                if(VEL1 <= 1450){
                    CCPR1L = (1500 - VEL1) >> 1;// Acerta tamanho
do sinal do PWM
                    SAIDA = SAIDA | 0b00000010; // Liga bit 1 da
vari·vel SAIDA
                    SAIDA = SAIDA & 0b11111110; // Desliga bit 0
da vari·vel SAIDA
                }
                else{
                    CCPR1L = 0x00;              // Deixa PWM sem
sinal nenhum
                }
            }
            if(VEL2 >= 1550){
                CCPR2L = (VEL2 - 1500) >> 1;
                SAIDA = SAIDA | 0b00000100;
                SAIDA = SAIDA & 0b11111111;
            }

            else{
            // O mesmo procedimento viso acima para VEL1 È feito
para tratamento de VEL2
                if(VEL2 <= 1450){
                    CCPR2L = (1500 - VEL2) >> 1;
                    SAIDA = SAIDA | 0b00001000;
                    SAIDA = SAIDA & 0b11011011;
                }

                else{
                    CCPR2L = 0x00;
                }

            }

        PORTA = SAIDA;                              // Atualiza
Port A com mudanÁas

    }
}
```

A.10

velepwm6.c

```
// Nome do programa: velepwm6.c
//
// RecepÁ„o do r·dio e PWM v.5 - PIC16F874A - Programa usado para
receber o sinal do r·dio,decodific·-lo
// e reproduzi-lo em PWM. Trabalha com dois motores;
//
// OBS:
//  - Programa anterior otimizado;
//
// OBS. Antigas:
//  - Tratando agora com interrupÁ„o para otimizar o programa. Os
resultados s„o muito melhores;
//
//  - Fazendo o programa trabalhar com dois canais;
//
// - Vers„o para PIC16F874A. Perceba que os bits 0 e 1 da porta A
s„o utilizados para controle de direÁ„o do motor,
// quando os dois bits est„o ligados, o C.I. que recebe estes
sinais de controle mantem o motor parado;
//
// - Tentando otimizar o programa anterior para conseguir melhor
leitura do sinal do r·dio. Para tal a funÁ„o
// que lÍ VEL foi simplificada e se utiliza um sÛ canal, o que n„o
È a situaÁ„o real para o projeto.Este programa
// apresenta melhor resultado porÈm ainda n„o muito bom. Numa
prÛxima etapa foram conseguidos
// melhores resultados trabalhando-se com interrupÁ„o;
//
// - Similar ao velocidade.c (RecepÁ„o do r·dio), porÈm este
programa gera saÌda em PWM
// relativa ao Canal 1. As diferenÁas do programa est„o somente em
main();
//
// - Este programa n„o funcionou muito bem devido a baixa
velocidade do PIC. o sinal
// recebido do r·dio n„o era medido com precis„o;
//
// - Trabalha com dois canais de r·dio / duas velocidades;
//
// ConfiguraÁ„o:
//
// PIC: PIC16F874A
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
```

```
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define RADIO1 RB7
#define RADIO2 RB6

char SAIDA;                         // Port A
bit VEL1AC, VEL2AC;                 // Bits de aÁ„o
bit CANAL1,CANAL2;                  // Armazena o estado dos
canais
char VELD1H,VELD1L,VELD2H,VELD2L;   // Vel. desejadas High e Low
unsigned int VELD1,VELD2;           // Vel. desejadas completo
bit ENC1, ENC2;

void interrupt int_isr(void){       // Trata interrupÁıes
    if(RBIF){

        if ( RADIO1 != CANAL1   ){                          //
Verifica mudanÁa de estado em RB0
            if( CANAL1 ){                           // Caso o
sinal tenha decido
                VELD1H = TMR1H;
                VELD1L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            CANAL1 = RADIO1;                        // Atualiza
estado de RB0 em CANAL1
            VEL1AC = 1;
        }

        if ( RADIO2 != CANAL2   ){                          //
Verifica mudanÁa de estado em RB0
            if( CANAL2 ){                           // Caso o
sinal tenha decido
                VELD2H = TMR1H;
                VELD2L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
```

```
            }
            CANAL2 = RADIO2;                        // Atualiza
estado de RB0 em CANAL1
            VEL2AC = 1;
        }
        RBIF = 0;
    }
}

void vel1(){
    VELD1 = ( VELD1H << 8 ) + VELD1L;       // Calcula velocidade
em cima do timer 1
    if(VELD1 >= 1550){
        CCPR1L = (VELD1 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000001;
        SAIDA = SAIDA & 0b11111101;
    }
    else{
        if(VELD1 <= 1450){
            CCPR1L = (1500 - VELD1) >> 1;
            SAIDA = SAIDA | 0b00000010;
            SAIDA = SAIDA & 0b11111110;
        }
        else{
            CCPR1L = 0x00;
        }
    }
    PORTA = SAIDA;                              // Atualiza Port A
com mudanÁas
}

void vel2(){
    VELD2 = ( VELD2H << 8 ) + VELD2L;       // Calcula velocidade
em cima do timer 1
    if(VELD2 >= 1550){
        CCPR2L = (VELD2 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000100;
        SAIDA = SAIDA & 0b11111111;
    }
    else{
        if(VELD2 <= 1450){
            CCPR2L = (1500 - VELD2) >> 1;
            SAIDA = SAIDA | 0b00001000;
            SAIDA = SAIDA & 0b11011011;
        }
        else{
            CCPR2L = 0x00;
        }
    }
    PORTA = SAIDA;                              // Atualiza Port A
com mudanÁas
}
```

```
void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    GIE = 1;                // Liga interrupÁ„o global
    RBIE = 1;               // Liga interrupÁ„o da porta B
    RBIF = 0;               // Zera flag de int. da porta B

    PR2 = 0x4F;             // PerÌodo PWM
    CCPR1L = 0x00;          // Tamanho PWM - High Bits
    CCPR2L = 0x00;          // Tamanho PWM - High Bits
    CCP1CON = 0b00001100;   // Configura PWM
    CCP2CON = 0b00001100;   // Configura PWM

    T2CON = 0b11111101;     // ConfiguraÁ„o Timer 2
    TMR2IE = 0;
    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    VEL1AC = 0;
    VEL2AC = 0;             // Zera aÁıes

    VELD1 = 0;              // Zera velocidade
    VELD2 = 0;
    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
    CANAL2 = RADIO2;

    while (1){
        if(VEL1AC){
        vel1();
        VEL1AC = 0;
        }
        if(VEL2AC){
        vel2();
        VEL2AC = 0;
        }
    }
}
```

A.11

velepwmenc.c

```
// Nome do programa: velpwmenc.C
//
// Controle malha aberta 2 canais, 1 PWM
//
// OBS ATUAIS:
// Em processo de adicionar um encoder - Lendo vel + ou -.
//
// OBS ANTIGAS:
// Otimizando programa anterior.
// Tentando fazer para 2 canais
// Tentado tratar com interrupÁ„o - RESULTADO: Funciona muito
melhor.
// Vers„o para PIC 16F874A
// DUAS DIRE«'ES
// Vers„o buscando maior velocidade do programa.
//
// ConfiguraÁ„o:
//
// PIC: PIC16F877
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define RADIO1 RB7
#define RADIO2 RB6
#define ENC1C1 RB5
#define ENC2C1 RB4
#define ENC1C2 RB3
#define ENC2C2 RB2


char SAIDA;                             // Port A
bit V1OK, V2OK;                         // Para saber inÌcios de
timer1 referentes aos sinais de r·dio
bit VEL1AC, VEL2AC;                     // Bits de aÁ„o
bit CANAL1,CANAL2;                      // Armazena o estado dos
canais
char VELD1H,VELD1L,VELD2H,VELD2L;       // Vel. desejadas High e
Low
unsigned int VELD1,VELD2;               // Vel. desejadas completo
bit IN1,IN2;                            // Guarda valor das
entradas dos encoders
```

```
bit DIR1,DIR2;                          // DireÁ„o do encoder
char POS1D, POS1E, POS2D, POS2E;        // PosiÁ„o do encoder
char POS1DF, POS1EF, POS1DF, POS1EF;    // PosiÁ„o do encoder -
buffer

void interrupt int_isr(void){       // Trata interrupÁıes
    if(RBIF){

        if(ENC1C1 & !IN1){          // Trata Encoder 1
            DIR1 = ENC1C2;
                POS1D += DIR1;
                POS1E += !DIR1;
        }
        IN1 = ENC1C1;

        if ( RADIO1 != CANAL1   ){                  // Verifica
mudanÁa de estado em RB0
            if( CANAL1 ){                           // Caso o
sinal tenha decido
                VELD1H = TMR1H;
                VELD1L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
                V1OK = 1;
            }
            CANAL1 = RADIO1;                        // Atualiza
estado de RB0 em CANAL1
            VEL1AC = 1;
        }

        if ( RADIO2 != CANAL2   ){                      //
Verifica mudanÁa de estado em RB0
            if( CANAL2 ){                           // Caso o
sinal tenha decido
                VELD2H = TMR1H;
                VELD2L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
                V2OK = 1;
            }
```

```
            CANAL2 = RADIO2;                        // Atualiza
estado de RB0 em CANAL1
            VEL2AC = 1;
        }
        RBIF = 0;
    }
/*
    if (TMR0IF) {

        TMR0IF = 0;

        TMR0 = 0b01100011;
        POS1DF = POS1D;
        POS1EF = POS1E;
        POS1D = POS1E = 0;
    }
*/
}

void vel1(){
    VELD1 = ( VELD1H << 8 ) + VELD1L;       // Calcula velocidade
em cima do timer 1
    if(VELD1 >= 1550){
//      CCPR1L = (VELD1 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000001;
        SAIDA = SAIDA & 0b11111101;
    }
    else{
        if(VELD1 <= 1450){
//          CCPR1L = (1500 - VELD1) >> 1;
            SAIDA = SAIDA | 0b00000010;
            SAIDA = SAIDA & 0b11111110;
        }
        else{
//          CCPR1L = 0x00;
        }
    }
}

void vel2(){
    VELD2 = ( VELD2H << 8 ) + VELD2L;       // Calcula velocidade
em cima do timer 1
    if(VELD2 >= 1550){
//      CCPR2L = (VELD2 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000100;
        SAIDA = SAIDA & 0b11111111;
    }
    else{
        if(VELD2 <= 1450){
//          CCPR2L = (1500 - VELD2) >> 1;
            SAIDA = SAIDA | 0b00001000;
            SAIDA = SAIDA & 0b11011011;
```

```
        }
        else{
//          CCPR2L = 0x00;
        }
    }
}


void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    GIE = 1;                // Liga interrupÁ„o global
    RBIE = 1;               // Liga interrupÁ„o da porta B
    RBIF = 0;               // Zera flag de int. da porta B

//  PR2 = 0x4F;             // PerÌodo PWM
//  CCPR1L = 0x00;          // Tamanho PWM - High Bits
//  CCPR2L = 0x00;          // Tamanho PWM - High Bits
//  CCP1CON = 0b00001100;   // Configura PWM
//  CCP2CON = 0b00001100;   // Configura PWM

//  T2CON = 0b11111101;     // ConfiguraÁ„o Timer 2
//  TMR2IE = 0;

    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    VEL1AC = 0;             // Zera aÁıes
    VEL2AC = 0;
    V1OK = 0;
    V2OK = 0;

    VELD1 = 0;              // Zera velocidade
    VELD2 = 0;
    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
    CANAL2 = RADIO2;
    RC1 = 1;

    while (1){
        if(VEL1AC){
        vel1();
        VEL1AC = 0;
        }
        if(VEL2AC){
        vel2();
        VEL2AC = 0;
        }
```

```
        PORTA = SAIDA;                                  // Atualiza
Port A com mudanÁas

        if(V1OK && V2OK){
        POS1DF = POS1D;
        POS1EF = POS1E;
        POS1D = POS1E = 0;
        PORTC = ((((POS1DF-POS1EF)) > 10) << 1) | (DIR1 << 2);
        V1OK = V2OK = 0;
        }
    }
}
```

A.12

velepwmenc2.c

```
// Nome do programa: velpwmenc2.c
//
// Controle malha aberta 2 canais, 1 PWM
//
// OBS ATUAIS:
// Melhorando leitura de 1 encoder.
//
// OBS ANTIGAS:
// Em processo de adicionar um encoder - Lendo vel + ou -.
// Otimizando programa anterior.
// Tentando fazer para 2 canais
// Tentado tratar com interrupÁ„o - RESULTADO: Funciona muito
melhor.
// Vers„o para PIC 16F874A
// DUAS DIRE«'ES
// Vers„o buscando maior velocidade do programa.
//
// ConfiguraÁ„o:
//
// PIC: PIC16F877
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define RADIO1 RB7
#define RADIO2 RB6
#define ENC1C1 RB5
#define ENC2C1 RB4
#define ENC1C2 RB3
#define ENC2C2 RB2


char SAIDA;                              // Port A
bit VEL1AC, VEL2AC, AMOST;               // Bits de aÁ„o
bit CANAL1,CANAL2;                       // Armazena o estado dos
canais
char VELD1H,VELD1L,VELD2H,VELD2L;        // Vel. desejadas High e
Low
unsigned int VELD1,VELD2;                // Vel. desejadas completo
bit IN1,IN2;                             // Guarda valor das
entradas dos encoders
bit DIR1,DIR2;                           // DireÁ„o do encoder
```

```
char POS1D, POS1E, POS2D, POS2E;        // PosiÁ„o do encoder
char POS1DF, POS1EF, POS2DF, POS2EF;    // PosiÁ„o do encoder -
buffer
bit AUX1, AUX2;                         // Vari·veis auxiliares
unsigned char POS1, POS1F;

void interrupt int_isr(void){       // Trata interrupÁıes
    AUX1 = ENC1C2;
    AUX2 = ENC2C2;

    if(RBIF){

        if(ENC1C1 & !IN1){          // Trata Encoder 1
            DIR1 = AUX1;
            POS1 ++;
        }
/*
        if(ENC2C1 & !IN2){          // Trata Encoder 1
            DIR2 = AUX2;
            if (DIR2){
                POS2D++;
            }
                else{
                POS2E++;
            }
        }
*/
        IN1 = ENC1C1;
//      IN2 = ENC2C1;

        if ( RADIO1 != CANAL1   ){                  // Verifica
mudanÁa de estado em RB0
            if( CANAL1 ){                           // Caso o
sinal tenha decido
                VELD1H = TMR1H;
                VELD1L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            CANAL1 = RADIO1;                        // Atualiza
estado de RB0 em CANAL1
            VEL1AC = 1;
        }

        if ( RADIO2 != CANAL2   ){                      //
Verifica mudanÁa de estado em RB0
```

```
            if( CANAL2 ){                               // Caso o
sinal tenha decido
                VELD2H = TMR1H;
                VELD2L = TMR1L;
            }
            else{                                       // Caso tenha
subido
                TMR1ON = 0;                             // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            CANAL2 = RADIO2;                            // Atualiza
estado de RB0 em CANAL1
            VEL2AC = 1;
        }
        RBIF = 0;
    }

    if (TMR0IF) {

        TMR0 = 0b01100011;
        POS1F = POS1;
        POS1 = 0;
        AMOST = 1;
        TMR0IF = 0;

    }

}

void vel1(){
    VELD1 = ( VELD1H << 8 ) + VELD1L;       // Calcula velocidade
em cima do timer 1
    if(VELD1 >= 1550){
//      CCPR1L = (VELD1 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000001;
        SAIDA = SAIDA & 0b11111101;
    }
    else{
        if(VELD1 <= 1450){
//          CCPR1L = (1500 - VELD1) >> 1;
            SAIDA = SAIDA | 0b00000010;
            SAIDA = SAIDA & 0b11111110;
        }
        else{
//          CCPR1L = 0x00;
        }
    }
}
```

```
void vel2(){
    VELD2 = ( VELD2H << 8 ) + VELD2L;       // Calcula velocidade
em cima do timer 1
    if(VELD2 >= 1550){
//      CCPR2L = (VELD2 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000100;
        SAIDA = SAIDA & 0b11111111;
    }
    else{
        if(VELD2 <= 1450){
//          CCPR2L = (1500 - VELD2) >> 1;
            SAIDA = SAIDA | 0b00001000;
            SAIDA = SAIDA & 0b11011011;
        }
        else{
//          CCPR2L = 0x00;
        }
    }
}


void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    GIE = 1;                // Liga interrupÁ„o global
    RBIE = 1;               // Liga interrupÁ„o da porta B
    RBIF = 0;               // Zera flag de int. da porta B

//  PR2 = 0x4F;             // PerÌodo PWM
//  CCPR1L = 0x00;          // Tamanho PWM - High Bits
//  CCPR2L = 0x00;          // Tamanho PWM - High Bits
//  CCP1CON = 0b00001100;   // Configura PWM
//  CCP2CON = 0b00001100;   // Configura PWM

//  T2CON = 0b11111101;     // ConfiguraÁ„o Timer 2
//  TMR2IE = 0;

    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    PS2 = PS1 = 1;          // Configura Timer 0
    PS0 = 0;
    PSA = 0;
    T0CS = 0;
    TMR0IE = 1;
    TMR0IF = 0;

    VEL1AC = 0;             // Zera aÁıes
    VEL2AC = 0;
```

```
    VELD1 = 0;              // Zera velocidade
    VELD2 = 0;

    AUX1 = 0;
    AUX2 = 0;

    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
    CANAL2 = RADIO2;
    RC1 = 1;

    while (1){
        if(VEL1AC){
            vel1();
            VEL1AC = 0;
        }
        if(VEL2AC){
            vel2();
            VEL2AC = 0;
        }
        if(AMOST){
            PORTC = ((POS1F > 100) <<  1) | (DIR1 << 2);
            AMOST = 0;
        }
        PORTA = SAIDA;                              // Atualiza
Port A com mudanÁas
    }
}
```

A.13

velepwmenc3.c

```
// Nome do programa: velpwmenc3.c
//
// Controle malha aberta 2 canais, 1 PWM
//
// OBS ATUAIS:
// Tentando PID
//
// OBS ANTIGAS:
// Melhorando leitura de 1 encoder.
// Em processo de adicionar um encoder - Lendo vel + ou -.
// Otimizando programa anterior.
// Tentando fazer para 2 canais
// Tentado tratar com interrupÁ„o - RESULTADO: Funciona muito
melhor.
// Vers„o para PIC 16F874A
// DUAS DIRE«'ES
// Vers„o buscando maior velocidade do programa.
//
// ConfiguraÁ„o:
//
// PIC: PIC16F877
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define RADIO1 RB7
#define RADIO2 RB6
#define ENC1C1 RB5
#define ENC2C1 RB4
#define ENC1C2 RB3
#define ENC2C2 RB2


char SAIDA;                             // Port A
bit VEL1AC, VEL2AC, AMOST;              // Bits de aÁ„o
bit CANAL1,CANAL2;                      // Armazena o estado dos
canais
char VELD1H,VELD1L,VELD2H,VELD2L;       // Vel. desejadas High e
Low
unsigned int VELD1,VELD2;               // Vel. desejadas completo
signed int VELDF1,VELDF2;               // Vel. desejadas final 0
a 60
```

```
bit IN1,IN2;                            // Guarda valor das
entradas dos encoders
bit DIR1,DIR2;                          // DireÁ„o do encoder
bit AUX1, AUX2;                         // Vari·veis auxiliares
unsigned char POS1, POS1F, POS2, POS2F; // PosiÁ„o dos encoders
signed int PIDOUT1;
bit PIDDIR1;
signed int POS1PID;

void interrupt int_isr(void){       // Trata interrupÁıes
    AUX1 = ENC1C2;
    AUX2 = ENC2C2;

    if(RBIF){

        if(ENC1C1 & !IN1){          // Trata Encoder 1
            DIR1 = AUX1;
            POS1++;
        }

        if(ENC2C1 & !IN2){          // Trata Encoder 1
            DIR2 = AUX2;
            POS2++;
        }
        if ( RADIO1 != CANAL1   ){                  // Verifica
mudanÁa de estado em RBO
            if( CANAL1 ){                           // Caso o
sinal tenha decido
                VELD1H = TMR1H;
                VELD1L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            VEL1AC = 1;
        }

        if ( RADIO2 != CANAL2   ){                          //
Verifica mudanÁa de estado em RBO
            if( CANAL2 ){                           // Caso o
sinal tenha decido
                VELD2H = TMR1H;
                VELD2L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
```

```
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            VEL2AC = 1;
        }

        IN1 = ENC1C1;
        IN2 = ENC2C1;
        CANAL1 = RADIO1;                        // Atualiza estado
de RB0 em CANAL1
        CANAL2 = RADIO2;                        // Atualiza estado
de RB0 em CANAL1
        RBIF = 0;
    }

    if (TMR0IF) {

        TMR0 = 0b01100011;
        POS1F = POS1;
        POS2F = POS2;
        POS1 = POS2 = 0;
        AMOST = 1;
        TMR0IF = 0;

    }

}

void vel1(){
    VELD1 = ( VELD1H << 8 ) + VELD1L;       // Calcula velocidade
em cima do timer 1
    if(VELD1 >= 1550){
        VELDF1 = (VELD1 - 1500) >> 2;
    }
    else{
        if(VELD1 <= 1450){
            VELDF1 = - ((1500 - VELD1) >> 2);
        }
        else{
            VELDF1 = 0;
        }
    }
}

void vel2(){
    VELD2 = ( VELD2H << 8 ) + VELD2L;       // Calcula velocidade
em cima do timer 1
    if(VELD2 >= 1550){
        CCPR2L = (VELD2 - 1500) >> 1;
        SAIDA = SAIDA | 0b00000100;
```

```
            SAIDA = SAIDA & 0b11111111;
        }
        else{
            if(VELD2 <= 1450){
                CCPR2L = (1500 - VELD2) >> 1;
                SAIDA = SAIDA | 0b00001000;
                SAIDA = SAIDA & 0b11011011;
            }
            else{
                CCPR2L = 0x00;
            }
        }
}


void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    GIE = 1;                // Liga interrupÁ„o global
    RBIE = 1;               // Liga interrupÁ„o da porta B
    RBIF = 0;               // Zera flag de int. da porta B

    PR2 = 0x4F;             // PerÌodo PWM
    CCPR1L = 0x00;          // Tamanho PWM - High Bits
    CCPR2L = 0x00;          // Tamanho PWM - High Bits
    CCP1CON = 0b00001100;   // Configura PWM
    CCP2CON = 0b00001100;   // Configura PWM

    T2CON = 0b11111110;     // ConfiguraÁ„o Timer 2
    TMR2IE = 0;

    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    PS2 = PS0 = 1;          // Configura Timer 0
    PS1 = 0;
    PSA = 0;
    T0CS = 0;
    TMR0IE = 1;
    TMR0IF = 0;

    VEL1AC = 0;             // Zera aÁıes
    VEL2AC = 0;

    VELD1 = 0;              // Zera velocidade
    VELD2 = 0;

    AUX1 = 0;
    AUX2 = 0;
```

```
    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
    CANAL2 = RADIO2;

    while (1){
        if(VEL1AC){
            vel1();
            VEL1AC = 0;
        }
        if(VEL2AC){
            vel2();
            VEL2AC = 0;
        }
        if(AMOST){
            if (DIR1){
                POS1PID = (POS1F<<2);
            }
            else{
                POS1PID = - (POS1F<<2);
            }
            PIDOUT1 = (VELDF1 - POS1PID) + PIDOUT1;
            if(PIDOUT1<0){
                if(PIDOUT1<-120){
                    PIDOUT1 = -120;
                }
                CCPR1L = -PIDOUT1;
                PIDOUT1 = - (CCPR1L >> 1);
                SAIDA = SAIDA | 0b00000001;
                SAIDA = SAIDA & 0b11111101;
            }
            else{
                if(PIDOUT1>120){
                    PIDOUT1 = 120;
                }
                CCPR1L = PIDOUT1;
                PIDOUT1 = PIDOUT1 >> 1;
                PIDDIR1 = 1;
                SAIDA = SAIDA | 0b00000010;
                SAIDA = SAIDA & 0b11111110;
            }
            AMOST = 0;
            PORTA = SAIDA;                                  //
Atualiza Port A com mudanÁas
        }
    }
}
```

A.14

velepwmenc4.c

```
// Nome do programa: velpwmenc4.c
//
// Controle malha aberta 2 canais, 1 PWM
//
// OBS ATUAIS:
// PID p 2 encoders
//
// OBS ANTIGAS:
// Tentando PID
// Melhorando leitura de 1 encoder.
// Em processo de adicionar um encoder - Lendo vel + ou -.
// Otimizando programa anterior.
// Tentando fazer para 2 canais
// Tentado tratar com interrupÁ„o - RESULTADO: Funciona muito
melhor.
// Vers„o para PIC 16F874A
// DUAS DIRE«'ES
// Vers„o buscando maior velocidade do programa.
//
// ConfiguraÁ„o:
//
// PIC: PIC16F877
// Oscillator: XT
// Watchdog: Off
// Power Up Timer: OFF
// Brown Out Detect: Disabled
// Master Clear Enable: Disabled
// Low Voltage Program: Disabled
// Data EE Read Protect: Disabled
// Code Protect: Off

#include <pic.h>
#define RADIO1 RB7
#define RADIO2 RB6
#define ENC1C1 RB5
#define ENC2C1 RB4
#define ENC1C2 RB3
#define ENC2C2 RB2


char SAIDA;                             // Port A
bit VEL1AC, VEL2AC, AMOST;              // Bits de aÁ„o
bit CANAL1,CANAL2;                      // Armazena o estado dos
canais
char VELD1H,VELD1L,VELD2H,VELD2L;       // Vel. desejadas High e
Low
unsigned int VELD1,VELD2;               // Vel. desejadas completo
signed int VELDF1,VELDF2;               // Vel. desejadas final 0
```

```
a 60
bit IN1,IN2;                            // Guarda valor das
entradas dos encoders
bit DIR1,DIR2;                          // DireÁ„o do encoder
bit AUX1, AUX2;                         // Vari·veis auxiliares
unsigned char POS1, POS1F, POS2, POS2F; // PosiÁ„o dos encoders
signed int PIDOUT1, PIDOUT2;
signed int POS1PID, POS2PID;

void interrupt int_isr(void){       // Trata interrupÁıes
    AUX1 = ENC1C2;
    AUX2 = ENC2C2;

    if(RBIF){

        if(ENC1C1 & !IN1){          // Trata Encoder 1
            DIR1 = AUX1;
            POS1++;
        }

        if(ENC2C1 & !IN2){          // Trata Encoder 1
            DIR2 = AUX2;
            POS2++;
        }
        if ( RADIO1 != CANAL1   ){                  // Verifica
mudanÁa de estado em RBO
            if( CANAL1 ){                           // Caso o
sinal tenha decido
                VELD1H = TMR1H;
                VELD1L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            VEL1AC = 1;
        }

        if ( RADIO2 != CANAL2   ){                          //
Verifica mudanÁa de estado em RBO
            if( CANAL2 ){                           // Caso o
sinal tenha decido
                VELD2H = TMR1H;
                VELD2L = TMR1L;
            }
            else{                                   // Caso tenha
subido
                TMR1ON = 0;                         // Reseta
```

```
timer 1
                TMR1H = 0x00;
                TMR1L = 0x00;
                TMR1ON = 1;
            }
            VEL2AC = 1;
        }

        IN1 = ENC1C1;
        IN2 = ENC2C1;
        CANAL1 = RADIO1;                        // Atualiza estado
de RB0 em CANAL1
        CANAL2 = RADIO2;                        // Atualiza estado
de RB0 em CANAL1
        RBIF = 0;
    }

    if (TMR0IF) {

        TMR0 = 0b01100011;
        POS1F = POS1;
        POS2F = POS2;
        POS1 = POS2 = 0;
        AMOST = 1;
        TMR0IF = 0;

    }

}

void vel1(){
    VELD1 = ( VELD1H << 8 ) + VELD1L;       // Calcula velocidade
em cima do timer 1
    if(VELD1 >= 1550){
        VELDF1 = (VELD1 - 1500) >> 2;
    }
    else{
        if(VELD1 <= 1450){
            VELDF1 = - ((1500 - VELD1) >> 2);
        }
        else{
            VELDF1 = 0;
        }
    }
}

void vel2(){
    VELD2 = ( VELD2H << 8 ) + VELD2L;       // Calcula velocidade
em cima do timer 1
    if(VELD2 >= 1550){
        VELDF2 = (VELD2 - 1500) >> 2;
    }
```

```
    else{
        if(VELD2 <= 1450){
            VELDF2 = - ((1500 - VELD2) >> 2);
        }
        else{
            VELDF2 = 0;
        }
    }
}


void main(void){

    TRISA = 0x00;           // PORTA = saÌda
    TRISB = 0xFF;           // PORTB = entrada
    TRISC = 0x00;           // PORTC = saÌda

    GIE = 1;                // Liga interrupÁ„o global
    RBIE = 1;               // Liga interrupÁ„o da porta B
    RBIF = 0;               // Zera flag de int. da porta B

    PR2 = 0x4F;             // PerÌodo PWM
    CCPR1L = 0x00;          // Tamanho PWM - High Bits
    CCPR2L = 0x00;          // Tamanho PWM - High Bits
    CCP1CON = 0b00001100;   // Configura PWM
    CCP2CON = 0b00001100;   // Configura PWM

    T2CON = 0b11111110;     // ConfiguraÁ„o Timer 2
    TMR2IE = 0;

    T1CON = 0x00;           // Configura Timer 1
    TMR1ON = 1;             // Liga Timer 1

    PS2 = PS0 = 1;          // Configura Timer 0
    PS1 = 0;
    PSA = 0;
    T0CS = 0;
    TMR0IE = 1;
    TMR0IF = 0;

    VEL1AC = 0;             // Zera aÁıes
    VEL2AC = 0;

    VELD1 = 0;              // Zera velocidade
    VELD2 = 0;

    AUX1 = 0;
    AUX2 = 0;

    SAIDA = 0b00000000;     // Zera saÌda inicial
    PORTA = 0x00;
    CANAL1 = RADIO1;
```

```
CANAL2 = RADIO2;

while (1){
    if(VEL1AC){
        vel1();
        VEL1AC = 0;
    }
    if(VEL2AC){
        vel2();
        VEL2AC = 0;
    }
    if(AMOST){
        if (DIR1){
            POS1PID = (POS1F<<2);
        }
        else{
            POS1PID = - (POS1F<<2);
        }
        PIDOUT1 = (VELDF1 - POS1PID) + PIDOUT1;
        if(PIDOUT1<0){
            if(PIDOUT1<-120){
                PIDOUT1 = -120;
            }
            CCPR1L = -PIDOUT1;
            PIDOUT1 = - (CCPR1L >> 1);
            SAIDA = SAIDA | 0b00000001;
            SAIDA = SAIDA & 0b11111101;
        }
        else{
            if(PIDOUT1>120){
                PIDOUT1 = 120;
            }
            CCPR1L = PIDOUT1;
            PIDOUT1 = PIDOUT1 >> 1;
            SAIDA = SAIDA | 0b00000010;
            SAIDA = SAIDA & 0b11111110;
        }

        if (DIR2){
            POS2PID = (POS2F<<2);
        }
        else{
            POS2PID = - (POS2F<<2);
        }
        PIDOUT2 = (VELDF2 - POS2PID) + PIDOUT2;
        if(PIDOUT2<0){
            if(PIDOUT2<-120){
                PIDOUT2 = -120;
            }
            CCPR2L = -PIDOUT2;
            PIDOUT2 = - (CCPR2L >> 1);
            SAIDA = SAIDA | 0b00000100;
```

```
                SAIDA = SAIDA & 0b11110111;
            }
            else{
                if(PIDOUT2>120){
                    PIDOUT2 = 120;
                }
                CCPR2L = PIDOUT2;
                PIDOUT2 = PIDOUT2 >> 1;
                SAIDA = SAIDA | 0b00001000;
                SAIDA = SAIDA & 0b11111011;
            }


            AMOST = 0;
            PORTA = SAIDA;                                  //
Atualiza Port A com mudanÁas
        }
    }
}
```