Centro Universitário da FEI

Relatório de Iniciação Científica

DESENVOLVIMENTO DE UM NOVO PROTÓTIPO PARA O TIME DE FUTEBOL DE ROBÔS DA FEI – RoboFEI

Relatório Final

Bolsista: Murilo Fernandes Martins

e-mail: murilo_fm@yahoo.com.br

Orientador: Flavio Tonidandel

e-mail: flaviot@fei.edu.br

Departamento: engenharia elétrica

CAPÍTULO 1 ..... **3**
1.1 INTRODUÇÃO **3**
1.1.1 HISTÓRIA DO FUTEBOL DE ROBÔS **3**
1.1.2 A EQUIPE RoboFEI **5**
1.1.2.1 COMO TUDO COMEÇOU **5**
1.1.2.2 A ESTRUTURA DA EQUIPE **6**
1.2 OBJETIVOS **8**
1.3 JUSTIFICATIVA **8**
1.4 METODOLOGIA **8**
1.5 CRONOGRAMA **10**

CAPÍTULO 2 ..... **11**
REVISÃO BIBLIOGRÁFICA **11**

CAPÍTULO 3 ..... **20**
3.1 ROBÔS PARA JOGAR FUTEBOL **20**
3.2 O PROTÓTIPO DESENVOLVIDO **21**
3.2.1 MICROCONTROLADOR E SOFTWARE **22**
3.2.1.1 O PROTOCOLO DE COMUNICAÇÃO **23**
3.2.1.2 RECEPÇÃO E CHECAGEM DOS DADOS RECEBIDOS **26**
3.2.1.3 ATRIBUIÇÃO DAS VELOCIDADES E GERAÇÃO DOS SINAIS PWM **30**
3.2.2 BATERIAS **41**
3.2.3 MOTORES **42**
3.2.4 CONTROLE DE VELOCIDADE DOS MOTORES **42**
3.2.5 CONTROLE DE SENTIDO DE ROTAÇÃO E ETAPA DE POTÊNCIA **44**
3.2.6 ENCODERS **46**
3.2.7 O CIRCUITO ELÉTRICO **48**

CAPÍTULO 4 ..... **50**
CONCLUSÃO **50**

CAPÍTULO 5 ..... **52**
REFERÊNCIAS BIBLIOGRÁFICAS **52**

ANEXO 1 ..... **55**
CÓDIGO-FONTE DO PIC16F876A **55**

CAPÍTULO 1

1.1 INTRODUÇÃO

1.1.1 HISTÓRIA DO FUTEBOL DE ROBÔS

A idéia de robôs jogando futebol foi mencionada pela primeira vez pelo professor *Alan Mackworth*[1] (*University of British Columbia*, Canadá) em um artigo intitulado "*On Seeing Robots*"[2], apresentado no *Vision Interface'92* (*VI'92*) e posteriormente publicado em um livro chamado *Computer Vision: System, Theory, and Applications*[3].

Paralelamente, um grupo de pesquisadores japoneses organizou um *Workshop* em Grandes Desafios para a Inteligência Artificial, em Outubro de 1992, Tókio, discutindo e propondo problemas que representavam grandes desafios. Esse Workshop os levou a sérias discussões sobre usar um jogo de futebol para promover ciência e tecnologia. Estudos foram feitos para analisar a viabilidade prática dessa idéia. Os resultados desse estudo mostraram que a idéia era viável, desejável e englobava diversas aplicações práticas. Em 1993 um grupo de pesquisadores incluindo *Minoru Asada*, *Yasuo Kuniyoshi* e *Hiroaki Kitano*[4], lançaram uma competição robótica chamada de *Robot J-League* (fazendo uma analogia à *J-League*, que é o nome da Liga Japonesa de Futebol Profissional). Em um mês vários pesquisadores já se pronunciavam dizendo que a iniciativa devia ser estendida ao âmbito internacional. Surgia, então, a *Robot World Cup Initiative* (*RoboCup*).

O ano de 1997 é lembrado como um marco na história da Robótica e Inteligência Artificial. Em maio de 1997 o supercomputador da IBM, o *DeepBlue*[5], derrotava *Garry Kasparov*, o humano campeão mundial de xadrez. Ainda nesse ano, a missão *Pathfinder*[6],

da *NASA* obteve sucesso com a sonda *Sojourner*, primeiro sistema robótico autônomo para exploração da superfície de Marte.

O Futebol de Robôs posicionava-se como o novo desafio para os pesquisadores de Inteligência Artificial. A *RoboCup* dava seus primeiros passos em busca do desenvolvimento de robôs jogadores de futebol capazes de enfrentar a seleção humana campeã da Copa do Mundo de 2050, que é a meta, o principal objetivo da *RoboCup* nos dias de hoje. Atualmente a *RoboCup* possui diversas categorias, sendo a Categoria *Small F180* uma das duas categorias das quais a FEI participa em competições.

Na Coréia o Futebol de Robôs surgiu em 1995, pela idéia do prof. coreano *Jong-Hwan Kim*. A primeira competição internacional foi realizada no ano seguinte, 1996, em *Daejeon*, Coréia. Em 1997, durante o Torneio *Micro-Robot Soccer'97* (*MiroSot'97*), foi criada a *FIRA* (*Federation of International Robot-soccer Association*)[7]. Os campeonatos organizados pela *FIRA* acontecem todo ano e atingem uma escala global. Paralelamente ao campeonato acontece o *FIRA Robot World Congress*, onde as equipes submetem e apresentam artigos sobre seus robôs e seus sistemas com o intuito de compartilhar a experiência e tecnologia adquirida em suas pesquisas. Os objetivos principais da FIRA são o de promover o desenvolvimento de sistemas robóticos autônomos multi-agentes e juntar as habilidades e conhecimentos de pesquisadores e estudantes de diversas áreas, como robótica, inteligência artificial, processamento de imagens, telecomunicações, entre outros, em um novo e crescente campo interdisciplinar de robótica autônoma inteligente para uma partida de futebol.

Entre as categorias da *FIRA* está a *Mirosot*[8], pela qual a FEI também compete e é baseado nas regras e limitações impostas por essa categoria que esse projeto foi desenvolvido.

No Brasil, em 1998 aconteceu a 1ª Copa Brasil de Futebol de robôs, na Escola Politécnica da USP, contando com a participação de 6 times de universidades e institutos de pesquisa brasileiros. Em 1998, ainda, o Time Guaraná[9], formado por integrantes da POLI-USP e UNESP, participaram do campeonato mundial organizado pela *FIRA*, na França, conquistando o vice-campeonato na categoria *Mirosot*. Já em 1999, aconteceu o *FIRA CUP BRAZIL'99*[10] no Colégio Notre Dame, em Campinas. Estavam presentes mais de 30 equipes de 7 países de 4 continentes, para participar de campeonatos nas categorias *Narosot* e *Mirosot*, *Robosot* e *Kheperasot*.

1.1.2 A EQUIPE RoboFEI

1.1.2.1 COMO TUDO COMEÇOU

No início do ano de 2003 o prof. Reinaldo Bianchi, o qual participou do desenvolvimento dos times FUTEPOLI[11] e Guaraná, iniciava um grupo de pesquisas sobre Futebol de Robôs a nível de simulação. Em setembro do mesmo ano aconteceria o 6º SBAI (Simpósio Brasileiro de Automação Inteligente) e, paralelamente, a *Second IEEE Student Robotics Competition*, que incluía a categoria de Futebol de Robôs *Mirosot*. Decidiu-se, então, criar o Time de Futebol de Robôs da FEI – RoboFEI, o qual contava com a orientação dos professores Reinaldo Bianchi e Flavio Tonidandel.

O resultado muito satisfatório na competição, um vice-campeonato, assim como a grande repercussão sobre a pesquisa e o desenvolvimento para se chegar ao resultado final alcançado foram determinantes para o crescimento do projeto. Sendo assim, para 2004 traçaram-se metas para a preparação de uma nova versão do time para a competição

nacional que aconteceria em Agosto de 2004, em Salvador, juntamente com o Simpósio Brasileiro de Computação.

Com o objetivo alcançado e o excelente resultado da conquista do campeonato em Salvador novas metas foram impostas à equipe visando dois competições que irão acontecer no ano de 2005. Em maio de 2005 acontecerá um encontro de equipes previsto no cronograma da 7ª Semana da Engenharia Mecatrônica – Automática 2005[12], organizada pela Mecatron. Em setembro de 2005 acontecerá a Competição Latino-americana de Robótica[13] juntamente com o VII SBAI/II LARS[14], que acontecerá em São Luís, Maranhão, organizada pela Universidade Federal do Maranhão – UFMA.

1.1.2.2 A ESTRUTURA DA EQUIPE

A plataforma do projeto é composta por um campo para o jogo, com as medidas de 150 x 90 cm, uma câmera de vídeo para cada time posicionada acima e centralizada em relação ao campo, seu respectivo sistema de aquisição de imagens, um computador para cada time, um sistema de transmissão de dados por radiofreqüência e 3 robôs para cada time, os quais não devem exceder o tamanho máximo de um cubo com aresta de 7,5 cm, além de uma bola de golfe de cor laranja, todas essas características de acordo com as regras impostas pela *FIRA* para a categoria *Mirosot*.

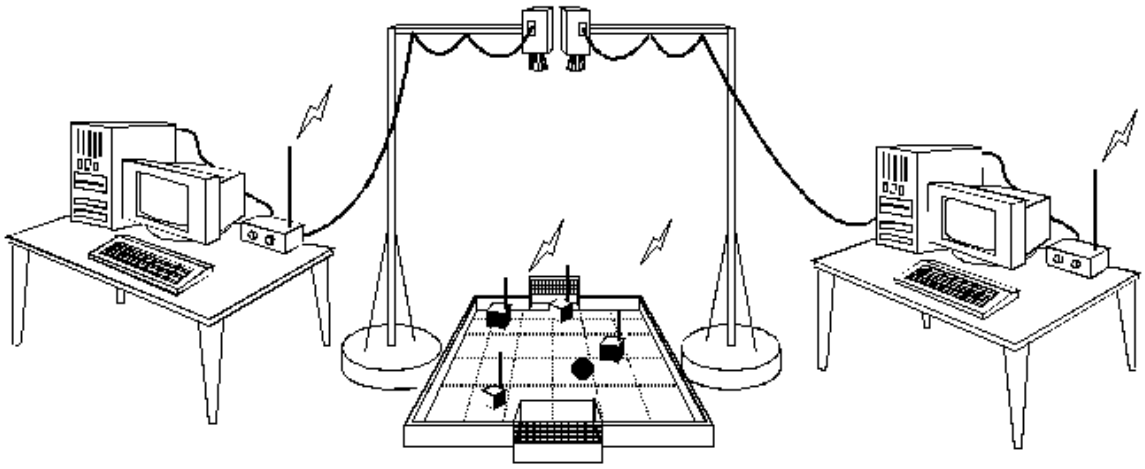

Figura 1 – Estrutura proposta pela FIRA

A Equipe RoboFEI possui o campo para o jogo obedecendo as normas da *FIRA*, assim como 2 bolas de golfe de cor laranja. Possui também uma câmera com sensor tipo CCD e 4 computadores rodando o sistema operacional linux, dos quais 3 possuem a placa de aquisição de imagens. Dois transmissores de radiofreqüência, cada qual com a opção de configuração entre 2 canais de freqüência fazem a comunicação dos computadores com os robôs, os quais são 8 e foram construídos utilizando o mesmo projeto do Time Guaraná. Cada robô possui um receptor configurável para 2 freqüências de funcionamento, um jumper de configuração de endereço do robô, 2 motores de corrente contínua, sendo um para cada roda e um microcontrolador. O software, desenvolvido em linguagem de programação C++, residente nos computadores é encarregado de capturar a imagem da câmera, analisar essa imagem e determinar as posições dos robôs e da bola, para a partir daí determinar a ação conjunta do time determinando a movimentação de cada um dos 3 robôs que compõem o time. Ao determinar tais movimentações, as velocidades dos pares de motores de cada robô são definidas e as ordens de controle são, então, enviadas por radiofreqüência para os robôs.

1.2 OBJETIVOS

O objetivo do presente trabalho é desenvolver hardware e software para o novo robô para o Time de Futebol de Robôs da FEI – RoboFEI.

1.3 JUSTIFICATIVA

Após dois anos utilizando a mesma plataforma de hardware, surgiu a necessidade de robôs dotados de determinadas características, as quais não comprometessem o funcionamento do sistema como um todo, dadas as metas a serem atingidas para o ano de 2005. Necessita-se de robôs mais velozes, suaves e precisos em seus movimentos, robôs que sejam capazes de proteger a bola, que não derrapem, que tenham uma estrutura mais robusta, capaz de agüentar choques e suportar um maior tempo de funcionamento ininterrupto. O presente projeto baseou-se nas melhorias desejáveis para a pesquisa e desenvolvimento do protótipo para o novo modelo de robô.

1.4 METODOLOGIA

Primeiramente foram feitos um estudo e análise de hardware e software do robô atual, levantando dados como circuito elétrico, arquitetura do microcontrolador, tipo de motores utilizado, tipo de controle dos motores, modelo de realimentação para a odometria, relação de redução das engrenagens do sistema mecânico, entre outros, para que fossem encontradas as fontes das características limitantes do robô. Em seguida foi analisado o

atual sistema de comunicação de dados para a obtenção de informações como a taxa de transferência de dados, freqüência de funcionamento, imunidade a ruído.

Com as informações relevantes do sistema de hardware atual iniciou-se uma pesquisa sobre como outras equipes robôs, microcontroladores e sistemas de controle mais utilizados, motores mais adequados para o Futebol de Robôs, entre outras características. Tendo as informações sobre o sistema atual e com dados pesquisados sobre como outras equipes têm desenvolvimento robôs para essa finalidade, definiu-se a especificação para o novo robô. Definiu-se o microcontrolador mais adequado ao projeto, preocupando-se com seu custo, funcionalidades, ambiente e facilidade de desenvolvimento. O estudo sobre motores elétricos foi determinante na escolha dos mesmos, visto que, depois de feita uma pesquisa de mercado, os modelos mais utilizados pelas equipes consultadas não estavam disponíveis no Brasil ou seu custo inviabilizava sua escolha. Então uma nova pesquisa por motores com determinadas características foi feita, até que fosse encontrado um modelo compatível com as especificações de projeto.

Definidos microcontrolador e motores, foi necessário adquirir conhecimento sobre as várias maneiras de se controlar motores elétricos de corrente contínua. Após analisadas as possibilidades, foi escolhida a maneira mais adequada para acionamento e controle dos motores em questão.

Pôde-se, então, desenvolver o circuito eletrônico do robô, assim como iniciar o desenvolvimento do software do microcontrolador. Primeiramente foi implementado o controle dos motores. Em seguida foi implementado o novo protocolo de comunicação, o qual deve receber dados do computador relativos à velocidade dos motores a serem controlados e atribuir a tais motores essa velocidade. Para concluir o desenvolvimento do

software, a odometria, responsável pela manutenção da real velocidade do motor igual ou muito próxima da velocidade teórica, foi implementada.

O circuito foi montado em protoboard e a comunicação com o computador, a atribuição das velocidades e o acionamento dos motores foram testados. O circuito eletrônico foi passado para uma placa de circuito impresso e um protótipo em Lego foi montado para obter-se os resultados finais do projeto.

1.5 CRONOGRAMA

- estudo e análise de hardware e software do robô atual;
- análise do sistema de transmissão de dados atual;
- pesquisa sobre outras equipes e seus sistemas de hardware;
- análise, pesquisa e definição das características do novo robô;
- estudo sobre microcontroladores e definição da plataforma a ser utilizada;
- estudo e pesquisa sobre motores elétricos;
- estudo e pesquisa sobre acionamento e controle de motores elétricos;
- desenvolvimento do circuito eletrônico do robô;
- desenvolvimento do software do microcontrolador do robô;
- implementação do software e testes em protoboard;
- montagem de protótipo em Lego;
- resultados e conclusões;

CAPÍTULO 2

2.1. REVISÃO BIBLIOGRÁFICA

Robôs móveis são uma categoria de robôs caracterizados por sua mobilidade, sua capacidade de navegação por ambientes diversos. Diferentemente de braços robóticos e outras categorias de robôs, que são instalados em um local fixo, um ambiente estático, onde a maioria dos acontecimentos futuros são previsíveis e não são capazes de se movimentar além de seus limites, não são capazes de alcançar pontos fora de sua área de trabalho, robôs móveis são capazes de se locomover, seja aleatoriamente, ou controlado por algum algoritmo de tomada de decisões. Robôs móveis são utilizados para trabalharem em ambientes desconhecidos, ambientes dinâmicos onde não se tem como prever o estado futuro das variáveis desse ambiente. Esses robôs são utilizados para exploração de planetas, locomover-se, detectar e desativar minas terrestres em um campo de batalha, cortar a grama do jardim, limpar a piscina, entre outras infinitas aplicações, como veículos de combate não tripulados. Esses robôs podem ser controlados remotamente ou podem ser totalmente autônomos, capazes de tomar decisões em situações adversas e até mesmo interagir com outros robôs.

Uma das principais formas de se obter movimento de partes móveis de um equipamento é através de motores CC (corrente contínua)[15]. Robôs móveis têm várias configurações de motores, seja pela quantidade como pela disposição dos mesmos em sua construção, para que se obtenha a mobilidade desejada. A maneira mais comum, a mais empregada em robôs móveis, para se obter a maior mobilidade possível, é o emprego de dois motores CC, sendo um motor para cada roda. Na categoria *Mirosot* todos os times empregam esse tipo de configuração, pois assim consegue-se obter boa mobilidade e boa

precisão no controle dos robôs, que têm tamanho muito reduzido. Geralmente as rodas ficam posicionadas simetricamente, para que o robô possa se deslocar para frente, para trás, efetuar curvas de raios diversos e ainda girar em seu próprio eixo.

Motores CC são máquinas elétricas que convertem energia elétrica em energia mecânica (movimento). Pequenos motores CC são compostos por estator e rotor. Estator é a parte estacionária, um imã fixo, a carcaça do motor. Rotor é a parte que gira, uma bobina, o eixo do motor. É aplicada uma tensão nos contatos dessa bobina, gerando uma corrente elétrica. Essa corrente elétrica cria um campo magnético na bobina e ela passa a se comportar como um imã. Esse campo, cujo sentido depende do sentido da corrente elétrica, que por sua vez depende da polarização da tensão aplicada, interage com o campo do imã fixo de modo que surge uma força que empurra a bobina, forçando-a a girar e encontrar uma posição em que as forças não mais atuem. No entanto, ao girar para encontrar essa posição de equilíbrio, o motor aciona um sistema de comutadores elétricos em seu eixo, os quais são denominados de escovas, que invertem a corrente na bobina e, conseqüentemente, o sentido de seu campo magnético. O resultado é que, com esta inversão, a posição de equilíbrio que estava quase sendo alcançada, desaparece. A nova posição de equilíbrio passa ser meia volta à frente, fazendo com que a bobina tenda a continuar a se mover para alcançá-la. Mais meia volta, a escova entra em ação, invertendo a corrente de modo que a posição de equilíbrio adianta-se, e assim indefinidamente, a bobina seguirá girando permanentemente sem nunca encontrar a posição de equilíbrio. Enquanto houver energia sendo fornecida à bobina, ela irá girar, procurando alcançar a posição de equilíbrio. Os motores CC são especificados pelos fabricantes para operar com uma determinada faixa de tensão. A tensão nominal é a tensão em que, teoricamente, pode-se obter o melhor desempenho do motor. A faixa de tensão dos motores utilizados, por exemplo, é de 4,8V

até 7,2V. Isso significa que a rotação e a corrente drenada pelo motor irão variar de acordo com a tensão aplicada. Os fabricantes também fornecem valores como a corrente drenada pelo motor sem carga, assim como sua velocidade de rotação sem carga. Velocidades de rotação muito próximas à velocidade de rotação sem carga indicam que o motor é capaz de fornecer mais força do que a carga à ele aplicada está exigindo, enquanto velocidades muito baixas indicam que o motor está funcionando sobrecarregado. Conseqüências de sobrecarga são maior consumo de energia, pois a corrente consumida aumenta e mais calor é gerado, ou seja, mais energia é desperdiçada, além de poder danificar o motor.

É necessário que haja, então, um controle de velocidade, assim como um sensor de corrente, a fim de evitar danos permanentes ao motor pelo excesso de corrente drenada, causado pela sobrecarga. Para que a velocidade de rotação de um motor CC varie deve-se variar a tensão aplicada em sua bobina. Existem duas técnicas básicas para que se obtenha essa variação de tensão. O controle linear de velocidade apenas varia a tensão aplicada na bobina do motor, resultando em uma variação de velocidade. Sua implementação e utilização apresentam duas desvantagens. A primeira é que ao variar-se a tensão, conseqüentemente, a potência também irá variar. Para velocidades baixas a potência será menor e o torque máximo do motor não será alcançado, fazendo com que ele não seja capaz de movimentar-se devido à carga aplicada e à baixa potência. Outro inconveniente é que a potência dissipada na forma de calor, pois a etapa de potência que controla a carga funciona como um reostato, assim quando o motor estiver funcionando na faixa média de rotações o circuito de controle irá dissipar tanta potência quanto o próprio motor. Outra forma de controle de velocidade é o PWM (*pulse width modulation*), como descrito em diversos artigos publicados na *Encoder*, *newsletter* da SRS (*Seattle Robotics Society*)[16]. Essa forma de controle permite manter o torque máximo do motor. O sinal PWM é uma onda

quadrada de freqüência fixa em que sua tensão de nível alto é a tensão máxima de acionamento do motor. Para que se tenha uma variação de velocidade, a corrente média aplicada ao motor é variada e essa variação se dá através da variação da duração dos pulsos da onda quadrada. Ao invés de acionar o motor com uma corrente contínua, é aplicada em sua bobina essa onda quadrada, que funciona como um dispositivo que liga e desliga rapidamente, sempre com a tensão máxima de acionamento. No caso dos motores empregados, por exemplo, a tensão máxima de acionamento é de 7,2V, então o motor recebe pulsos de 7,2V de amplitude sempre. A variação da duração desses pulsos caracteriza a variação da velocidade. A tensão média e, conseqüentemente, a corrente média aplicada no motor pode ser obtida pela fórmula:

$$V = \frac{T_{Duty}}{T_{freq}} * V_{máx}$$

A tensão média é representada por V, enquanto $V_{máx}$ é a tensão máxima utilizada no acionamento do motor, T_{duty} é o tempo de duração do pulso e T_{freq} é o período do sinal PWM. A duração do pulso, chamada de ciclo ativo (*duty cicle*), geralmente é expressa em porcentagem. Um inconveniente do controle PWM é que, como seu sinal está comutando rapidamente, a transição rápida de estado do circuito irá gerar transientes, sinais de alta freqüência, gerando interferências eletromagnéticas (*EMI*). Essas interferências podem afetar os sinais de radiofreqüência e até mesmo os circuitos lógicos próximos. Isso pode ser evitado com o emprego de filtros de sinais[17]. Em relação à potência dissipada, o controle PWM dissipa muito menos energia que o controle linear, pois, como seu sinal está sempre comutando, os transistores responsáveis por essa comutação apenas irão dissipar potência, muito pouca potência, nas mudanças de estado, enquanto o controle linear dissipa quase tanta potência quanto o motor.

Um problema a ser resolvido no controle de velocidade de um motor CC é seu acionamento. O gerador do sinal de controle PWM nem sempre é capaz de fornecer tensão e correntes exigidas pelo motor. Faz-se necessária, então, uma etapa de potência, a qual deve ser capaz de fornecer níveis de tensão e corrente exigidos pelo motor, além de outras funcionalidades, como a inversão do sentido de rotação do motor, através da inversão do sentido da corrente. Essa etapa de potência pode ser implementada de duas maneiras. Utilizando relés de contatos reversíveis, capazes de inverter o sentido de circulação da corrente no motor e isolando a tensão lógica de níveis TTL da tensão de acionamento do motor. O inconveniente de utilizar relés é seu tamanho, sua baixa velocidade de chaveamento e sua vida útil, visto que um relé é um dispositivo eletromecânico. Outra maneira é empregar o uso de transistores de potência para fornecer tensão e corrente necessários. Esses transistores são ligados de tal maneira que seu circuito se assemelha à letra "H", sendo muito conhecido como Ponte H. A figura a seguir ilustra o circuito elétrico de uma Ponte H:

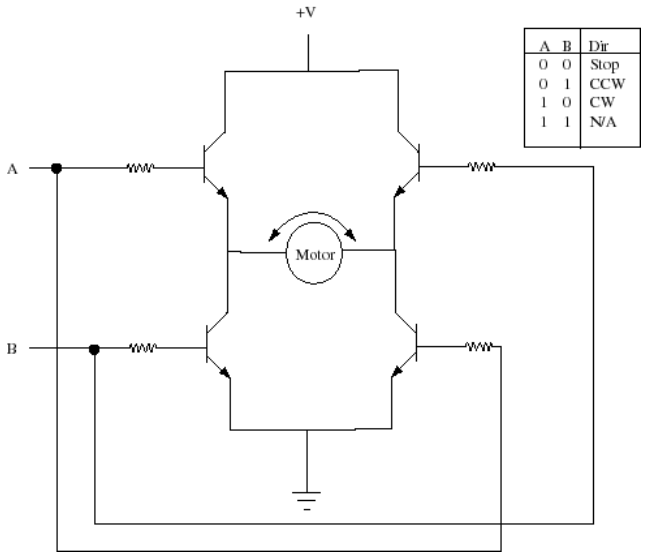

Figura 2 – Ponte H completa

Os transistores funcionam como chaves, ou seja, comutando entre os modos de corte e saturação, dissipando uma potência muito baixa. Os pontos “A” e “B” são as entradas do circuito. Essas entradas irão determinar o sentido da corrente na bobina do motor, cortando ou saturando os transistores que estão ligados em cada entrada. Com isso tem-se uma interface de controle onde se pode controlar o sentido de rotação dos motores utilizando tensões de níveis lógicos TTL, por exemplo, assim como uma etapa de potência capaz de fornecer os níveis de tensão e corrente exigidos pelo motor.

O robô necessita de um circuito de controle, capaz de gerenciar todas as suas funcionalidades, até mesmo gerar os sinais de controle PWM. Para tal, geralmente é empregado um microcontrolador, um dispositivo que possui memórias RAM e ROM internas, oscilador interno de *clock*, I/O interno, contadores/temporizadores, comparadores analógicos, conversores A/D, interface de comunicação serial (USART), módulos geradores de sinais PWM, entre outros periféricos. Esse microcontrolador pode gerar os sinais de controle PWM, que por sua vez são aplicados ao circuito da Ponte H, que fazem a interface de potência com o motor. Mas esse tipo de controle, caracterizado como um controle em malha aberta, nem sempre fornece bons resultados no controle de velocidade, que deve ser preciso. Para alcançar a desejada precisão no controle da velocidade é aplicado um controle em malha fechada[18]. O que caracteriza a malha fechada é que a informação de velocidade real do motor é fornecida ao circuito de controle, geralmente o microcontrolador, realimentando o circuito. Essa informação de velocidade deve estar codificada em um formato capaz de ser interpretado pelo microcontrolador. Para isso existem os *Encoders*[19], que são transdutores de movimento, capazes de converter movimentos lineares ou angulares em informações elétricas que o software do microcontrolador possa interpretar como distância, velocidade, ângulo de posicionamento,

etc. Diversos artigos publicados na *Encoder*, como [20], por exemplo, discorrem sobre o assunto, detalhando características, implementações e construções de *encoders.*

Encoders possuem internamente discos com furos simétricos em toda a sua circunferência que funcionam como máscaras, como mostra a figura 3.

Figura 3 – Disco-encoder

Esse disco fica posicionado entre transmissores e receptores infravermelho. A luz infravermelha do transmissor passa pelo furo e chega ao receptor, que gera um pulso, o qual passa por um circuito elétrico que possui uma histerese para enquadrá-lo (um inversor *Schmitt-Trigger*, por exemplo), gerando assim um trem de pulsos cuja freqüência, ou seja, o número de pulsos em determinado instante de tempo, irá representar a velocidade de rotação do motor, por exemplo. Pode-se utilizar sensores reflexivos e um disco que contenha um reticulado estampado alternando entre claro e escuro, refletindo ou não para o receptor, o sinal do transmissor infravermelho. A resolução dos encoders determina a precisão dos mesmos. Essa resolução é a quantidade de pulsos que ele é capaz de gerar a cada volta completa do seu disco, ou seja, a quantidade de furos presente no disco. A figura 4 mostra alguns exemplos de encoders com diferentes resoluções.

Figura 4 – Discos de encoders com diferentes resoluções

Os *encoders* são empregados em diversos equipamentos e dispositivos, como mouses, por exemplo, onde um par de *encoders* verifica seu deslocamento em 2 eixos no plano em que se movimenta. Em robôs móveis os *encoders* podem fornecer informações de velocidade dos motores, posição, sentido de rotação dos motores. Sua maior utilização em robôs móveis, no entanto, é no controle de velocidade dos motores, para que haja boa precisão nos movimentos do robô.

O Time Guaraná tinha robôs que utilizavam motores CC com tensão nominal de 5V. Um sensor reflexivo infravermelho detectava as informações de rotação através de um reticulado reflexivo instalado em cada roda. Essa informação era interpretada pelo microcontrolador AT89C2051[21], da Atmel. Esse microcontrolador controlava a tensão aplicada nos motores através de duas Pontes H completas, ligadas em seus pinos de I/O, o que permitia um controle de direção dos motores e um controle de velocidade utilizando a técnica PWM. Essa arquitetura é a mesma utilizada atualmente pelos robôs do time da FEI.

Outras equipes, como a Mineirosot[22] e a equipe de futebol de robôs da UFPR[23], utilizaram plataformas semelhantes. A Mineirosot utilizou uma plataforma desenvolvida em seus laboratórios denominada Placa Microcontrolada para Aplicações Robóticas e Sistemas Controlados – *Boc Board*[24], a qual utiliza um microcontrolador da

Microchip[25], um par de motores CC, Pontes H e sinais PWM para acionamento dos motores. Utilizam também um circuito integrado que implementa um filtro PID[26], o circuito integrado LM629[27], da fabricante *National Semiconductor*.

O time AUSTRO, da IHRT (*Institute of Handling Devices and Robotics*), da Universidade de Tecnologia de Vienna, na Áustria[28], utiliza o microcontrolador 89C52[29], da Atmel, como unidade de processamento central, utilizando o integrado LM629 para implementar o filtro PID e Pontes H para a etapa de potência dos motores. Outro time interessante é o Sul-coreano Robotis, da empresa Robotis[30]. Utilizam DSPs, FPGAs e Pontes H para a etapa de potência dos motores. Segundo o projetista do robô, usando o DSP eles conseguem, além de grande rapidez de processamento, uma precisão de 0,02 mm nos movimentos do robô. A estratégia roda metade no computador e metade no robô. É utilizado um sistema de controle adaptativo ao invés de controle PID.

CAPÍTULO 3

3.1 ROBÔS PARA JOGAR FUTEBOL

Um robô basicamente pode ser dividido em 4 partes. O hardware, o software, o sistema de radiofreqüência e a parte mecânica.

O hardware nada mais é do que o circuito elétrico do robô, composto pelas seguintes partes: o microcontrolador, que é o cérebro do robô, responsável pelo controle de todo o restante do circuito; a etapa de potência para os motores, responsável por fornecer tensão e corrente suficientes para o correto funcionamento dos motores, mas sendo controlada por níveis lógicos; os sensores, capazes de fornecer informações importantes, como encoders de velocidade, sensores de corrente dos motores para evitar sua queima, sensores de bateria para indicar a carga das mesmas; baterias, com carga suficiente para alimentar todo o hardware do robô; motores, com bom torque e velocidade de rotação, para que o robô possa se locomover com precisão e rapidamente.

O software é o programa que roda no microcontrolador, responsável por comunicar-se com o sistema de radiofreqüência para receber informações do computador, assim como interpretar as informações recebidas, acionar os motores de acordo com tais informações e ainda é desejável que esse software seja capaz de manter a velocidade dos motores constante, através de uma odometria, cujo valor real da velocidade pode ser obtido através de encoders de velocidade. No software é implementado o protocolo de comunicação de dados e ainda é feita a checagem de erros. É esse software que controla todas as funções do robô, no entanto os movimentos que ele executa são apenas a interpretação das ordens de movimento recebidas pelo sistema de radiofreqüência.

O sistema de radiofreqüência é responsável por estabelecer um link de comunicação de dados serial sem fios entre computador e robôs. Pode-se utilizar diversos métodos para implementação desse tipo de comunicação. Esse sistema deve ser capaz de modular/demodular o sinal, assim como fazer a codificação/decodificação dos dados, além de transmiti-lo em uma determinada freqüência. No entanto o presente projeto não irá tratar do sistema de radiofreqüência.

A parte mecânica representa as rodas, a redução mecânica e o acoplamento dos motores às rodas e as paredes que formam o corpo do robô. Algumas características interessantes são a relação de redução motor-roda, assim como o diâmetro das rodas e a espessura das mesmas. O material empregado na construção deve ser leve. O peso deve ser bem distribuído e não deve haver folgas nem balanços indesejáveis que comprometeriam um controle preciso do robô. O presente projeto também não trata da parte mecânica.

Assim como o sistema de radiofreqüência, a parte mecânica está sendo desenvolvida por outros alunos.

3.2 O PROTÓTIPO DESENVOLVIDO

Hardware e software abrangem muitos módulos existentes no protótipo. O hardware diz respeito à parte física da eletrônica do robô, como os motores, as baterias, os circuitos de acionamento e de potência para os motores, os sensores, encoders, enfim, circuitos integrados, cada qual com sua função, que juntos determinam as características do robô. O software do microcontrolador implementa as funcionalidades do robô, controla e integra os módulos de hardware (circuitos integrados, motores, sensores, etc.). Tanto o software como

os principais módulos de hardware e circuitos integrados utilizados no protótipo são descritos a seguir.

3.2.1 MICROCONTROLADOR E SOFTWARE

O microcontrolador PIC 16F876A[31],[32],[33],[34] da Microchip foi escolhido por satisfazer todas as necessidades de projeto. Ele possui um canal de comunicação serial, dois periféricos chamados CCP (Capture/Compare/PWM) capazes de gerar os sinais de controle PWM para os motores, 8 kbytes de memória flash de programa, 368 bytes de memória de dados e executa instruções em apenas 1 ciclo de máquina, que são 200 ns para o cristal de 20 Mhz utilizado.

O PIC é responsável por receber os dados enviados serialmente pelo computador, checar se houve erros, interpretar tais erros se for o caso, atribuir as velocidades aos motores e executar a odometria para o controle da velocidade dos motores.

Para receber os dados enviados pelo computador foi utilizado o canal de comunicação serial. O canal de comunicação serial foi configurado para funcionar no modo assíncrono, com 8 bits de dados, 1 bit de parada, e taxa de transmissão de 9600 bps.

Um protocolo de comunicação de dados entre computador e robôs foi criado para que os robôs pudessem ser endereçáveis, ou seja, para que, de acordo com o formato dos dados recebidos e com a configuração de endereço de cada robô, os bytes referentes àquele robô pudessem ser identificados.

3.2.1.1 O PROTOCOLO DE COMUNICAÇÃO

O protocolo de comunicação foi desenvolvido pensando em aplicações futuras, sendo bastante flexível em relação à quantidade de bytes a serem transmitidos. As especificações desse protocolo determinam que é necessário um byte inicial de transmissão, seguido dos bytes de dados, sendo finalizado por um byte de fim de transmissão. Os bytes de dados são uma quantidade par, sendo estipulado 6 bytes de dados, 2 bytes de dados para cada robô. Como existe um byte inicial e um byte final, qualquer quantidade de bytes de dados pode ser, teoricamente, transmitida. Observe a formatação do protocolo:

Byte inicial	Byte de dados	Byte de dados	Byte de dados	Byte de dados	Byte de dados	Byte de dados	Byte final

Da mesma maneira, os robôs foram desenvolvidos, a princípio, para serem configurados com 3 endereços diferentes ou um modo teste, mas sendo facilmente modificado para trabalhar com mais endereços, ou ainda, receber mais bytes de dados. Isso torna o projeto expansível para novas aplicações futuras.

Um par de chaves ligadas aos pinos 2 e 3 do microcontrolador determinam o endereço identificador de cada robô pela combinação de suas posições. Ambas desligadas, posição $(00)_2$, ativam o modo teste do robô. As outras 3 possíveis combinações de posição representam, em notação binária, o endereço identificador de cada robô, sendo $(01)_2$ referente ao robô 1, $(10)_2$ referente ao robô 2 e $(11)_2$ referente ao robô 3.

A sub-rotina SETMODE lê os pinos onde as chaves estão ligadas e salva esses valores nos bits menos significativos de uma variável com tamanho de um byte. Essa variável é comparada com o byte 0x00 que corresponde ao modo teste do robô. Caso sejam iguais, o robô está configurado em modo teste e, então, a comunicação serial é desabilitada,

em seguida o programa retorna. Caso não esteja configurado como modo teste, o valor de sua configuração é comparado com a configuração para robô 1. Caso sejam iguais, então o robô é configurado como robô 1 carregando o endereço que aponta para o primeiro byte de dados recebido em uma variável que faz referência ao primeiro byte de dados do robô e o programa retorna ao ponto onde a sub-rotina havia sido chamada. Caso contrário, é testado o modo de configuração para robô 2. Se os valores das variáveis comparadas forem iguais, então o endereço que aponta para o terceiro byte de dados recebido é carregado na variável que faz referência ao primeiro byte de dados para o robô, retornando em seguida. Caso não seja essa a configuração, é testada a configuração para robô 3 e o mesmo procedimento é realizado, assim como na configuração para robô 1 ou para robô 2. Caso nenhuma dessas configurações seja reconhecida na comparação de valores, o programa retorna e o robô não recebe nenhuma configuração, mas essa possibilidade nunca será atingida, visto que todas as configurações de posições possíveis das chaves já foram tratadas. A figura 5 é o fluxograma da sub-rotina SETMODE.

O protocolo pode ser visto, então, da seguinte maneira:

Byte inicial	1º Byte do robô 1	2º Byte do robô 1	1º Byte do robô 2	2º Byte do robô 2	1º Byte do robô 3	2º Byte do robô 3	Byte final

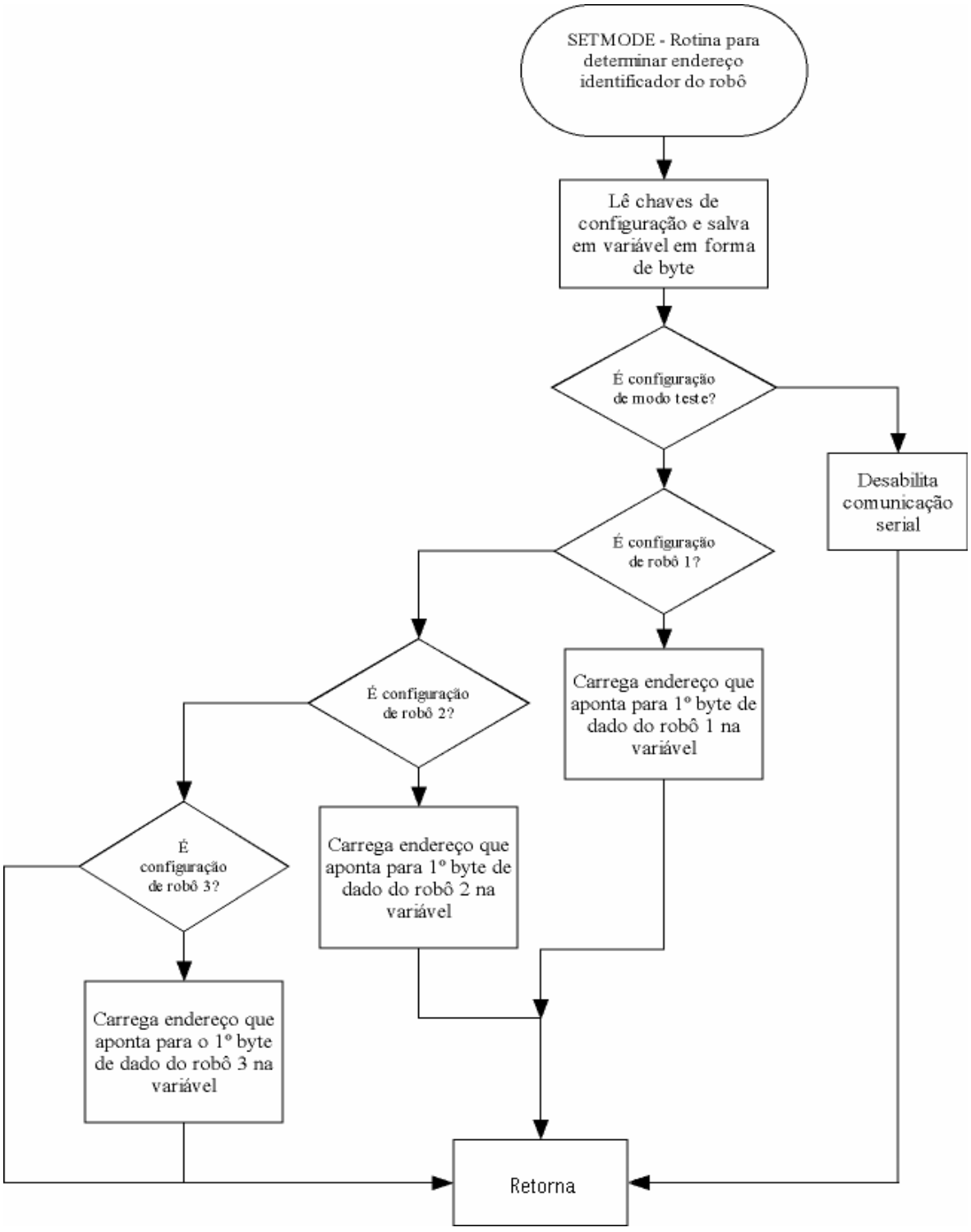

Figura 5 – Fluxograma da sub-rotina SETMODE

3.2.1.2 RECEPÇÃO E CHECAGEM DOS DADOS RECEBIDOS

Com o robô devidamente configurado conforme suas chaves de configuração, já é possível que o robô receba dados através de seu canal serial, obedecendo ao formato do protocolo estabelecido.

Ao ser recebido um byte pelo canal serial uma interrupção é gerada e o programa desvia para tratar essa interrupção. É checado se houve erro na recepção do byte recebido, caso tenha havido erro o byte é ignorado e o programa retorna para o ponto estava no momento que aconteceu a interrupção. Caso não tenha havido erro, então, o byte é salvo em uma variável. O *flag* de recepção de *Start* Byte é testado para verificar se o byte inicial, que corresponde ao caractere ASCII 'b', já foi recebido anteriormente. Caso não tenha sido recebido, então o byte recebido é comparado com o valor do byte inicial e, se forem iguais, então o *flag* de recepção de *Start* Byte é ligado e o programa retorna. Se o *Start* Byte já havia sido recebido, então o programa entende que o byte recebido é dado e verifica se todos os dados já foram recebidos. Caso ainda não tenha recebido todos os dados, o programa salva o byte na memória e retorna. Se todos os bytes de dados já foram recebidos, o programa desliga o *flag* de recepção de *Start* Byte, pois já recebeu uma mensagem completa, mas testa se o byte recebido é o Byte Final, que corresponde ao caractere ASCII 'e'. Se na comparação for constatado que os bytes são iguais, o *flag* indicador de Comando Válido é ligado e o programa retorna. Caso contrário, todos os dados recebidos são ignorados por não se ter recebido o Byte Final.

Então a variável do robô que aponta para seu primeiro byte de dados já contém o endereço para referenciar esse byte, sendo que para acessar o 2º byte de dados é necessário apenas incrementar a variável que aponta para o primeiro byte de dados. A figura 6 ilustra a

recepção de um byte pelo canal serial, quando é gerada a interrupção de recebimento de dados.

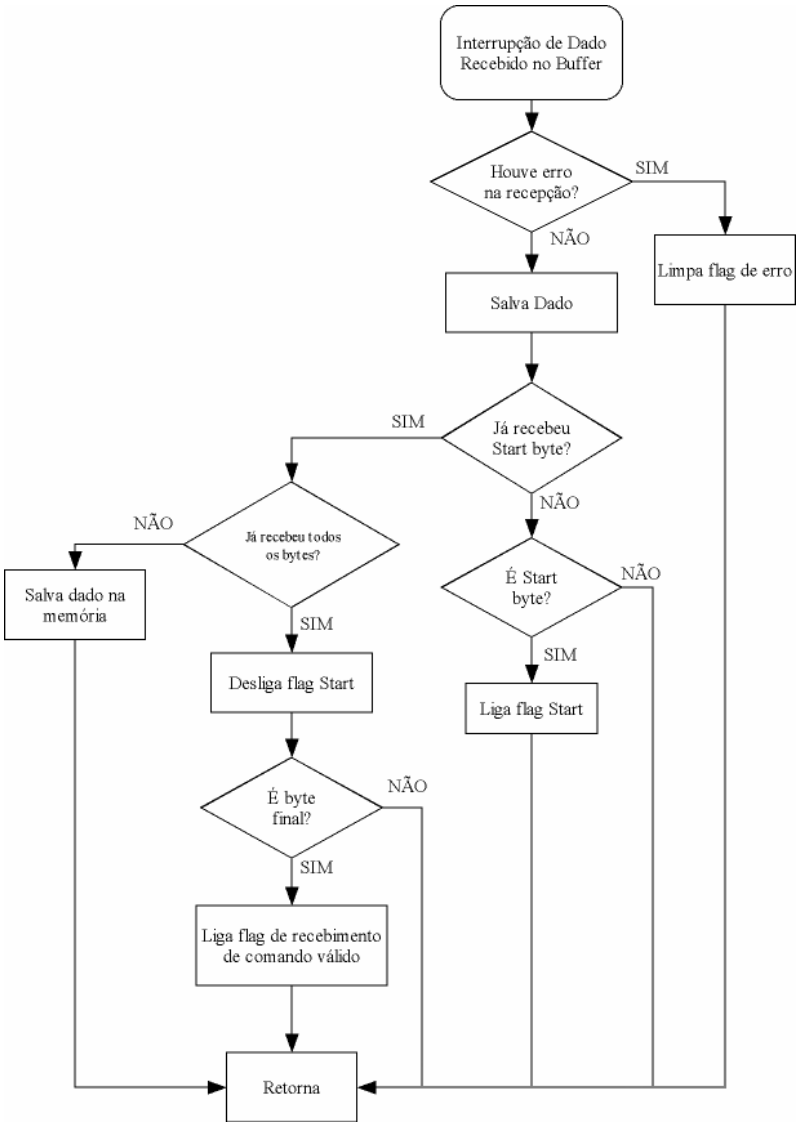

Figura 6 – Fluxograma da rotina de tratamento da interrupção de recepção de dados pelo canal serial

No protocolo estabelecido para comunicação entre PC e robôs ainda foi implementada uma codificação de dados para checagem de erros.

O formato dos bytes a serem enviados aos robôs deve obedecer algumas regras. Sempre o primeiro byte de dados do robô é referente ao motor esquerdo, sendo o segundo

byte referente ao motor direito. O bit mais significativo de cada byte representa a direção para a qual a roda deve girar para movimentar o robô, sendo que 0 indica movimento para frente e 1 indica movimento para trás. Como as rodas são simétricas e cada uma é controlada por um robô, obviamente, caso uma roda tenha movimento para frente e a outra, movimento para trás, ambas com a mesma velocidade, o motor irá girar no seu próprio eixo. Os 5 próximos bits são referentes ao valor decimal de velocidade. Esse valor pode variar de 0 até 31, sendo que 0 determina o desligamento do motor e 31 representa a velocidade máxima. Os últimos 2 bits são para checagem de erros. O computador, ao determinar o sentido de rotação do motor e a velocidade para o mesmo, gera os bits de checagem. O bit 1 é gerado invertendo-se o valor do bit 7, ou seja, invertendo o valor do bit referente ao sentido do motor. O bit 0, o bit menos significativo do byte, é gerado invertendo-se o valor do bit 3.

Nesse caso, admitindo-se que o robô esteja configurado como robô 2, então os bytes de dados são os indicados na figura 7. O primeiro byte refere-se ao sentido e velocidade do motor esquerdo, enquanto o segundo byte refere-se ao motor direito. Para o motor esquerdo tem-se na ilustração a configuração para o motor movimentar-se para trás, com velocidade 10 $(01010)_2$ e os bits de checagem de erro. Como o bit 7 e o bit 3 têm valor 1, então os bits de checagem têm valor 0. No caso do motor direito tem-se o comando para o motor movimentar-se para frente, com velocidade 7 $(00111)_2$ e os bits de checagem de erro. Como o bit 7 tem valor 0, então o bit 1 de checagem de erro tem valor 1. Já o bit 0 tem valor 0 porque o bit 3 tem valor 1, como havia sido explicado anteriormente.

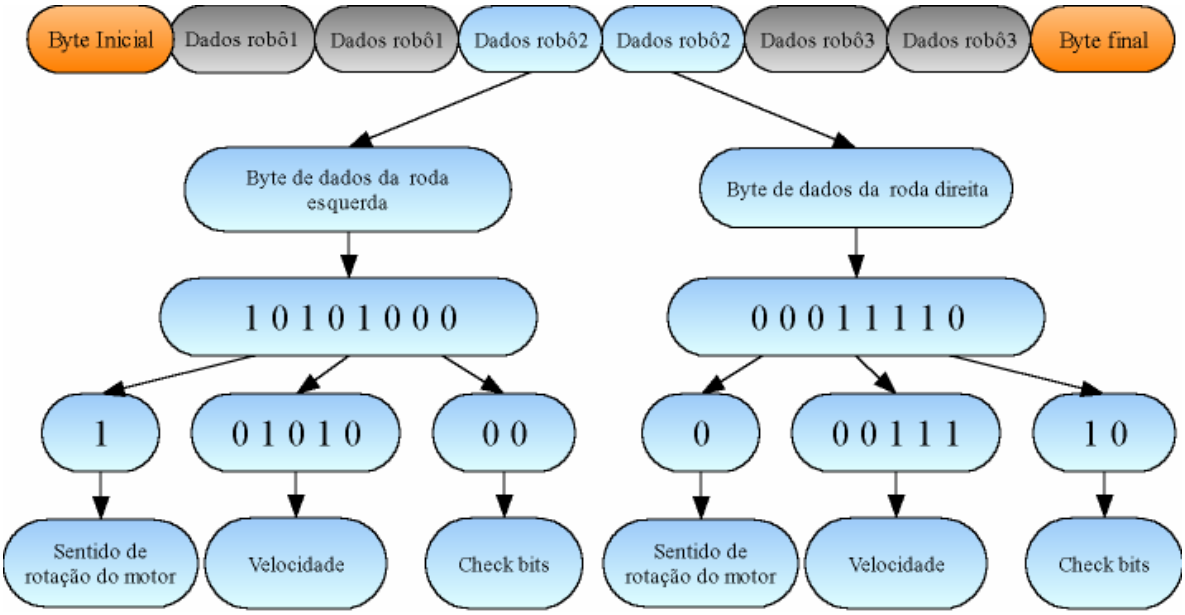

Figura 7 – Formatação dos bytes de dados

A checagem dos bits de conferência acontece quando a sub-rotina de checagem é chamada, o que acontece quando um comando válido é recebido pelo canal serial. Esse *flag* é desligado. O acesso aos bytes de dados acontece de forma indireta, por acesso indireto à memória. Um registrador contém o endereço de memória para o qual deve apontar. Então, nesse registrador é carregado o endereço de memória onde se encontra o primeiro byte de dados. O byte de dados é carregado em uma variável temporária e uma máscara de bits limpa os bits, exceto os bits 1 e 0, de conferência. Os bits de conferência são salvos em outra variável denominada BYTE_CHECK. A seguir, o byte de dados é acessado indiretamente e o bit 7 é testado. Caso ele seja igual a 0, então o bit 1 da variável BYTE_CONF é ligado. Caso contrário o programa segue e o bit 0 do byte de dados é testado. Caso seja 0, então o bit 0 da variável BYTE_CONF é ligado, senão o programa segue. Assim, a variável BYTE_CONF contém os bits de checagem gerados pelo robô, enquanto a variável BYTE_CHECK contém os bits de checagem recebidos pelo canal

serial. Essas variáveis são comparadas. Caso elas não sejam iguais, significa erro na checagem dos bits de conferência. O LED bicolor é aceso em vermelho e o programa retorna. Caso sejam iguais, então a variável que aponta para o primeiro byte de dados é incrementada. A seguir é comparada com o valor do registrador que aponta para o endereço que contém o byte de dados. Se forem iguais significa que os 2 bytes de dados foram checados, então o LED bicolor é aceso em verde, o bit referente à checagem correta é ligado e o programa retorna. Se o valor do registrador (endereço para acesso indireto) não for igual ao endereço do segundo byte de dados, esse registrador é incrementado e o programa desvia para o início da sub-rotina para checar o segundo byte de dados. O fluxograma da sub-rotina de checagem dos bits pode ser visto na figura 8.

3.2.1.3 ATRIBUIÇÃO DAS VELOCIDADES E GERAÇÃO DOS SINAIS PWM

Com os bytes já verificados, o sentido dos motores e a velocidade podem ser atribuídos. As velocidades são valores discretos, decimais, recebidos no byte de dados. No entanto, o controle da velocidade dos motores se dá variando-se a taxa de ciclo ativo do sinal PWM. Essa taxa geralmente é dada por um valor de porcentagem referente ao tempo de sinal em nível alto em relação ao período desse mesmo sinal. Assim, 0% não aciona o motor (sinal sempre em nível baixo) e 100% aciona o motor à plena potência (sinal sempre em nível alto).

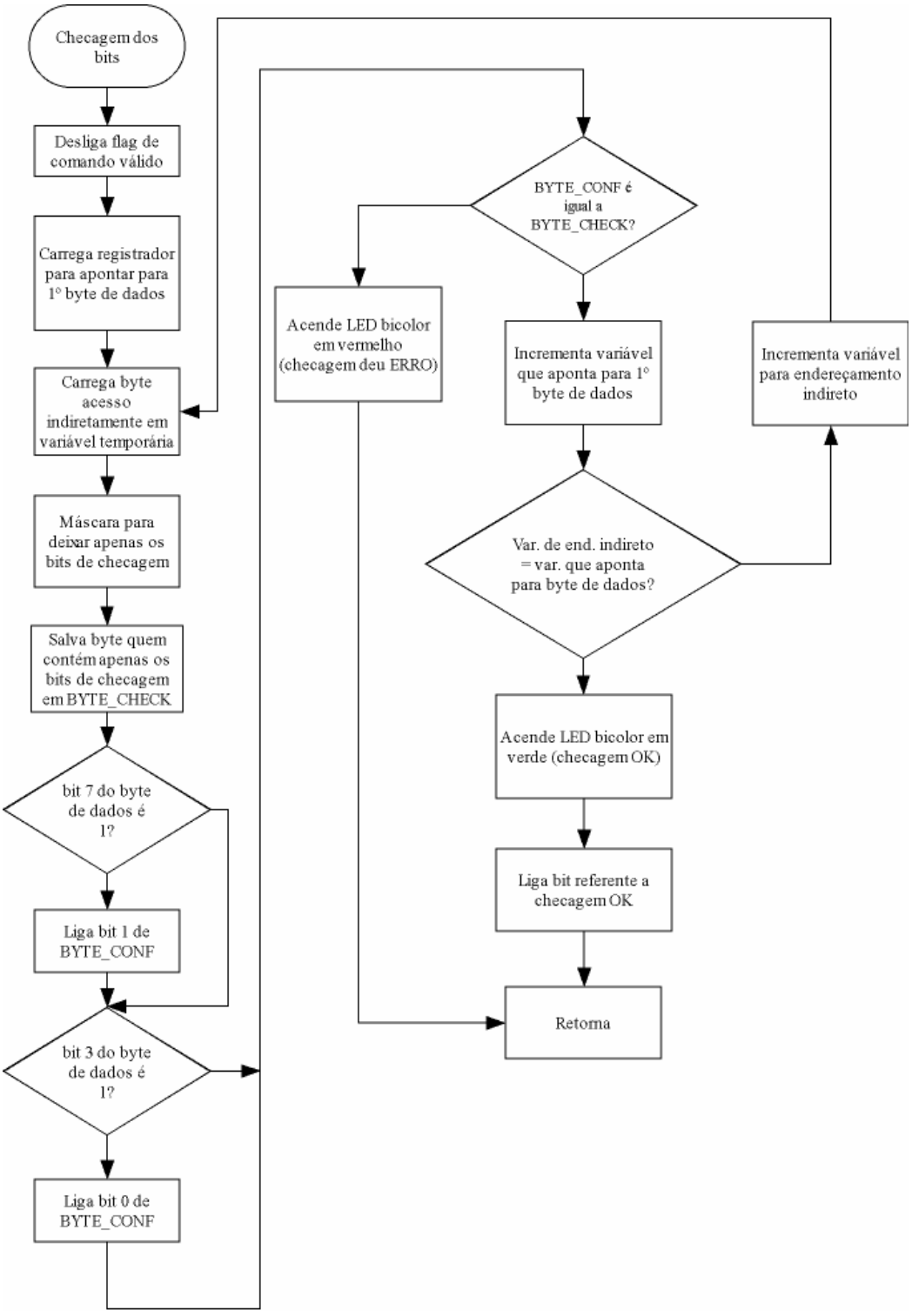

Figura 8 – Fluxograma da sub-rotina de checagem dos bits de conferência

Para que o sinal PWM possa ser gerado no PIC, deve-se carregar alguns registradores com determinados valores, para que se configure a freqüência do sinal, assim como o ciclo ativo do mesmo. Para o cálculo da freqüência deve-se utilizar a seguinte fórmula: $T_{PWM} = [(PR2) + 1]*4*T_{OSC}*(\text{Prescaler TMR2})$

Nessa fórmula o que precisa ser calculado é o valor de PR2, para atribuir a esse registrador, visto que o período T_{PWM}, T_{OSC} e o Prescaler do Timer2 são conhecidos.

Para determinar o ciclo ativo, deve-se calcular um valor de 10 bits para que seja carregado nos respectivos registradores responsáveis por gerar tal ciclo ativo. O ciclo ativo é dado por: $T_{CICLO\ ATIVO} = (<\text{valor de 10 bits}>)*T_{OSC}*(\text{Prescaler TMR2})$

O valor de 10 bits é um valor decimal correspondente aos 10 bits resultantes da associação dos registradores (CCPRxL:CCPxCON<5:4>). Algumas considerações sobre esse valor devem ser observadas. Caso esse valor seja igual a zero, então o ciclo ativo será de 0%. Por outro lado, se esse valor for maior que PR2*4, o ciclo ativo será de 100%. O Timer2 é utilizado para a contagem do tempo e geração do sinal PWM. O sinal tem uma base de tempo (período) e um tempo em que o sinal fica em nível alto (ciclo ativo), como na figura 9.

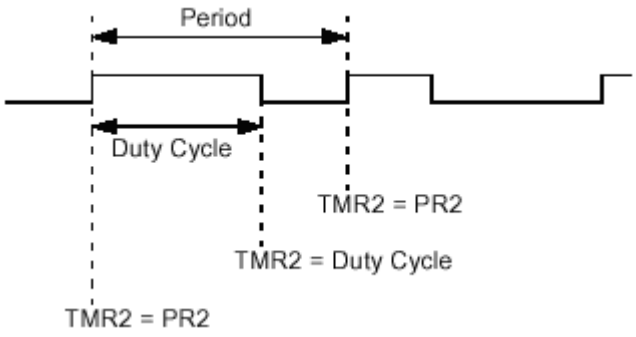

Figura 9 – O sinal PWM controlado pelo Timer 2

Além da característica indutiva dos motores CC e da desejável freqüência acima da faixa de freqüências audível ao ouvido humano, como já citado, a freqüência de 25 KHz foi escolhida por ter-se observado uma característica peculiar, muito interessante para o projeto, além das considerações observadas no artigo *Choosing the right PWM frequency*[35] e em uma lista de discussão sobre PWM[36]. A velocidade referente ao motor é apenas um valor de 0 até 31 correspondente a um ciclo ativo cada uma. O software, então, deve analisar esse valor, calcular o ciclo ativo, ou seja, calcular o valor de 10 bits a partir do valor da velocidade e atribuir tal valor aos registradores para que os módulos CCP possam gerar os sinais PWM como desejado.

À princípio duas dificuldades foram encontradas. Implementar instruções matemáticas complexas, como multiplicação e divisão no PIC é possível, mas seria gasto muito tempo nesse cálculo. A dificuldade em relação à atribuição dos 10 bits aos registradores é grande, visto que os 2 bits menos significativos desse valor devem ser atribuídos ao registrador CCPxCON<5:4>, enquanto os 8 bits restantes devem ser atribuídos ao registrador CCPRxL. A solução utilizada surgiu após algumas tentativas e cálculos utilizando alguns valores de freqüência. O resultado é uma atribuição de velocidade com tempo fixo, rápida e muito interessante. O diagrama de blocos da figura 10 ilustra a solução.

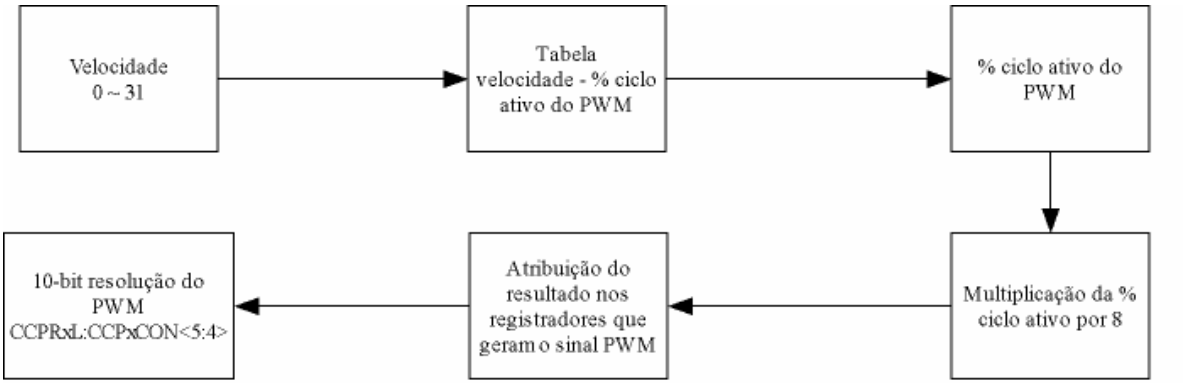

Figura 10 – Diagrama de blocos da atribuição das velocidades

O valor de velocidade, que varia de 0 (motor desligado) até 31 (100% de ciclo ativo), é interpretado como o índice de uma tabela que relaciona os valores de velocidade com uma taxa de ciclo ativo. A tabela retorna um valor referente à porcentagem desse ciclo ativo. Para a freqüência de 25 kHz adotada, calculou-se o valor de PR2 utilizando Prescaler do Timer2 igual a 1. PR2 deve ser carregado com o valor $(199)_{10}$. Modificando a fórmula do cálculo do ciclo ativo convenientemente, chega-se a uma outra fórmula:

(valor de 10 bits) = (% * T_{PWM}) / (T_{OSC}*(TM2 prescale value))

Como o período do sinal PWM é conhecido (inverso da freqüência, no caso 25 kHz), assim como o período do sinal de *clock* (inverso da freqüência, no caso 20 MHz, correspondente ao cristal utilizado no circuito) e também o valor do prescaler do Timer2, então, ao substituir-se tais valores chega-se à seguinte fórmula:

$(\text{valor de 10 bits})_{10} = (\% * 8)_{10}$

Logo, quando a tabela retorna a porcentagem referente à velocidade desejada, é necessário, apenas, multiplicar esse valor por 8. Ao calcular-se alguns valores de

velocidade e de ciclo ativo para que um algoritmo fosse criado, percebeu-se que, configurado para 25 kHz, com PR2 carregado com valor $(199)_{10}$ e prescaler do Timer2 igual a 1, sempre, para qualquer porcentagem de ciclo ativo, os 2 bits menos significativos serão zero. Com isso resolveu-se o problema da atribuição do valor de 10 bits aos registradores, pois os bits CCPxCON<5:4> serão sempre zero e deve-se apenas carregar os 8 bytes restantes no registrador CCPRxL. Para multiplicar os valores de porcentagem de ciclo ativo por 8, bastava apenas fazer 3 deslocamentos para a esquerda, visto que cada deslocamento multiplica o valor por 2. No entanto, como os 2 bits menos significativos serão sempre zero e estão em outro registrador, a única coisa a ser feita é deslocar um bit à esquerda desse valor de porcentagem para ter-se um valor de 10 bits multiplicado por 8. Por exemplo, a velocidade 30. A tabela irá retornar 99% de taxa de ciclo ativo. O valor $(99)_{10}$, ou $(01100011)_2$ multiplicado por 8 resulta em $(792)_{10}$, ou $(1100011000)_2$. Pode-se observar que o valor que deve ser carregado nos registradores tem seus 2 bits menos significativos iguais a zero, enquanto os 8 bits mais significativos restantes são o valor da taxa de ciclo deslocado uma vez à esquerda. Assim conclui-se que é apenas necessário utilizar o valor discreto de velocidade como índice na tabela, para que seja retornado um valor referente à taxa de ciclo ativo, deslocado uma vez à esquerda e atribuído ao registrador CCPRxL. Podia-se fazer com que a tabela já retornasse o valor deslocado à esquerda, no entanto, é mais fácil de se entender e facilita também futuras alterações da taxa de ciclo referente a cada velocidade, até porque o tempo de uma única instrução a ser executada a mais não irá influenciar no funcionamento do programa.

Para atribuir as velocidades, a sub-rotina inicia desligando o *flag* referente à checagem de bits OK. Acessando indiretamente, o 1º byte de dados é copiado em uma variável temporária. O bit 7, que representa a direção de rotação do motor, é testado. Caso

ele seja igual a 1 o *flag* referente à direção do motor é ligado, caso contrário, nada é feito e a sub-rotina continua. Uma máscara de bits é utilizada para que apenas os bits de velocidade se mantenham, enquanto os demais são apagados. Esse valor de velocidade é salvo em uma variável chamada VEL_ESQ e rotacionado 2 vezes à direita, para que o valor de velocidade esteja posicionado a partir do bit 0 dessa variável. O registrador de endereçamento indireto é incrementado para apontar para o segundo byte de dados. O byte de dados é carregado em uma variável temporária, o bit 7 é testado e, caso seja igual a 1, o *flag* referente à direção do motor direito é ligado, senão a sub-rotina continua. É utilizada uma máscara para que apenas os bits de velocidade não sejam alterados, esse dado é salvo na variável VEL_DIR, que é rotacionada duas vezes para posicionar o valor da velocidade a partir do bit 0. A tabela de conversão de valores de velocidade para valores de porcentagem de taxa de ciclo ativo é consultada com a velocidade contida em VEL_ESQ e retorna o valor da porcentagem. Esse valor retornado é salvo na mesma VEL_ESQ, que é rotacionada à esquerda e esse valor é atribuído ao registrador CCPR1L, responsável por gerar a taxa de ciclo ativo do sinal PWM para o motor esquerdo. Em seguida, a tabela é consultada novamente, agora com o valor de velocidade contido em VEL_DIR, a qual retorna a porcentagem na mesma variável VEL_DIR, que é rotacionada à esquerda e esse valor á atribuído ao registrador CCPR2L, responsável por gerar a taxa de ciclo ativo do sinal PWM para o motor direito. Agora o *flag* de direção do motor esquerdo é testado, caso seja igual a zero, o motor esquerdo irá girar no sentido anti-horário (para deslocar robô para frente), senão irá girar no sentido horário (para deslocar robô para trás). O *flag* de direção do motor direito é, então, testado. Caso seja igual a zero, motor irá girar no sentido horário (para deslocar robô para frente), caso contrário, irá girar no sentido anti-horário(para deslocar robô para trás). Um valor é carregado na variável MEU_WDT é carregado para a

contagem de aproximadamente 200 ms. O *flag* que indica que os motores estão funcionando é ligado. A sub-rotina retorna. A figura 11 é o fluxograma dessa sub-rotina.

O software ainda tem duas partes importantes, a parte onde a interrupção do Timer0 é tratada e o *LOOP* principal do programa, as quais serão descritas a seguir.

O tratamento da interrupção do Timer0 acontece quando o valor carregado em seu registrador estoura, gerando a interrupção. Então, o *flag* de interrupção do Timer0 é desligado, a variável Contador é decrementada e testada. Caso seja diferente de zero, a sub-rotina desvia e continua sua execução. Caso seja igual a zero, o LED de atividade é testado para verificar se está aceso. Se o LED de atividade estiver aceso, ele é apagado, se estiver apagado, ele é aceso. Então ele irá piscar em uma freqüência determinada pelo tempo configurado no Timer0 multiplicado pelo valor carregado em Contador. Após acender ou apagar o LED, o valor é carregado em Contador novamente, pois ele havia chegado a zero. A sub-rotina então prossegue com sua execução. É testado o *flag* que indica o funcionamento dos motores. Caso eles não estejam ligados, a sub-rotina retorna. Senão, é decrementada a variável MEU_WDT, responsável por gerar uma base de tempo de aproximadamente 200 ms. Caso ela seja diferente de zero, a sub-rotina retorna, pois ainda não atingiu os 200 ms. Caso contrário, os motores são desligados, o *flag* que indica o funcionamento dos motores também é desligado, o LED de indicação da checagem de bits de conferência é apagado. Essa situação garante que, caso ocorra algum problema com a transmissão/recepção dos dados, seja por não recebimento, ou por verificação de erros encontrados, os motores desligam após 200 ms para que o robô pare e não se mova sem a devida ordem de movimento a ele transmitida. Isso garante que ele pare, evitando colisões com outros robôs, ocasionando faltas no jogo.

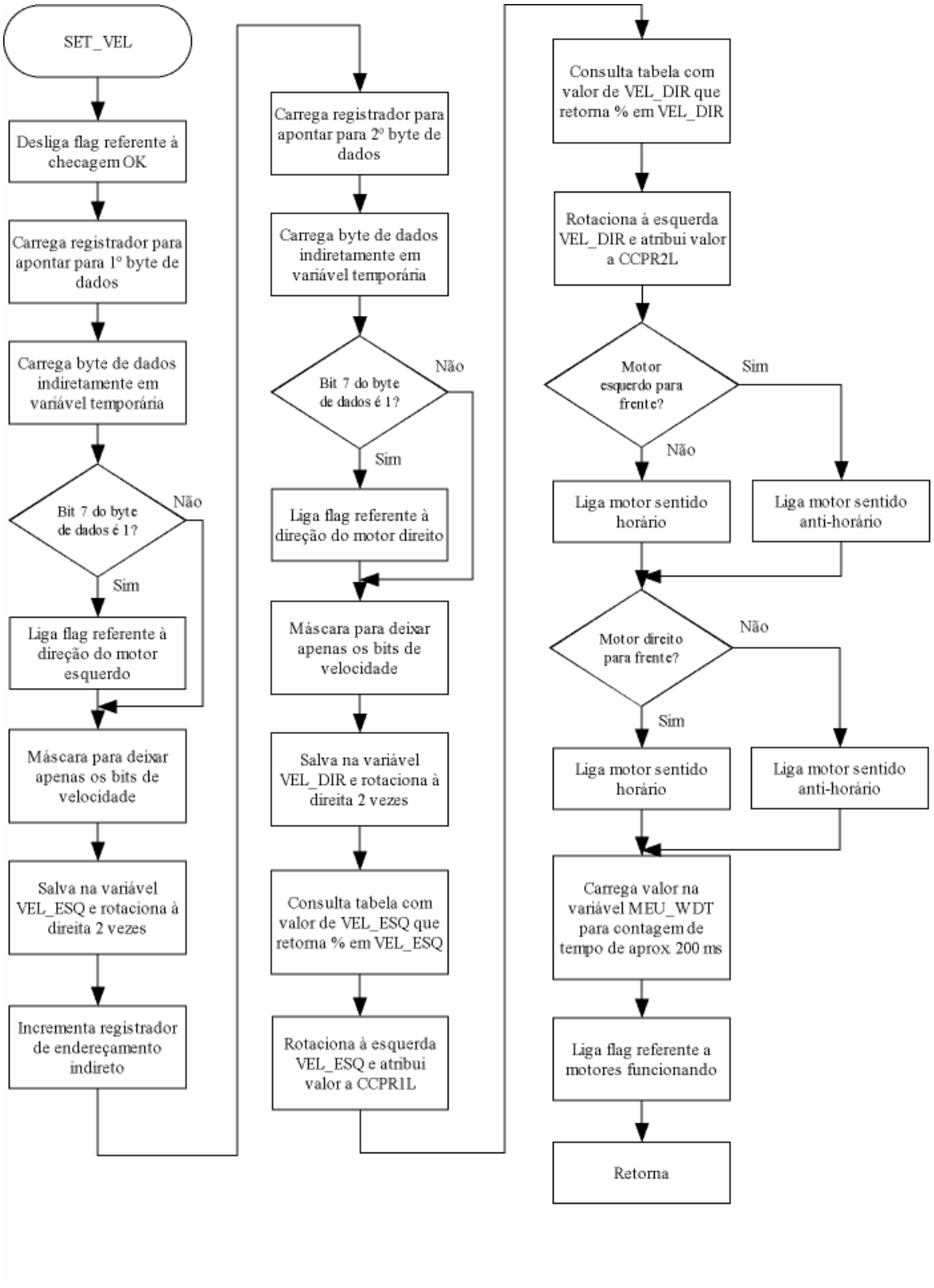

Figura 11 – Fluxograma da sub-rotina de atribuição das velocidades

Obviamente que a maneira mais interessante de controlar o robô é enviando as mensagens de movimento no menor tempo possível, respeitando o tempo total gasto em um *loop* completo do software, que seria o recebimento da mensagem pelo canal serial, verificação da integridade dos dados, atribuição das velocidades e uma futura odometria, que só pode ser testada quando se tiver um protótipo mecânico. A figura 12 ilustra o fluxograma da rotina de tratamento da interrupção do Timer0.

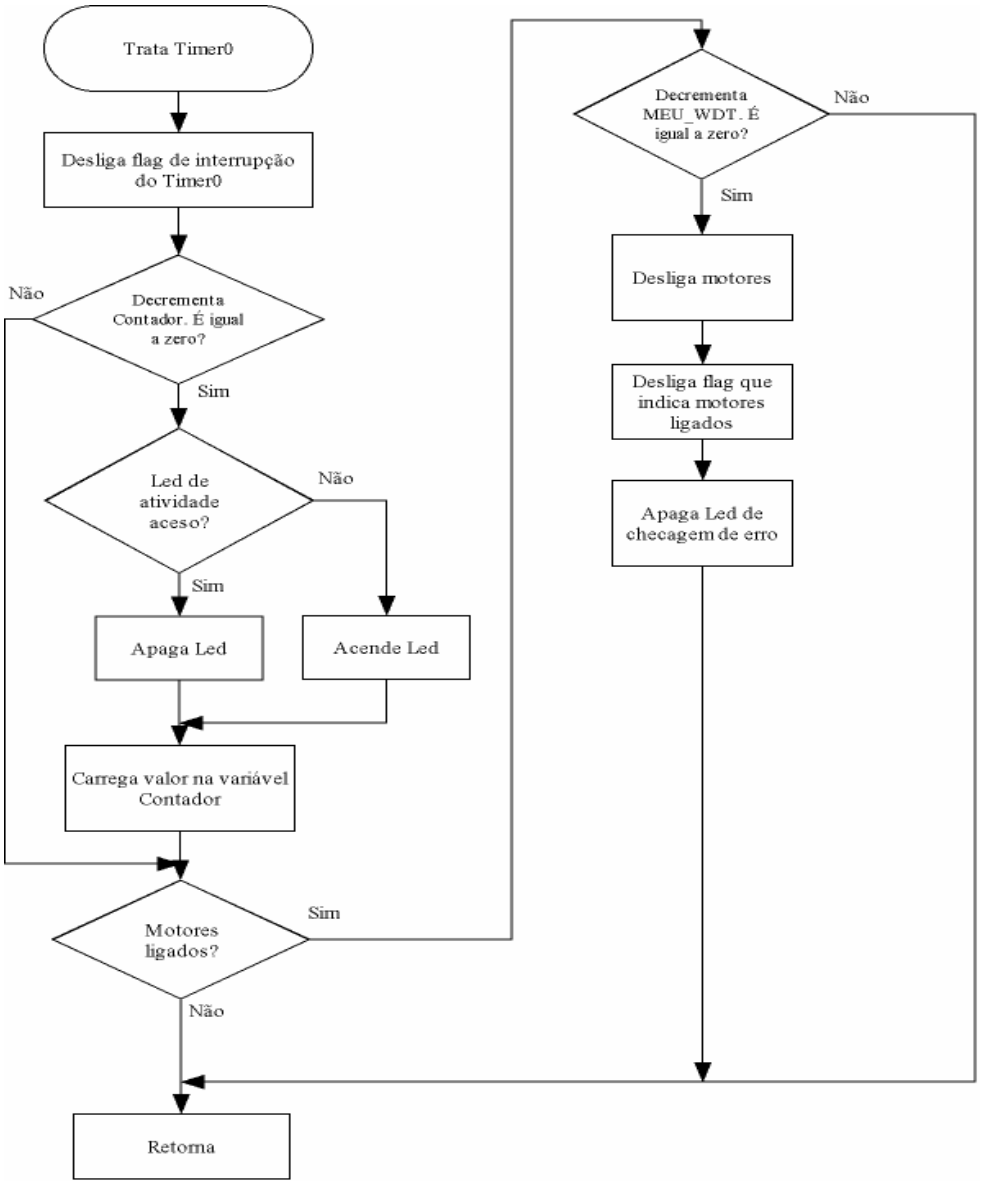

Figura 12 – Fluxograma da rotina de tratamento da interrupção de Timer 0

O *loop* principal do programa não faz nada além de testar *flags* indicativos e desviar para tratar as informações referentes a esses *flags*. O primeiro a ser testado é o *flag* indicativo de mensagem completa. Esse *flag* é ligado na Interrupção de Tratamento de Dado Recebido no Buffer do canal serial quando uma mensagem completa é recebida. Quando isso acontece, o programa desvia para a sub-rotina CHECA_BITS, responsável pela checagem dos bits de conferência. Após retornar, é testado o *flag* que indica o status da verificação dos bits de conferência. Caso tenha havido sucesso nessa checagem, esse *flag* é ligado e nesse ponto o programa chama a sub-rotina SET_VEL, a qual tem a função de interpretar os valores de velocidade recebidos para cada motor, assim como a direção de rotação dos mesmos e atribuir os valores de taxa de ciclo ativo aos registradores para que os sinais PWM sejam gerados e os motores devidamente controlados. Feito isso, o programa volta a testar o *flag* indicativo de mensagem válida e segue toda essa descrição novamente. O programa roda em um *looping* infinito. Vale ressaltar algumas considerações. A configuração do endereço do robô é feita apenas uma vez. Caso seja necessário alterar o endereço, é necessário *resetar* o microcontrolador. Os dados recebidos pelo canal serial, assim como a contagem de tempo para o LED atividade piscar e os motores permanecerem ligados geram interrupções que são tratadas imediatamente ao acontecerem. Com isso, os motores sempre serão desligados após um intervalo de 200 ms e nenhum dado recebido pelo canal serial será ignorado. O fluxograma da figura 13 ilustra o *loop* principal.

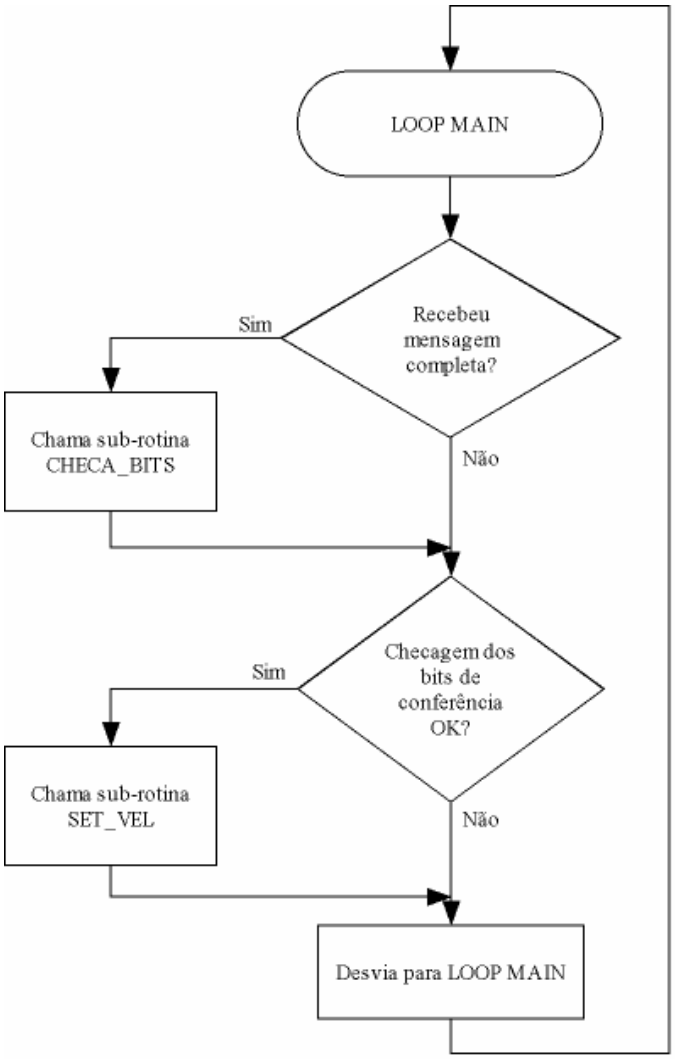

Figura 13 – Fluxograma do *loop* principal

3.2.2 BATERIAS

Foram utilizadas 6 pilhas de Ni-MH recarregáveis tamanho AA, que fornecem tensão de 1,2V e corrente de 3000 mAh. Ligando essas pilhas em série obtém-se uma tensão máxima de 7,2V, que é utilizada na etapa de potência para acionamento dos motores, ou seja, tais motores serão acionados à sua tensão máxima de operação para que se obtenha o maior torque capaz de ser gerado. Uma tensão intermediária de 4,8V alimenta

o circuito lógico. A obtenção da tensão intermediária de 4,8V se fez necessária porque o circuito lógico funciona com níveis de tensão TTL. Essa tensão é obtida na quarta pilha, das 7 que estão ligadas em série, como mostra a figura 14.

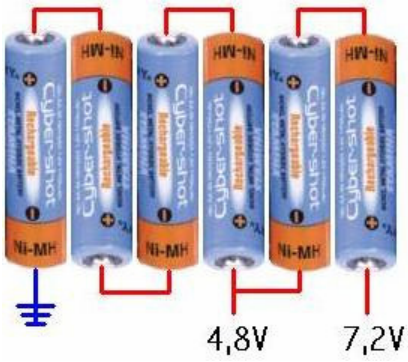

Figura 14 – Ligação das pilhas para obtenção das tensões necessárias

3.2.3 MOTORES

O protótipo do projeto utiliza 2 motores CC modelo SPEED 280 SPORT[37], da fabricante Graupner, utilizados eu aeromodelismo. Esse modelo foi escolhido pelas suas características elétricas, como a tensão nominal de operação de 6V e a faixa de tensão de funcionamento de 4,8V a 7,2V, além de sua velocidade de rotação de 18000 RPM, que, combinada com uma redução mecânica, irá resultar em uma boa velocidade final e um torque suficientemente forte em relação ao peso final do modelo. Os motores são dotados de capacitores de supressão de ruídos, de acordo com o manual fornecido pelo fabricante.

3.2.4 CONTROLE DE VELOCIDADE DOS MOTORES

Existem várias técnicas de controle de velocidade de motores CC. O controle linear consiste em aplicar uma tensão proporcional à velocidade de rotação. O principal problema

dessa técnica de controle é que em baixas rotações não se tem o torque máximo do motor, pois uma tensão menor que a tensão nominal está sendo aplicada ao motor. Uma outra técnica muito utilizada para controle de velocidade de motores CC é o controle PWM, que consiste em um sinal de onda retangular cujo ciclo ativo é proporcional à velocidade de rotação. A onda retangular tem como limiares a tensão nominal do motor e o terra (0V). O motor, então, será acionado à sua plena potência, sua tensão nominal, mas esse acionamento será chaveado pela onda retangular.

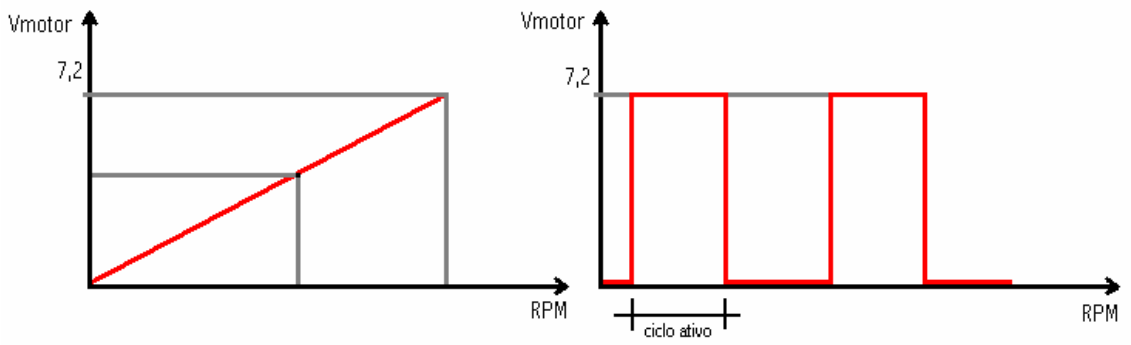

Figura 15 – Controle linear X Controle PWM

Assim o controle PWM mantém o torque máximo do motor, enquanto a taxa de ciclo ativo determina a velocidade de rotação. A técnica de controle PWM foi escolhida por ser a mais adequada ao tipo de controle necessário aos motores, mantendo sempre o torque máximo e controlando a velocidade através da largura do pulso de um sinal de freqüência constante. Apesar de o controle PWM ser largamente utilizado, a determinação da freqüência desse sinal deve levar em consideração características muito particulares de motores e de circuitos para gerar essa forma de onda. A característica indutiva dos motores impede que altas freqüências sejam utilizadas para seu controle. Sendo assim, a faixa de

freqüência de algumas dezenas de kHz é a mais utilizada para controle de motores. A freqüência do sinal PWM do projeto é de 25 kHz, acima da faixa de freqüências audíveis do ser humano, para que não haja ruídos audíveis inconvenientes.

3.2.5 CONTROLE DE SENTIDO DE ROTAÇÃO E ETAPA DE POTÊNCIA

O sinal de controle PWM é gerado pelo microcontrolador, então sua tensão em nível lógico alto é de 4,8V. No entanto os motores devem ser acionados com tensão de 7,2V. Necessitou-se, então, de uma etapa de potência, com a qual fosse possível, também, alterar o sentido de rotação dos motores, ou seja, o sentido do fluxo de corrente nos motores. Um circuito eletrônico conhecido como Ponte H é capaz de controlar o sentido do fluxo de corrente, bem como aumentar a capacidade de corrente fornecida pelo circuito. A Ponte H é composta por 4 transistores que trabalham em suas regiões de corte e saturação, isto é, funcionam como chaves lógicas. O circuito integrado L298N[38], da *National Semiconductor*, cujo circuito elétrico pode ser visto na figura 16, possui tamanho reduzido e é composto por duas Pontes H completas, que possuem entradas de habilitação de funcionamento, pinos sensores de corrente, entradas para o controle do sentido dos motores e saídas de acionamento dos motores. Esse circuito não possui diodos de proteção internos, mas suas demais características se encaixam perfeitamente nas necessidades de projeto. Os diodos de proteção utilizados são de chaveamento rápido do tipo Schottky, modelo BYV27200[39]. Eles se fazem necessários para que a tensão reversa na junção coletor-emissor dos transistores das Pontes H não se exceda a ponto de queimá-los. Esse circuito deve ser alimentado com a tensão nominal dos motores (Vs = 7,2V) para acioná-los

corretamente, além da tensão lógica (Vss = 4,8V), para que possam ser utilizados sinais em níveis lógicos para o controle dos motores.

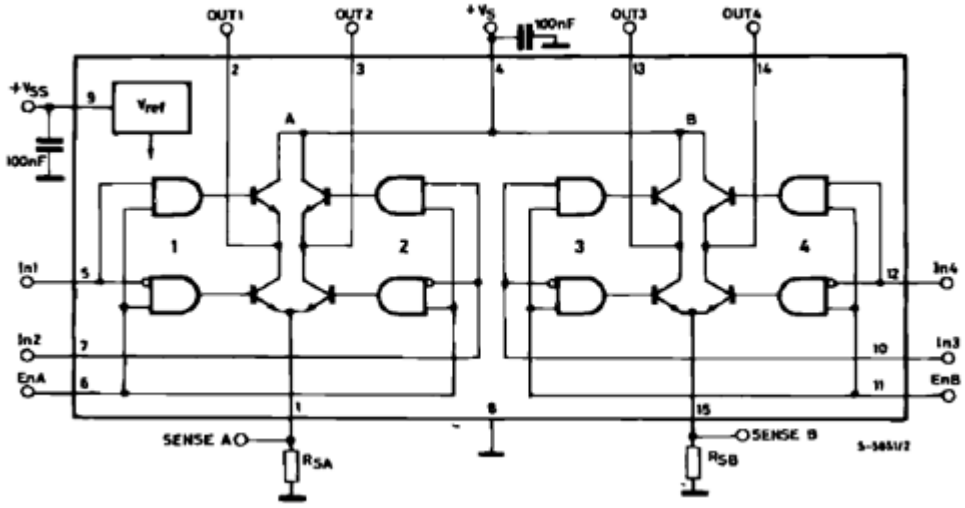

Figura 16 – Circuito elétrico do integrado L298N

Esse circuito integrado implementa a etapa de potência para o acionamento dos motores, assim como o controle do sentido de rotação dos mesmos. Cada Ponte H é capaz de fornecer até 2A para a carga a ser controlada, os motores, nesse caso, além de aceitar tensões de acionamento das cargas até 46V. Os pinos de habilitação de cada Ponte H recebem os sinais de controle PWM, enquanto os pinos de entrada recebem os sinais de controle de sentido de rotação dos motores. As duas entradas em nível lógico baixo deixam o motor girar livremente, enquanto as duas entradas em nível lógico alto, ou seja, os 2 pólos do motor ligados à tensão nominal, travam seu eixo, o que é conhecido como freio dinâmico. Aplicando níveis lógicos opostos nas entradas (nível alto em uma entrada e nível baixo em outra entrada, por exemplo) controla-se o sentido de rotação dos motores.

3.2.6 ENCODERS

O controle de velocidade dos motores é feito em malha fechada. Quando os motores são acionados sem carga a sua velocidade de rotação é constante e a curva de variação de rotação em relação à largura do pulso do sinal PWM que controla o motor é linear. No entanto existirá uma carga, a qual irá proporcionar um esforço maior ao motor. Outro fator implicante é que a carga será variável, podendo variar de acordo com fatores como sujeira no acoplamento mecânico, peso da bola ao empurrá-la, entre outros. Mas é desejável que ao aplicar-se uma determinada velocidade, tal velocidade se mantenha constante, mesmo que haja uma variação de carga. Então é interessante que um sinal correspondente à velocidade real retorne ao circuito de controle para que haja uma comparação de velocidades real e teórica, para que seja feita a devida correção, caso seja necessária. Essa realimentação do circuito de controle com o sinal da velocidade teórica é que caracteriza um controle em malha fechada. Essa realimentação pode ser obtida de diversas maneiras, entre as quais 2 técnicas se destacam. Pode ser acoplado ao eixo do motor um dínamo que irá gerar uma tensão linear e proporcional à velocidade de rotação. A outra técnica, que foi a escolhida, possui vantagens como a precisão e por fornecer um sinal digital, ao invés do sinal analógico fornecido pela técnica anterior. Um LED emissor de infravermelho, PHIV459[40] fica alinhado com um fototransistor, o PHFT458[41], ambos da fabricante nacional Politronic. Entre eles e acoplado ao eixo do motor é instalado um disco-encoder, que possui vários cortes, os quais impedem ou deixam passar o feixe de luz do LED emissor de infravermelho, ilustrado na figura 17.

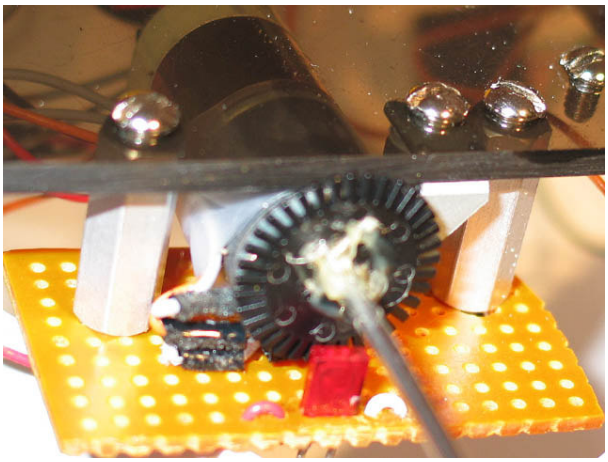

Figura 17 – Ilustração de um disco-encoder acoplado ao motor

O sinal proveniente do coletor do fototransistor é enquadrado por um inversor *Shmitt-Trigger*, implementado no circuito integrado 74HC14[42], resultando em uma onda quadrada, cuja freqüência representa uma determinada velocidade de rotação do motor. O circuito elétrico referente à realimentação dos valores de velocidade dos motores pode ser visto na figura 18.

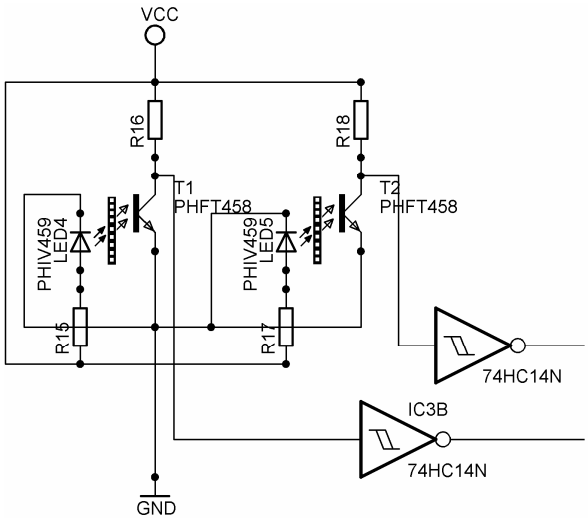

Figura 18 – Circuito elétrico de realimentação da velocidade

Esse sinal pode, então, ser ligado diretamente ao microcontrolador.

3.2.7 O CIRCUITO ELÉTRICO

Observando o circuito elétrico, na figura 19, pode-se identificar algumas partes já descritas anteriormente, como a etapa de potência dos motores, os diodos de proteção, os circuitos dos encoders. Existe, ainda, um par de circuitos compostos por um transistor BC558[43], um diodo zener e resistores para polarização do transistor. Cada um desses dois circuitos é alimentado por uma tensão, são elas a tensão de alimentação do circuito lógico (Vcc) e a tensão de alimentação dos motores (Vmotor), as quais deseja-se monitorar. Quando a tensão de alimentação for maior que a tensão do diodo (Vz) mais a queda de tensão na junção emissor-base (~0,7V), ou seja, quando a alimentação estiver acima de Vz+0,7V, então a junção emissor-base estará polarizada diretamente e haverá uma corrente de base Ib, fazendo com que o transistor sature e a tensão no coletor seja a tensão de alimentação. A queda da tensão de alimentação para um valor menor ou igual a Vz+0,7V faz com quem a junção emissor-base não seja polarizada diretamente, não havendo corrente de base Ib, fazendo com que o transistor trabalhe na sua região de corte e, sendo assim, a tensão no coletor será de 0V, ou seja, o nível de terra. Esses circuitos impõem limiares para o funcionamento da eletrônica do robô. Para a alimentação do circuito lógico foi utilizado um diodo zener cuja queda de tensão é de 3,3V, resultando em um limite de 4V. Para a alimentação dos motores foi utilizado um diodo zener com queda de tensão igual a 5,6V, o que resulta em um limite de 6,3V. Caso alguma das 2 tensões se igualem aos limites, o transistor corta e leva o sinal de saída ao nível zero, então os circuitos são ativos em nível baixo. Os LEDs 2 e 3 indicam nível baixo de baterias, isto é, que as tensões de alimentação estão abaixo dos limites. O LED 2 acende quando a tensão lógica não é suficiente para garantir o correto funcionamento do circuito, ou seja, quando atinge o limite inferior de 4V.

Já o LED 3 acende quando a tensão dos motores atinge o limite inferior estipulado em 6,3V.

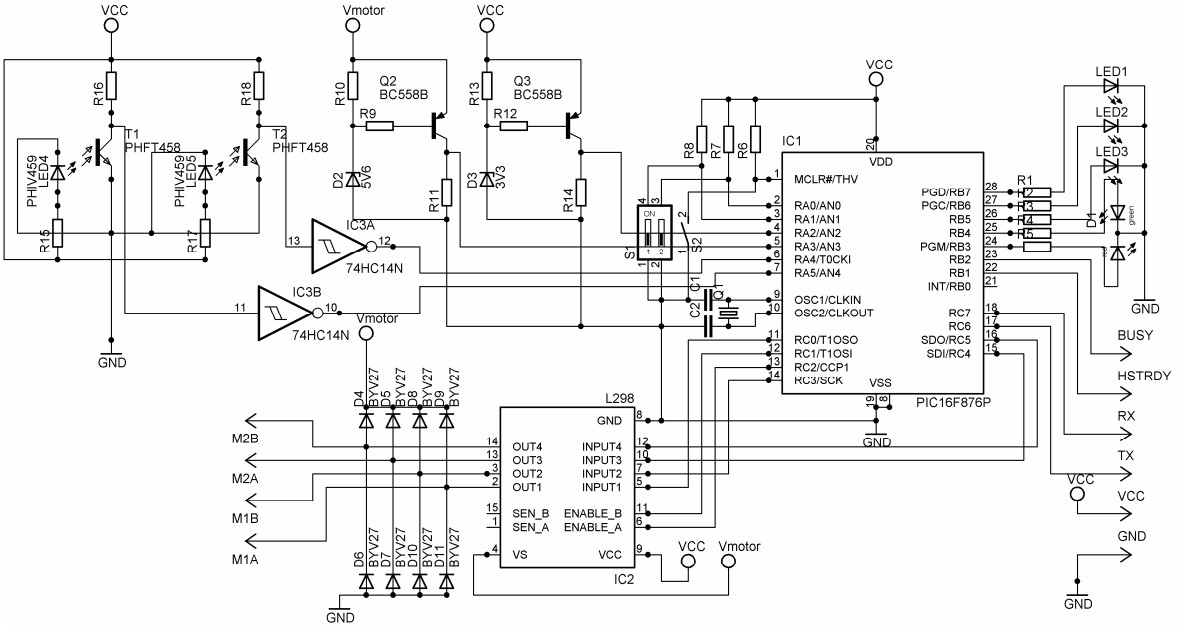

Figura 19 – Circuito elétrico proposto para o protótipo do robô

CAPÍTULO 4

CONCLUSÃO

Inicialmente pretendia-se desenvolver um protótipo completo de um novo robô. A idéia era desenvolver novo hardware, software e a parte mecânica, utilizando o sistema de transmissão/recepção já existente.

Como previsto em cronograma, estudos sobre a viabilidade de tais atividades foram feitos. Constatou-se que a construção da parte mecânica envolvia materiais e métodos que fugiam do escopo principal do objetivo inicial, sendo que tal atividade foi proposta a outro aluno, que possuía as devidas competências para desenvolver o projeto mecânico, como o cálculo de engrenagens da relação de redução motor-roda, material utilizado na construção, a construção propriamente dita, entre outros aspectos.

O sistema de comunicação de dados atual não satisfazia as necessidades iniciais de projeto, dada a sua baixa taxa de transferência de dados. No entanto o desenvolvimento de um sistema de comunicação de dados sem fio é complexo o suficiente para ser apenas parte de um projeto, além de ter um custo elevado na sua implementação. Como sugestão foram apontadas várias alternativas e soluções já prontas de circuitos integrados que implementam tais funcionalidades, sugestões as quais estão sendo analisadas em outro projeto, o qual tem como objetivo implementar a comunicação de dados sem fio para o presente projeto.

Estudos e desenvolvimento se concentraram no hardware e software do novo protótipo, o qual não foi implementado em Lego, pois não houve como integrar os motores utilizados às engrenagens Lego, sem que as peças fossem danificadas.

Alguns aspectos foram deixados de lado, enquanto outros foram incorporados ao projeto.

Como resultado, um protótipo muito robusto e flexível, no que diz respeito a protocolo de comunicação e configuração, o qual pode ser utilizado em outras categorias de Futebol de Robôs ou até mesmo como uma plataforma de robô móvel para diversas aplicações.

Funcionalidades como odometria e controle PID digital foram parcialmente implementadas em caráter de teste, os quais não foram bem sucedidos. Por premência de tempo os problemas encontrados não foram resolvidos e tais funcionalidades não foram inseridas nesse relatório, nem no protótipo final em protoboard, para que, ao final do projeto, houvesse um protótipo funcional. Fica como sugestão para melhoria do protótipo a implementação de uma odometria capaz de corrigir desvios de velocidades nas rodas, utilizando um controle PID digital para corrigir tais desvios.

CAPÍTULO 5

REFERÊNCIAS BIBLIOGRÁFICAS

[1] - Alan Mackworth - http://www.cs.ubc.ca/spider/mack/

[2] - *On Seeing Robots Paper* - http://www.cs.ubc.ca/spider/mack/links/papers/osr/osr.html

[3] - *Computer Vision: System, Theory, and Applications*, páginas 1-13, *World Scientific Press*, Singapore, 1993

[4] - Hiroaki Kitano - http://www.csl.sony.co.jp/person/kitano/

[5] - *DeepBlue* - http://www.chess.ibm.com/

[6] - *Pathfinder Mission* - http://www.chess.ibm.com/

[7] - *Fira* - http://www.fira.net

[8] - Categoria *Mirosot* - http://www.fira.net/soccer/mirosot/overview.html

[9] - Costa, A. H. R.; Pegoraro, R. - Construindo Robôs Autônomos para Partidas de Futebol: O Time Guaraná (1999)

[10] - *1999 FIRA CUP BRAZIL* - http://fira.net/firacup/1999.html

[11] - FUTEPOLI - Futebol de Robôs - USP - http://www.lti.pcs.usp.br/robotics/futepoli

[12] - Automática 2005 - http://www.fem.unicamp.br/~mecatron

[13] - *LARC - Latin American Robotics Competition* - http://www.larc.dee.ufma.br/

[14] - VII SBAI/II LARS - http://www.dee.ufma.br/sbailars/apresentacao.php

[15] - Motores CC - Documento da UFBA - http://www.eletronica.org/modules.php?name=News&file=article&sid=181

[16] - *Encoder Newsletter - Seattle Robotics Society* - http://www.seattlerobotics.org/encoder

[17] - Noceti Filho, S. - Filtros - Seletores de Sinais, Editora da UFSC, Florianópolis, 2003

[18] - Ogata, K. - Engenharia de Controle Moderno, Capítulo 1, 3ª ed., Editora LTC

[19] - Keramas, J. G. - Robot Technology Fundamentals, Capítulo 6, Delmar, 1999

[20] - Johnson, T. – Mousebot -

http://www.seattlerobotics.org/encoder/200311/johnson/Mousebot.html

[21] - AT89C2051 datasheet -

http://www.atmel.com/dyn/products/product_card.asp?part_id=1938

[22] - Mineirosot - http://www.lrvpa.dcc.ufmg.br/mirosot/pt/

[23] - Time de Futebol de Robôs da UFPR - http://www.ufpr.br/futrobos

[24] - Boc Board - http://www.lrvpa.dcc.ufmg.br/projetos/boc/bocboard/

[25] - Microchip Technology Inc. - http://www.microchip.com

[26] - Ogata, K. - Engenharia de Controle Moderno, Capítulo 10, 3ª ed., Editora LTC

[27] - LM629 datasheet - http://www.national.com/pf/LM/LM629.html

[28] - AUSTRO - IHRT - http://www.robosoccer.at

[29] - AT89C52 datasheet - Atmel - http://www.atmel.com

[30] - Robotis - http://www.robotis.com

[31] - PIC 16F876A datasheet -

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&d

DocName=en010240

[32] - Souza, D. J. - Desbravando o PIC - Ampliado e Atualizado para PIC 16F628A,

Editora Érica, 2003

[33] - Pereira, F. - Microcontroladores PIC Técnicas Avançadas, Editora Érica, 2002

[34] - Zanco, W. S. - Microcontroladores PIC 16F628A/648A Uma Abordagem Prática e

Objetiva, Editora Érica, 2005

[35] - Jorgensen, B. P. - *Choosing the right PWM frequency* - http://www.seattlerobotics.org/encoder/200011/pwm.html

[36] - Discussão sobre PWM - http://www.seattlerobotics.org/encoder/sep99/pwmmail.html

[37] - *SPEED 280 SPORT – Graupner Modellbau* - http://www.graupner.com/

[38] - L298N datasheet - http://www.st.com/stonline/books/ascii/docs/1773.htm

[39] - BYV27200 *Schottky diode* datasheet - http://www.vishay.com/diodes/list/product-86042/

[40] - PHIV459 datasheet - http://www.politronic.com.br/especs/div/PHIV459.pdf

[41] - PHFT458 datasheet - http://www.politronic.com.br/especs/div/PHIV459.pdf

[42] - 74HC14N datasheet - http://www.semiconductors.philips.com/pip/74HC14N.html

[43] - BC558 datasheet - http://www.fairchildsemi.com/pf/BC/BC558.html

ANEXO 1

CÓDIGO-FONTE DO PIC16F876A

```
;*****************************************************
;*                    FIRMWARE DO ROBÔ 2.0 DO TIME DE FUTEBOL DE ROBÔS DA FEI - ROBOFEI                    *
;*                    AUTOR: MURILO FERNANDES MARTINS
          *
;*                    DATA: 2004/2005
          *
;*****************************************************



;*****************************************************
;*                                        ARQUIVOS DE DEFINIÇÕES E CONFIGURAÇÕES                                    *
;*****************************************************
#INCLUDE <P16F876A.INC>                ;arquivo de definição do PIC 16F876A

;*          LINHA DE CONFIGURAÇÃO QUE SEGUE A SEQUÊNCIA DOS BITS DE CONFIGURAÇÃO DO MPLAB                    *
          __CONFIG _HS_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF & _BODEN_OFF & _LVP_OFF & _WRT_OFF & _DEBUG_OFF &
_CPD_OFF

;*          HS -> XTAL 20 MHZ - PWRTE_ON -> RESET AO LIGAR (72 MS) - BODEN_ON -> BROWN OUT (V MÍN=4V)
;*          WRT_OFF -> WRITE PROTECTION DA MEMÓRIA DE PROGRAMA DESABILITADO

          RADIX DEC



;*****************************************************
;*                                                  PAGINAÇÃO DE MEMÓRIA
          *
;*****************************************************
BANK0     MACRO
          BCF       STATUS,RP0
          BCF       STATUS,RP1
          BCF       STATUS,IRP          ; limpa bit para endereçamento indireto
          ENDM

BANK1     MACRO
          BSF       STATUS,RP0
          BCF       STATUS,RP1
          BCF       STATUS,IRP          ; limpa bit para endereçamento indireto
          ENDM

BANK2     MACRO
          BCF       STATUS,RP0
          BSF       STATUS,RP1
          BSF       STATUS,IRP          ; seta bit para endereçamento indireto
          ENDM

BANK3     MACRO
          BSF       STATUS,RP0
          BSF       STATUS,RP1
          BSF       STATUS,IRP          ; seta bit para endereçamento indireto
          ENDM

          ; macros para acessar páginas diretamente
PAGE0     MACRO
          BCF       PCLATH,3
          BCF       PCLATH,4
          ENDM
```

```
PAGE1   MACRO
        BSF     PCLATH,3
        BCF     PCLATH,4
        ENDM

PAGE2   MACRO
        BCF     PCLATH,3
        BSF     PCLATH,4
        ENDM

PAGE3   MACRO
        BSF     PCLATH,3
        BSF     PCLATH,4
        ENDM


;*****************************************************
;*                                                   MACROS                                                *
;*****************************************************
M1_BRAKEMACRO                           ;MACRO PARA LIGAR AS SAÍDAS PARA O MOTOR 1
        BSF     M1A                             ;RESULTANDO EM UM FREIO DINÂMICO NA PONTE H
        BSF     M1B                             ;QUE CONTROLA O MOTOR.
        ENDM

M1_OFF          MACRO                           ;MACRO PARA DESLIGAR MOTOR 1
        BCF     M1A                             ;ATRIBUINDO 0 AOS PINOS 5 E 7 DO L298N
        BCF     M1B
        ENDM

M1_CWISE MACRO                          ;MACRO PARA ATRIBUIR 1 AO PINO 5 DO L298N
        BSF     M1A                             ;E 0 AO PINO 7 FAZENDO COM QUE O MOTOR GIRE EM
        BCF     M1B                             ;SENTIDO HORÁRIO
        ENDM

M1_CCWISE       MACRO                           ;MACRO PARA ATRIBUIR 0 AO PINO 5 E 1 AO PINO 7
        BCF     M1A                             ;DA PONTE H PARA FAZER COM QUE O MOTOR GIRE
        BSF     M1B                             ;EM SENTIDO ANTI-HORÁRIO
        ENDM

M2_BRAKEMACRO                           ;FAZ O MESMO PARA O MOTOR 2
        BSF     M2A
        BSF     M2B
        ENDM

M2_OFF          MACRO                           ;MACRO PARA DESLIGAR MOTOR 2
        BCF     M2A                             ;ATRIBUINDO 0 AOS PINOS 10 E 12 DO L298N
        BCF     M2B
        ENDM

M2_CWISE MACRO                          ;FAZ O MESMO PARA O MOTOR 2 (PINOS 10 E 12
        BSF     M2A                             ;RESPECTIVAMENTE)
        BCF     M2B
        ENDM

M2_CCWISE       MACRO                           ;FAZ O MESMO PARA O MOTOR 2
        BCF     M2A
        BSF     M2B
        ENDM


;*****************************************************
;*                                                   VARIÁVEIS                                             *
;*****************************************************

        CBLOCK  0X20                            ;INÍCIO DO BLOCO DE MEMÓRIA
                POINTER                         ;PONTEIRO PARA OS ENDEREÇOS DOS BYTES DE DADOS
                BYTE1                           ;ENDEREÇO 0X21 DA RAM
                BYTE2                           ;ENDEREÇO 0X22 DA RAM
                BYTE3                           ;ENDEREÇO 0X23 DA RAM
```

```
                BYTE4                           ;ENDEREÇO 0X24 DA RAM
                BYTE5                           ;ENDEREÇO 0X25 DA RAM
                BYTE6                           ;ENDEREÇO 0X26 DA RAM
                LAST_BYTE                       ;CONTÉM ENDEREÇO+1 DO ÚLTIMO BYTE DE DADOS A SER RECEBIDO

                WHOAMI                          ;VARIÁVEL QUE CONTÉM A CONFIG. DO MODO DE OPERAÇÃO DO
ROBÔ
                BYTE_CHECK                      ;BYTE QUE CONTÉM BITS DE CONFERÊNCIA RECEBIDOS
                BYTE_CONF                       ;BYTE QUE CONTÉM BITS DE CONFERÊNCIA GERADOS
                VEL_ESQ                         ;VALOR DA VEL. ATUAL ATRIBUÍDA AO MOTOR DA RODA ESQUERDA
                VEL_DIR                         ;VALOR DA VEL. ATUAL ATRIBUÍDA AO MOTOR DA RODA DIREITA
                PULSOS_M1                       ;Nº DE PULSOS GERADOS PELA VEL. DO MOTOR 1 (ESQUERDO)
                PULSOS_M2                       ;Nº DE PULSOS GERADOS PELA VEL. DO MOTOR 2 (DIREITO)
                MEU_WDT                         ;O VALOR DESSA VARIÁVEL DETERMINA O MULTIPLICADOR QUE SERÁ
                                                ;UTILIZADO PARA GERAR O DESLIGAMENTO DOS MOTORES,
LEMBRANDO
                                                ;QUE O TEMPO TOTAL = APROX. 7,7 X MEU_WDT
                CONTADOR                        ;VARIÁVEL PARA PISCAR LED ATIVIDADE (F=1/(7,7 X CONTADOR))
                W_TEMP
                STATUS_TEMP                     ;REGISTRADORES TEMPORÁRIOS PARA USO COM AS INTERRUPÇÕES
                AUX                             ;BYTE AUXILIAR
                CONFLAGS                        ;REGISTRADOR COM FLAGS PARA O CONTROLE DO FLUXO DO
PROGRAMA
                TEMP
        ENDC                                    ;FIM DO BLOCO DE MEMÓRIA



;*****************************************************
;*                                                   CONTANTES
        *
;*****************************************************
START_BYTE      EQU     0X62                    ;BYTE DE INÍCIO REFERENTE AO PROTOCOLO DE COM. COM O PC
END_BYTE EQU    0X65                    ;BYTE DE FINALIZAÇÃO
PISCALED EQU    0X0D                    ;VALOR PARA CONTADOR (FREQ. DO LED ATIVIDADE)
RUN_TIME EQU    0X1A                    ;VALOR PARA MEU_WDT (TEMPO EM QUE MOTORES FICAM LIGADOS)


;*****************************************************
;*                                                   FLAGS INTERNOS                                                *
;*****************************************************

;---------------------------------------- CONFLAGS ----------------------------------------
;       *---------*---------*---------*---------*---------*---------*---------*---------*
;       | M1E_DIR | M2D_DIR | M1E_ANT | M2D_ANT |  EXEC  |  RUN   |RCV_START |MOTOR_RUN |
;       *---------*---------*---------*---------*---------*---------*---------*---------*

MOTOR_RUN       EQU     0                       ;0 -> MOTORES DESLIGADOS
                                                ;1 -> MOTORES FUNCIONANDO
RCV_START       EQU     1                       ;0 -> NÃO RECEBEU BYTE INICIAL
                                                ;1 -> BYTE INICIAL RECEBIDO
RUN             EQU     2                       ;0 -> NÃO RECEBEU COMANDO VÁLIDO
                                                ;1 -> RECEBEU COMANDO VÁLIDO PARA LIGAR MOTORES
EXEC            EQU     3                       ;0 -> BITS DE CONFERÊNCIA NÃO CONFEREM
                                                ;1 -> BITS CONFEREM. PODE ATRIBUIR VELOCIDADES
M1E_DIR         EQU     7                       ;0 -> SENTIDO HORÁRIO (ROTAÇÃO MOTOR 1 ESQUERDO)
                                                ;1 -> SENTIDO ANTI-HORÁRIO
M2D_DIR         EQU     6                       ;0 -> SENTIDO HORÁRIO (ROTAÇÃO MOTOR 2 DIREITO)
                                                ;1 -> SENTIDO ANTI-HORÁRIO
M1E_ANT         EQU     5                       ;0 -> ESTADO ANTERIOR DO SINAL DO SENSOR DO MOTOR 1
                                                ;1 -> ESTADO ANTERIOR DO SINAL DO SENSOR DO MOTOR 1
M2D_ANT         EQU     4                       ;0 -> ESTADO ANTERIOR DO SINAL DO SENSOR DO MOTOR 2
                                                ;1 -> ESTADO ANTERIOR DO SINAL DO SENSOR DO MOTOR 2

;*****************************************************
;*                                                   ENTRADAS
        *
;*****************************************************
```

```
;---------------------------------- ENTRADAS PARA MODO DE FUNCIONAMENTO ----------------------------------
#DEFINE   SWCON1  PORTA,0                         ;BIT MAIS SIGNIFICATIVO PARA SELECIONAR ENDEREÇO DO ROBÔ
#DEFINE   SWCON2  PORTA,1                         ;BIT MENOS SIGNIFICATIVO
                                                  ;00 = MODO TESTE
                                                  ;01 = ROBÔ 1
                                                  ;10 = ROBÔ 2
                                                  ;11 = ROBÔ 3
#DEFINE   M1_IN     PORTA,4
#DEFINE   M2_IN     PORTA,5


;*****************************************************
;*                                                   SAÍDAS
;*****************************************************
#DEFINE   M1A               PORTC,0               ;DEFINE PINOS DE SAÍDA PARA CONTROLAR SENTIDO DE ROTAÇÃO
#DEFINE   M1B               PORTC,3               ;DOS MOTORES.
#DEFINE   M2A               PORTC,4               ;LIGAR DIRETAMENTE NA PONTE H, PINOS 5, 7, 10 E 12
#DEFINE   M2B               PORTC,5               ;RESPECTIVAMENTE.

#DEFINE   LED               PORTB,5               ;LED INDICADOR DO TIMER0
#DEFINE   LED_OK            PORTB,4               ;LED INDICADOR DE CONFERÊNCIA - OK
#DEFINE   LED_ERR           PORTB,3               ;LED INDICADOR DE CONFERÊNCIA - ERRO



;*****************************************************
;*                                                VETOR DE RESET                                                *
;*****************************************************
          ORG       0X00
          GOTO      INICIO



;*****************************************************
;*                                                VETOR DE INTERRUPÇÃO
;*****************************************************

;---------------------------------------- SALVA CONTEXTO ATUAL ----------------------------------------
          ORG       0X04
          MOVWF     W_TEMP                        ;FAZ BACKUP DE W EM W_TEMP
          SWAPF     STATUS,W                ;FAZ BACKUP DE STATUS EM W
          MOVWF     STATUS_TEMP                   ;SALVA STATUS EM STATUS_TEMP

;------------------------ VERIFICAÇÃO DE QUAL PERIFÉRICO GEROU A INTERRUPÇÃO ----------------------------
          BTFSC     PIR1,RCIF               ;TESTA SE QUEM GEROU A INT. FOI A RECEPÇÃO SERIAL
          GOTO      TRATA_RX                ;SE RCIF=1, GEROU INT. E PULA PARA TRATAR

          BTFSC     INTCON,T0IF                   ;TESTA PARA VER SE FOI INT. DO TIMER 0
          GOTO      TRATA_TMR0                    ;SE T0IF=1, É INT. DO TIMER 0 E PULA PARA TRATÁ-LA

          GOTO      RET_INT


;*****************************************************
;*                                          TRATAMENTO DAS INTERRUPÇÕES                                        *
;*****************************************************

;---------------------------------------- INTERRUPÇÃO DO TIMER 0 ----------------------------------------
TRATA_TMR0
          BCF       INTCON,T0IF                   ;LIMPA FLAG DE INT. DO TIMER 0
          DECFSZ    CONTADOR,F                    ;DECREMENTA E PULA A PRÓX. INSTRUÇÃO SE FOR = ZERO
          GOTO      RET_TMR0

          BTFSC     LED                           ;VERIFICA SE LED=1, SE SIM EXECUTA PRÓX. INSTRUÇÃO
```

```
        GOTO    APAGA_LED                       ;APAGA LED, POIS ESTAVA LIGADO
        BSF     LED
        GOTO    SAI_TIMER0
APAGA_LED
        BCF     LED

SAI_TIMER0
        MOVLW   PISCALED
        MOVWF   CONTADOR                        ;RECARREGA CONTADOR
RET_TMR0
        BTFSC   CONFLAGS,MOTOR_RUN              ;TESTA SE BIT=1 (SIGNIFICA MOTORES LIGADOS)
        GOTO    RUNNING                         ;SIM, MOTORES LIGADOS (PULA PARA CONTAR O TEMPO)
        GOTO    RET_INT                         ;NÃO, SAI DA INTERRUPÇÃO

RUNNING
        DECFSZ  MEU_WDT,F                       ;DECREMENTA E PULA A PRÓX. INSTRUÇÃO SE FOR = ZERO
        GOTO    RET_INT                         ;NÃO ESTOUROU O TEMPO... CONTINUA... SAI DA INT.
                                                ;MEU_WDT=0 ENTÃO DESLIGA MOTORES
        M1_OFF                                  ;DESLIGA MOTOR 1 ESQUERDO
        M2_OFF                                  ;DESLIGA MOTOR 2 DIREITO
        BCF     CONFLAGS,MOTOR_RUN              ;DESLIGA FLAG QUE INDICA MOTORES LIGADOS
        BCF     LED_ERR
        BCF     LED_OK
        GOTO    RET_INT                         ;RETORNA DA INT.

;----------------------------------- INTERRUPÇÃO DE RECEPÇÃO SERIAL -----------------------------------
TRATA_RX
        MOVLW   0X06                            ;MÁSCARA PARA BITS (VERIFICAÇÃO DE ERROS)
        ANDWF   RCSTA,W                         ;CHECA SE HOUVE ERROS NA RECEPÇÃO
        BTFSS   STATUS,Z                ;TESTA SE Z=0 (SIGNIFICA QUE HOUVE ERRO)
        GOTO    RCV_ERROR                       ;SIM, HOUVE ERRO, PULA PARA TRATAR...
        MOVF    RCREG,W                         ;SEM ERROS. PEGA DADO RECEBIDO
        MOVWF   AUX                             ;SALVA DADO EM AUX

        BTFSC   CONFLAGS,RCV_START              ;TESTA SE FLAG=1 (SIGNIFICA QUE JÁ RECEBEU BYTE DE START)
        GOTO    EH_DADO                         ;SIM. ENTÃO AGORA É DADO. PULA PARA TRATAR...
        MOVF    AUX,W                           ;AINDA NÃO RECEBEU START. CARREGA W COM O DADO RECEBIDO
        XORLW   START_BYTE                      ;COMPARA COM START_BYTE
        BTFSC   STATUS,Z                ;TESTA SE Z=1 (SIGNIFICA QUE VALORES SÃO IGUAIS)
        GOTO    IGUAL                           ;PULA PARA TRATAR DADOS IGUAIS
        GOTO    RET_INT                         ;PULA PARA RETORNO

IGUAL
        BSF     CONFLAGS,RCV_START              ;LIGA FLAG PARA INDICAR QUE RECEBEU START BYTE

        MOVLW   'I'                             ;CARREGA W COM DADO PARA SER TRANSMITIDO...
        CALL    ENVIA_DADOS
        MOVLW   'G'                             ;CARREGA W COM DADO PARA SER TRANSMITIDO...
        CALL    ENVIA_DADOS
        MOVLW   'U'                             ;CARREGA W COM DADO PARA SER TRANSMITIDO...
        CALL    ENVIA_DADOS
        MOVLW   'A'                             ;CARREGA W COM DADO PARA SER TRANSMITIDO...
        CALL    ENVIA_DADOS
        MOVLW   'L'                             ;CARREGA W COM DADO PARA SER TRANSMITIDO...
        CALL    ENVIA_DADOS

        GOTO    RET_INT                         ;PULA PARA RETORNO

EH_DADO
        MOVF    LAST_BYTE,W                     ;CARREGA W COM O ENDEREÇO DO ÚLTIMO BYTE
        XORWF   POINTER,W                       ;COMPARA ENDEREÇO DE POINTER COM ENDEREÇO DE LAST_BYTE
        BTFSC   STATUS,Z                ;TESTA SE Z=1 (SIGNIFICA QUE SÃO IGUAIS)
        GOTO    TESTA_FIM                       ;SÃO IGUAIS. JÁ RECEBEU 6 BYTES. PULA PARA TESTAR SE É FIM
        MOVF    POINTER,W                       ;CARREGA W COM O ENDEREÇO DO PRÓXIMO BYTE DE DADOS
        MOVWF   FSR                             ;CARREGA FSR PARA ENDEREÇAMENTO INDIRETO DOS BYTES
        MOVF    AUX,W                           ;CARREGA DADO RECEBIDO EM W
        MOVWF   INDF                            ;SALVA NO ENDEREÇO ESPECIFICADO EM FSR O DADO RECEBIDO
        INCF    POINTER,F               ;INCREMENTA VALOR DE POINTER (ENDEREÇO DOS BYTES)

        GOTO    RET_INT
```

```
TESTA_FIM
        BCF     CONFLAGS,RCV_START              ;LIMPA FLAG DE RECEPÇÃO DE START BYTE
        MOVLW   0X21
        MOVWF   POINTER                         ;REINICIALIZA POINTER COM ENDEREÇO DO PRIMEIRO BYTE
        MOVF    AUX,W                           ;CARREGA EM W O DADO RECEBIDO PARA VERIFICAR SE É FIM
        XORLW   END_BYTE                ;COMPARA W (DADO RECEBIDO) COM O BYTE DE FINALIZAÇÃO
        BTFSC   STATUS,Z                ;TESTA SE Z=1 (SIGNIFICA QUE VALORES SÃO IGUAIS)
        BSF     CONFLAGS,RUN                    ;LIGA FLAG REFERENTE AO RECEBIMENTO DE COMANDO VÁLIDO
                                                ;SIGNIFICA QUE É FIM!!!
        MOVLW   'F'
        CALL    ENVIA_DADOS
        MOVLW   'I'
        CALL    ENVIA_DADOS
        MOVLW   'M'
        CALL    ENVIA_DADOS

        GOTO    RET_INT                         ;RETORNA

RCV_ERROR
        BCF     RCSTA,CREN                      ;LIMPA STATUS DO RECEBIMENTO
        BSF     RCSTA,CREN                      ;HABILITA RECEPÇÃO NOVAMENTE
        GOTO    RET_INT

;------------------------------------ RETORNO DA INTERRUPÇÃO ----------------------------------------
RET_INT
        SWAPF   STATUS_TEMP,W                   ;RECUPERA STATUS DE ANTES DA INTERRUPÇÃO EM W...
        MOVWF   STATUS                          ;E AGORA NO PRÓPRIO STATUS
        SWAPF   W_TEMP,F
        SWAPF   W_TEMP,W                        ;RECUPERA W DE ANTES DA INTERRUPÇÃO
        RETFIE


;*****************************************************
;*                                                   SUBROTINAS                                                *
;*****************************************************

;------------------------ SUBROTINA TEMPORÁRIA PARA ENVIO DE DADOS PARA O PC ----------------------------
ENVIA_DADOS
        MOVWF   TXREG                           ;TRANSMITE...
        CALL    TESTTX                          ;AGUARDA TRANSMISSÃO...
        MOVLW   0X0D
        MOVWF   TXREG                           ;TRANSMITE BYTE REFERENTE AO ENTER
        CALL    TESTTX
        MOVLW   0X0A                            ;TRANSMITE BYTE REFERENTE AO LINE FEED
        MOVWF   TXREG
        CALL    TESTTX

        RETURN

;-------------------------- SUBROTINA DE VERIFICAÇÃO DE BUFFER DE TRANSMISSÃO --------------------------
TESTTX
        BANK1
WAIT
        BTFSS   TXSTA,TRMT                      ;TESTA SE O BUFFER ESTÁ VAZIO
        GOTO    WAIT                            ;AINDA NÃO ESTÁ VAZIO, ESPERA ESVAZIAR
        BANK0                                   ;BUFFER VAZIO, SELECIONA BANCO 0 (PADRÃO)
        RETURN                                  ;RETORNA E FICA PRONTO PARA PRÓXIMA TRANSMISSÃO


;------------------------ SUBROTINA DE VERIFICAÇÃO DOS BITS DE CONFERÊNCIA ----------------------------
CHECA_BITS
        BCF     CONFLAGS,RUN                    ;LIMPA FLAG REFERENTE AO COMANDO
        MOVF    WHOAMI,W                        ;CARREGA W COM O ENDEREÇO DO 1º BYTE DO ROBÔ
        MOVWF   FSR                             ;CARREGA FSR PARA ENDEREÇAMENTO INDIRETO
CHECA_B2
        MOVF    INDF,W                          ;CARREGA W COM O VALOR DO 1º BYTE
        ANDLW   0X03                            ;MÁSCARA PARA LIMPAR BITS FICANDO SÓ OS DE CONFERÊNCIA
        MOVWF   BYTE_CHECK                      ;CARREGA BYTE_CHECK COM OS BITS DE CONFERÊNCIA
        CLRF    BYTE_CONF                       ;LIMPA DESTINO DE BITS A SEREM GERADOS
```

```
        BTFSS   INDF,7                  ;TESTA BIT7 = 0 DO BYTE DE DADOS (PARA GERAR CONFERÊNCIA)
        BSF     BYTE_CONF,1             ;SE SIM LIGA BIT 1 DE BYTE_CONF
        BTFSS   INDF,3                  ;TESTA BIT3 = 0 DO BYTE DE DADOS (PARA GERAR CONFERÊNCIA)
        BSF     BYTE_CONF,0             ;SE SIM LIGA BIT 0 DE BYTE_CONF
        MOVF    BYTE_CONF,W             ;CARREGA EM W OS BITS GERADOS...
        XORWF   BYTE_CHECK,W            ;COMPARA BITS GERADOS COM BITS RECEBIDOS
        BTFSS   STATUS,Z        ;TESTA SE Z=0 (SIGNIFICA QUE BITS NÃO CONFEREM)
        GOTO    ERROR_BITS              ;BITS NÃO CONFEREM, PULA PARA TRATAR ERRO
        MOVF    WHOAMI,W                ;CARREGA EM W ENDEREÇO DO 1º BYTE
        ADDLW   0X01                    ;INCREMENTA W (AGORA É ENDEREÇO DO 2º BYTE DO ROBÔ)
        XORWF   FSR,W                   ;COMPARA FSR COM ENDEREÇO DO 2º BYTE
        BTFSC   STATUS,Z        ;TESTA SE Z=1 (SIGNIFICA QUE SÃO IGUAIS)
        GOTO    OK_BITS                 ;SIM. PULA PARA TRATAR CONFERÊNCIA COM ÊXITO
        INCF    FSR,F                   ;SÃO DIFERENTES. INCREMENTA FSR PARA APONTAR PARA 2º BYTE
        GOTO    CHECA_B2        ;VOLTA PARA CHECAR O 2º BYTE


ERROR_BITS
        BCF     LED_OK                  ;APAGA LED INDICADOR DE SUCESSO NA CONFERÊNCIA
        BSF     LED_ERR                 ;ACENDE LED DE ERRO DE CONFERÊNCIA

;       MOVLW   A'E'                    ;CARREGA W COM CARACTERE 'E'
;       CALL    ENVIA_DADOS             ;TRANSMITE...
;       MOVLW   A'R'                    ;CARREGA W COM CARACTERE 'R'
;       CALL    ENVIA_DADOS             ;TRANSMITE...
;       MOVLW   A'R'                    ;CARREGA W COM CARACTERE 'R'
;       CALL    ENVIA_DADOS             ;TRANSMITE...
;       MOVLW   A'O'                    ;CARREGA W COM CARACTERE 'O'
;       CALL    ENVIA_DADOS             ;TRANSMITE...

        GOTO    RET_CHECK

OK_BITS
        BCF     LED_ERR                 ;APAGA LED INDICADOR DE ERRO DE CONFERÊNCIA
        BSF     LED_OK                  ;ACENDE LED DE SUCESSO NA CONFERÊNCIA

        BSF     CONFLAGS,EXEC           ;LIGA BIT REFERENTE A ATRIBUIÇÃO DE VELOCIDADES VÁLIDAS

;       MOVLW   A'O'                    ;CARREGA W COM CARACTERE 'O'
;       CALL    ENVIA_DADOS             ;TRANSMITE...
;       MOVLW   A'K'                    ;CARREGA W COM CARACTERE 'K'
;       CALL    ENVIA_DADOS             ;TRANSMITE...

        GOTO    RET_CHECK

RET_CHECK

;       MOVF    WHOAMI,W                ;CARREGA W COM ENDEREÇO DO 1º BYTE
;       MOVWF   FSR                     ;CARREGA ENDEREÇO EM FSR PARA ENDEREÇAMENTO INDIRETO
;       MOVF    INDF,W                  ;CARREGA W COM O BYTE DA RODA ESQUERDA DO ROBÔ 1
;       CALL    ENVIA_DADOS             ;ENVIA DE VOLTA O DADO...
;       INCF    FSR,F                   ;INCREMENTA PONTEIRO DE ENDEREÇO
;       MOVF    INDF,W                  ;CARREGA W COM O BYTE DA RODA DIREITA DO ROBÔ 1
;       CALL    ENVIA_DADOS             ;ENVIA DE VOLTA O DADO...

        RETURN

;---------------------- SUBROTINA DE CONFIGURAÇÃO DO MODO DE FUNCIONAMENTO DO ROBÔ ----------------------
SETMODE
        BTFSC   SWCON1                  ;TESTA SE CHAVE1 ESTÁ LIGADA
        BSF     WHOAMI,1        ;SE SIM LIGA BIT 1 DA VARIÁVEL WHOAMI (MSB)
        BTFSC   SWCON2                  ;TESTA SE CHAVE2 ESTÁ LIGADA
        BSF     WHOAMI,0        ;SE SIM LIGA BIT 0 DA VARIÁVEL WHOAMI (LSB)

        MOVLW   0X00
        XORWF   WHOAMI,W                ;COMPARA CONFIG. COM 00 (MODO TESTE)
        BTFSC   STATUS,Z        ;TESTA SE Z=1 (SIGNIFICA QUE ESSA É A CONFIG.)
        GOTO    MODO_TESTE              ;PULA PARA ATRIBUIR MODO TESTE
        MOVLW   0X01
        XORWF   WHOAMI,W                ;COMPARA CONFIG. COM 01 (MODO ROBO1)
```

```
        BTFSC   STATUS,Z                ;TESTA SE Z=1 (SIGNIFICA QUE ESSA É A CONFIG.)
        GOTO    MODO_ROBO1                      ;PULA PARA ATRIBUIR MODO ROBO1
        MOVLW   0X02
        XORWF   WHOAMI,W                        ;COMPARA CONFIG. COM 10       (MODO ROBO2)
        BTFSC   STATUS,Z                ;TESTA SE Z=1 (SIGNIFICA QUE ESSA É A CONFIG.)
        GOTO    MODO_ROBO2                      ;PULA PARA ATRIBUIR MODO ROBO2
        MOVLW   0X03
        XORWF   WHOAMI,W                        ;COMPARA CONFIG. COM 11 (MODO ROBO3)
        BTFSC   STATUS,Z                ;TESTA SE Z=1 (SIGNIFICA QUE ESSA É A CONFIG.)
        GOTO    MODO_ROBO3                      ;PULA PARA ATRIBUIR MODO ROBO3
        GOTO    RET_SET                         ;PULA PARA RETORNAR... SÓ CHEGA AQUI SE DEU ALGUM PROBLEMA

MODO_TESTE
        BANK1
        BCF     PIE1,RCIE               ;DESLIGA INTERRUPÇÃO DE RECEPÇÃO SERIAL
        BANK0
        BCF     RCSTA,CREN                      ;DESABILITA RECEPÇÃO SERIAL
        GOTO    RET_SET

MODO_ROBO1
        MOVLW   0X21
        MOVWF   WHOAMI                          ;CARREGA O ENDEREÇO DO PRIMEIRO BYTE DO ROBÔ 1
        GOTO    RET_SET

MODO_ROBO2
        MOVLW   0X23
        MOVWF   WHOAMI                          ;CARREGA O ENDEREÇO DO PRIMEIRO BYTE DO ROBÔ 2
        GOTO    RET_SET

MODO_ROBO3
        MOVLW   0X25
        MOVWF   WHOAMI                          ;CARREGA O ENDEREÇO DO PRIMEIRO BYTE DO ROBÔ 3
        GOTO    RET_SET

RET_SET
        RETURN

;----------------------- SUBROTINA PARA ATRIBUIÇÃO DOS VALORES DE VELOCIDADE --------------------------
SET_VEL
        BCF     CONFLAGS,EXEC                   ;LIMPA FLAG QUE INDICA QUE NOVAS VEL. DEVEM SER ATRIBUÍDAS

        MOVF    WHOAMI,W                        ;CARREGA W COM O ENDEREÇO DO 1º BYTE DO ROBÔ
        MOVWF   FSR                             ;CARREGA FSR PARA ENDEREÇAMENTO INDIRETO

        BTFSC   INDF,7                          ;TESTA SE BIT7 = 1 (SIGNIFICA QUE É SENTIDO ANTI-HORÁRIO)
        BSF     CONFLAGS,M1E_DIR        ;BIT7 = 1 ENTÃO LIGA FLAG DO SENTIDO DE ROT. DO MOTOR 1

        MOVF    INDF,W                          ;CARREGA W COM O VALOR DO 1º BYTE
        ANDLW   B'01111100'                     ;MÁSCARA PARA DEIXAR APENAS OS BITS DE VELOCIDADE
        MOVWF   VEL_ESQ                         ;CARREGA O VALOR DA VELOCIDADE MULT. POR 4 EM VEL_ESQ
        RRF     VEL_ESQ,F                       ;DIVIDE VEL. POR 2 (AGORA TEM-SE A VELOCIDADE X 2)
        RRF     VEL_ESQ,F                       ;DIVIDE VEL. POR 2 (POSICIONADA A PARTIR DO BIT 0)
        INCF    FSR,F                           ;INCREMENTA ENDEREÇO (APONTA PARA O 2º BYTE)

        BTFSC   INDF,7                          ;TESTA SE BIT7 = 1 (SIGNIFICA QUE É SENTIDO ANTI-HORÁRIO)
        BSF     CONFLAGS,M2D_DIR        ;BIT7 = 1 ENTÃO LIGA FLAG DO SENTIDO DE ROT. DO MOTOR 2

        MOVF    INDF,W                          ;CARREGA W COM O VALOR DO 2º BYTE
        ANDLW   B'01111100'                     ;MÁSCARA PARA DEIXAR APENAS OS BITS DE VELOCIDADE
        MOVWF   VEL_DIR                         ;CARREGA O VALOR DA VELOCIDADE MULT. POR 4 EM VEL_DIR
        RRF     VEL_DIR,F               ;DIVIDE VEL. POR 2 (AGORA TEM-SE A VELOCIDADE X 2)
        RRF     VEL_DIR,F               ;DIVIDE VEL. POR 2 (POSICIONADA A PARTIR DO BIT 0)

        MOVF    VEL_ESQ,W                       ;CARREGA EM W A VELOCIDADE
        CALL    VEL_DUTY                ;CONSULTA TABELA, QUE RETORNA EM W A % DUTY CYCLE
        MOVWF   VEL_ESQ                         ;GUARDA ESSE VALOR EM VEL_ESQ
        RLF     VEL_ESQ,F                       ;MULTIPLICA POR 2 PARA ATRIBUIR DIRETO EM CCPR1L
        MOVF    VEL_ESQ,W
        MOVWF   CCPR1L
```

```
        MOVF    VEL_DIR,W               ;CARREGA EM W A VELOCIDADE
        CALL    VEL_DUTY                ;CONSULTA TABELA, QUE RETORNA EM W A % DUTY CYCLE
        MOVWF   VEL_DIR                         ;GUARDA ESSE VALOR EM VEL_ESQ
        RLF     VEL_DIR,F               ;MULTIPLICA POR 2 PARA ATRIBUIR DIRETO EM CCPR2L
        MOVF    VEL_DIR,W
        MOVWF   CCPR2L

;OS MOTORES FUNCIONAM DE MODO INVERTIDO (SENTIDO DE ROTAÇÃO DE UM EM RELAÇÃO AO OUTRO) PELO FATO
;DE TER-SE UM MOTOR PARA CADA EIXO, SENDO QUE PARA MOVIMENTAREM AS RODAS PARA A MESMA DIREÇÃO
;É NECESSÁRIO QUE ELES RODEM EM SENTIDOS CONTRÁRIOS
        BTFSC   CONFLAGS,M1E_DIR        ;TESTA SE BIT=1 (SIGNIFICA SENTIDO ANTI-HORÁRIO)
        GOTO    M1_CW                           ;SIM. PULA PARA LIGAR SENTIDO ANTI-HORÁRIO DE M1
        M1_CCWISE                               ;SE NÃO LIGA SENTIDO HORÁRIO
        GOTO    M2_TEST                         ;VAI LIGAR MOTOR 2
M1_CW
        M1_CWISE                        ;LIGA M1 EM SENTIDO ANTI-HORÁRIO

M2_TEST
        BTFSC   CONFLAGS,M2D_DIR        ;TESTA SE BIT=0 (SIGNIFICA SENTIDO ANTI-HORÁRIO)
        GOTO    M2_CW                           ;SIM. PULA PARA LIGAR SENTIDO ANTI-HORÁRIO DE M2
        M2_CWISE                        ;SE NÃO LIGA SENTIDO HORÁRIO
        GOTO    RET_DIRECTION                   ;RETORNA
M2_CW
        M2_CCWISE                               ;LIGA M2 EM SENTIDO ANTI-HORÁRIO
RET_DIRECTION

        MOVLW   RUN_TIME
        MOVWF   MEU_WDT                         ;CARREGA VALOR PARA A CONTAGEM DE APROX. 200 MILISEGUNDOS
        BSF     CONFLAGS,MOTOR_RUN              ;LIGA BIT PARA INDICAR QUE MOTORES ESTÃO LIGADOS

        RETURN

;*****************************************************
;*                                   TABELA VELOCIDADE -> DUTY CYCLE                                   *
;*****************************************************
VEL_DUTY
        MOVWF   TEMP
        MOVLW   LOW     VEL_DUTY_T+1
        ADDWF   TEMP,F
        MOVLW   HIGH    VEL_DUTY_T
        BTFSC   STATUS,C
        ADDLW   0X01
        MOVWF   PCLATH
        MOVF    TEMP,W
VEL_DUTY_T
        MOVWF   PCL                                     ;DESVIA PARA ENDEREÇO REFERENTE A VEL. REAL E
RETORNA %
        RETLW   0X00                            ;SE V = 0 ENTÃO RETORNA 0
        RETLW   D'70'                           ;SE V = 1 ENTÃO RETORNA 70
        RETLW   D'71'                           ;SE V = 2 ENTÃO RETORNA 71
        RETLW   D'72'                           ;SE V = 3 ENTÃO RETORNA 72
        RETLW   D'73'                           ;SE V = 4 ENTÃO RETORNA 73
        RETLW   D'74'                           ;SE V = 5 ENTÃO RETORNA 74
        RETLW   D'75'                           ;SE V = 6 ENTÃO RETORNA 75
        RETLW   D'76'                           ;SE V = 7 ENTÃO RETORNA 76
        RETLW   D'77'                           ;SE V = 8 ENTÃO RETORNA 77
        RETLW   D'78'                           ;SE V = 9 ENTÃO RETORNA 78
        RETLW   D'79'                           ;SE V = 10 ENTÃO RETORNA 79
        RETLW   D'80'                           ;SE V = 11 ENTÃO RETORNA 80
        RETLW   D'81'                           ;SE V = 12 ENTÃO RETORNA 81
        RETLW   D'82'                           ;SE V = 13 ENTÃO RETORNA 82
        RETLW   D'83'                           ;SE V = 14 ENTÃO RETORNA 83
        RETLW   D'84'                           ;SE V = 15 ENTÃO RETORNA 84
        RETLW   D'85'                           ;SE V = 16 ENTÃO RETORNA 85
        RETLW   D'86'                           ;SE V = 17 ENTÃO RETORNA 86
        RETLW   D'87'                           ;SE V = 18 ENTÃO RETORNA 87
        RETLW   D'88'                           ;SE V = 19 ENTÃO RETORNA 88
        RETLW   D'89'                           ;SE V = 20 ENTÃO RETORNA 89
        RETLW   D'90'                           ;SE V = 21 ENTÃO RETORNA 90
        RETLW   D'91'                           ;SE V = 22 ENTÃO RETORNA 91
```

```
            RETLW   D'92'                   ;SE V = 23 ENTÃO RETORNA 92
            RETLW   D'93'                   ;SE V = 24 ENTÃO RETORNA 93
            RETLW   D'94'                   ;SE V = 25 ENTÃO RETORNA 94
            RETLW   D'95'                   ;SE V = 26 ENTÃO RETORNA 95
            RETLW   D'96'                   ;SE V = 27 ENTÃO RETORNA 96
            RETLW   D'97'                   ;SE V = 28 ENTÃO RETORNA 97
            RETLW   D'98'                   ;SE V = 29 ENTÃO RETORNA 98
            RETLW   D'99'                   ;SE V = 30 ENTÃO RETORNA 99
            RETLW   D'96'                   ;SE V = 31 ENTÃO RETORNA 96

;------------------------------------------------ ODOMETRIA ------------------------------------------------
ODOMETRIA
            BCF     CONFLAGS,M1E_ANT
            BCF     CONFLAGS,M2D_ANT        ;LIMPA FLAGS DE ESTADO ANTERIOR
            MOVLW   0XB0
            MOVWF   TMR1L
            MOVLW   0X3C
            MOVWF   TMR1H                   ;CARREGA VALOR NO TIMER 1 PARA GERAR INT. A CADA 10 MS
            CLRF    PULSOS_M1
            CLRF    PULSOS_M2               ;LIMPA VARIÁVEIS DE CONTAGEM DOS PULSOS DOS MOTORES

            BSF     T1CON,TMR1ON            ;INICIA A CONTAGEM (LIGA TIMER 1)

TESTA_M1
            BTFSC   M1_IN                   ;TESTA SENSOR DO MOTOR 1 (SE FOR 0 PULA A PRÓX. LINHA)
            GOTO    EH1_M1
            BTFSS   CONFLAGS,M1E_ANT        ;VERIFICA SE É A MESMA LEITURA...
            GOTO    TESTA_M2
            INCF    PULSOS_M1,F
            BCF     CONFLAGS,M1E_ANT
            GOTO    TESTA_M2
EH1_M1
            BTFSC   CONFLAGS,M1E_ANT
            GOTO    TESTA_M2
            INCF    PULSOS_M1,F
            BSF     CONFLAGS,M1E_ANT

TESTA_M2
            BTFSC   M2_IN
            GOTO    EH1_M2
            BTFSS   CONFLAGS,M2D_ANT
            GOTO    TESTA_FIM_TMR1
            INCF    PULSOS_M2,F
            BCF     CONFLAGS,M2D_ANT
            GOTO    TESTA_FIM_TMR1
EH1_M2
            BTFSC   CONFLAGS,M1E_ANT
            GOTO    TESTA_FIM_TMR1
            INCF    PULSOS_M2,F
            BSF     CONFLAGS,M1E_ANT

TESTA_FIM_TMR1
            BTFSC   PIR1,TMR1IF
            GOTO    FIM_LEITURA
            GOTO    TESTA_M1

FIM_LEITURA
            BCF     T1CON,TMR1ON
            MOVF    PULSOS_M1,W
            SUBWF   PULSOS_M2,W
            BTFSC   STATUS,Z
            GOTO    RET_ODO
            BTFSC   STATUS,C
            GOTO    V1_MENOR_V2
            INCF    CCPR2L,F
            GOTO    RET_ODO
V1_MENOR_V2
            INCF    CCPR1L,F

RET_ODO
```

```
        RETURN

;-------------------------------------------------------------------------------------------

;*****************************************************
;*                                                   PROGRAMA
        *
;*****************************************************
INICIO
        BANK0
        MOVLW   0X00
        MOVWF   INTCON                          ;DESLIGA INTERRUPÇÕES PARA CONFIG. DOS PERIFÉRICOS

;---------------------------------- CONFIGURAÇÃO DOS PORTS, DO TIMER 0 E DO TIMER 1 ----------------
        CLRF    PORTA
        CLRF    PORTB
        CLRF    PORTC                           ;LIMPA OS LATCHES DOS PORTS
        MOVLW   0X6A
        MOVWF   TMR0                            ;CARREGA VALOR 106 NO REGISTRADOR DO TIMER 0 PARA DIVISÃO
                                                ;POR 150 PARA GERAR APROX. 130 INTERRUPÇÕES POR SEGUNDO
        MOVLW   0X00
        MOVWF   T1CON                           ;TIMER 1 COM CLOCK INTERNO, PRESCALER 1:1, DESLIGADO
        MOVLW   0XB0
        MOVWF   TMR1L                           ;CARREGA BYTE MENOS SIGNIFICATIVO
        MOVLW   0X3C
        MOVWF   TMR1H                           ;CARREGA BYTE MAIS SIGNIFICATIVO
                                                ;0X3CB0 = 15536 PARA GERAR INTERRUPÇÕES A CADA 10 MS

        BANK1
        MOVLW   0X06
        MOVWF   ADCON1                          ;DESLIGA A/D'S
        MOVLW   0X07
        MOVWF   CMCON                           ;DESLIGA COMPARADORES INTERNOS
        MOVLW   0X87
        MOVWF   OPTION_REG                      ;DESLIGA PULL-UP'S DO PORTB E CONFIGURA PRESCALER 1:256
        MOVLW   0XF3
        MOVWF   TRISA                           ;CONFIGURA PINOS DO PORTA: 1, 2, 6 E 7 = ENTRADA
        MOVLW   0X00
        MOVWF   TRISB                           ;CONFIGURA PINOS DO PORTB: TODOS OS PINOS COMO SAÍDA
        MOVLW   0X80
        MOVWF   TRISC                           ;CONFIGURA PINOS DO PORTC: PINO 18 (RX) = ENTRADA, APENAS

;---------------------------------- CONFIGURAÇÃO DOS PWM'S ------------------------------------
        MOVLW   0XC7                            ;ATRIBUI PERÍODO DO SINAL PWM CARREGANDO O VALOR EM PR2
        MOVWF   PR2                             ;PARA GERAR A FREQ. DESEJADA (PWM UTILIZA O TIMER2)
        BANK0
        MOVLW   0X0F
        MOVWF   CCP1CON                         ;CONFIGURA CCP1 PARA MODO PWM
        MOVLW   0X0F
        MOVWF   CCP2CON                         ;CONFIGURA CCP2 PARA MODO PWM
;       MOVLW   0X00
        MOVLW   0X96
        MOVWF   CCPR1L                          ;CONFIGURA DUTY CYCLE DO PWM 1 = 0%
;       MOVLW   0X00
        MOVLW   0X96
        MOVWF   CCPR2L                          ;CONFIGURA DUTY CYCLE DO PWM 2 = 0%
        CLRF    TMR2                            ;LIMPA CONTADOR DO TIMER 2
        MOVLW   0X04
        MOVWF   T2CON                           ;ATRIBUI PRESCALE = 1:1 E LIGA TIMER 2

;---------------------------------- CONFIGURAÇÃO DA USART -------------------------------------
        CLRF    TXREG                           ;LIMPA BUFFER DE TRANSMISSÃO
        CLRF    RCREG                           ;LIMPA BUFFER DE RECEPÇÃO
        CLRF    PIR1
        CLRF    PIR2                            ;LIMPA FLAGS DE INTERRUPÇÃO DE PERIFÉRICOS
        BANK1
        MOVLW   0X24                            ;HABILITA TRANSMISSÃO, 8 BITS DE DADOS DE TX, ASSÍNCRONO,
        MOVWF   TXSTA                           ;BAUD RATE ALTO (BRGH=1)
        MOVLW   0X81
        MOVWF   SPBRG                           ;CARREGA VALOR PARA TAXA DE 9600 BPS
```

```
        BANK0
        MOVLW   0X90                    ;HABILITA PORTA SERIAL, 8 BITS DE DADOS DE RX,
        MOVWF   RCSTA                   ;RECEPÇÃO CONTÍNUA

;-------------------------------------------------------------------------------------------------
        BANK1
        MOVLW   0X20                    ;HABILITA INTERRUPÇÃO DE RX APENAS
        MOVWF   PIE1                    ;E DESABILITA AS RESTANTES
        MOVLW   0X00
        MOVWF   PIE2                    ;DESABILITA AS OUTRAS INTERRUPÇÕES DE PERIFÉRICOS
        MOVLW   0XE0
        MOVWF   INTCON                  ;HABILITA INTERRUPÇÕES GERAIS, PERIFÉRICOS E TIMER 0
        BANK0

;---------------------------------- INICIALIZAÇÃO DAS VARIÁVEIS ----------------------------------
        MOVLW   PISCALED
        MOVWF   CONTADOR                ;CARREGA CONTADOR PARA PISCAR LED ATIVIDADE COM F=10 HZ
        MOVLW   RUN_TIME
        MOVWF   MEU_WDT                 ;CARREGA MEU_WDT COM 26 DECIMAL PARA CONTAR UM TEMPO DE
                                        ;APROX. 200MS DESLIGAR MOTORES
        BCF     LED                     ;APAGA LED
        BCF     LED_OK                  ;APAGA LED_OK - INDICADOR DE BITS DE CONFERÊNCIA OK
        BCF     LED_ERR                 ;APAGA LED_ERR - INDICADOR DE ERRO DE BITS DE CONFERÊNCIA
        CLRF    CONFLAGS                ;LIMPA FLAGS DO SOFTWARE
        CLRF    WHOAMI                  ;LIMPA VARIÁVEL DE CONFIG. DE MODO DE OPERAÇÃO

        MOVLW   0X21
        MOVWF   POINTER                 ;CARREGA POINTER COM O ENDEREÇO DO BYTE1

        MOVLW   0X27                    ;ENDEREÇO DO ÚLTIMO BYTE A SER RECEBIDO +1
        MOVWF   LAST_BYTE               ;CARREGA NA VARIÁVEL VALOR PARA A COMPARAÇÃO...

        M1_OFF                          ;DESLIGA MOTOR1 (COLOCA 0 NAS ENTRADAS 5 E 7 DO L298N)
        M2_OFF                          ;DESLIGA MOTOR2 (COLOCA 0 NAS ENTRADAS 10 E 12 DO L298N)

        BSF     PORTB,7
        BSF     PORTB,6

;---------------------------------- ROTINA PRINCIPAL DO PROGRAMA ----------------------------------
        CALL    SETMODE                 ;SUBROTINA CHAMADA APENAS NA INICIALIZAÇÃO DO SISTEMA
                                        ;PARA DEFINIR MODO DE OPERAÇÃO DO ROBÔ

MAIN
        NOP

        BTFSC   CONFLAGS,RUN            ;TESTA SE FLAG=1 (SIGNIFICA QUE RECEBEU COMANDO VÁLIDO)
        CALL    CHECA_BITS              ;CHECA BITS DE CONFERÊNCIA...

        BTFSC   CONFLAGS,EXEC           ;TESTA SE FLAG=1 (SIGNIFICA QUE PODE EXECUTAR VELOCIDADES)
        CALL    SET_VEL                 ;ATRIBUI VELOCIDADES...

        BTFSC   CONFLAGS,MOTOR_RUN      ;TESTE SE FLAG=1 (SIGNIFICA QUE MOTORES ESTÃO LIGADOS)
        CALL    ODOMETRIA               ;RODA A ODOMETRIA...

        NOP
        GOTO    MAIN


;*************************************************
;*                                               FIM
        *
;*************************************************

        END
```