DESENVOLVIMENTO DE UM ROBÔ COM RODAS AUTÔNOMO

Salomão Gonçalves de Oliveira Junior

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____
Prof. Antonio Carneiro de Mesquita Filho, Dr. d'État.

_____
Prof. Jorge Lopes de Souza Leão, Dr. Ing.

_____
Prof. Aloysio de Castro Pinto Pedroza, Dr

_____
Prof. José Franco Machado do Amaral, D. Sc.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 2008

OLIVEIRA JR, SALOMÃO GONÇALVES DE
Desenvolvimento De Um Robô Com Rodas Autônomo [Rio de Janeiro] 2008
XIII, 122p. 29,7 cm (COPPE/UFRJ, M.Sc, Engenharia Elétrica, 2008)
Dissertação - Universidade Federal do Rio de Janeiro, COPPE
1. Robótica Móvel
2. Sistema de Navegação
3. Autônomo
4. Tempo Real
I. COPPE/UFRJ II. Título (série)

Dedico este trabalho a minha esposa Sydia, minhas filhas Stela e Samya que são dádivas de Deus para minha vida e aos meus pais Sr Salomão e Sra Zenete que sempre me apoiaram e incentivaram.

AGRADECIMENTOS

Primeiramente a Deus por estar comigo em todas as situações proporcionando proteção e direção.

A meus irmãos incentivadores Roberto Charles Feitosa de Oliveira (Pós Doutor em Filosofia) e Carlos Zarden Feitosa de Oliveira (Mestre em Medicina Veterinária).

A todos os professores do ensino fundamental, ensino médio e graduação que foram os elementos primordiais na minha formação geral e técnica.

Ao meu orientador Antonio Carneiro Mesquita por ter sempre indicado de maneira clara e objetiva o melhor caminho a ser seguido e por me incentivar a realizar este trabalho.

Ao meu coorientador Jorge Lopes de Souza Leão pelas orientações na programação dos softwares utilizados neste trabalho e na interface com as outras áreas disciplinares.

Ao Sr José Augusto Ramos do Amaral, da Superintendência de Operação Técnica (SO.T), ao Sr Fernando Futuro e Sr Antonio Carlos de Oliveira Afonso, da Gerência de Modificações de Projeto (GMP.T), da Eletronuclear, por ter me permitido cursar o mestrado.

Aos meus colegas e amigos do grupo de Instrumentação e Controle da GMP.T que me incentivaram e ajudaram a chegar até aqui.

Ao meu amigo Marcio Luiz da Fonseca que muito me ajudou na elaboração deste trabalho.

A minha colega de pesquisa Sonia Cristina Bastos.

Muito obrigado a todos que contribuíram para que eu pudesse alcançar esta vitória.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DESENVOLVIMENTO DE UM ROBÔ COM RODAS AUTÔNOMO

Salomão Gonçalves de Oliveira Junior

Abril/2008

Orientadores: Antonio Carneiro de Mesquita Filho
Jorge Lopes de Souza Leão.

Programa: Engenharia Elétrica

O objetivo principal deste trabalho é projetar, construir e avaliar um robô autônomo com sistema de direção diferencial, ou seja, com duas rodas motoras e uma roda livre. O robô foi projetado para navegar com segurança em um ambiente conhecido com obstáculos. A seqüência de movimentos para execução da trajetória é recebida através de comunicação por rádio freqüência. Um computador portátil planeja a trajetória utilizando técnicas de inteligência artificial, definindo o melhor caminho com base em um mapa ou planta baixa da área industrial, e através de um ponto de origem e um ponto de destino previamente estabelecido. O problema do erro odométrico, acumulado a todo tempo, devido a não utilização de informação sensorial externa, foi avaliado. Técnicas de controle de velocidade em malha fechada das rodas do robô foram utilizadas.

Os resultados mostraram que o robô é capaz de executar o percurso com estabilidade e precisão.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Sciences (M.Sc.)

DEVELOPMENT OF AN AUTONOMUS WHEELED MOBILE ROBOT

Salomão Gonçalves de Oliveira Junior

April/2008

Advisors: Antonio Carneiro de Mesquita Filho
Jorge Lopes de Souza Leão.

Department: E Engineering

The aim of this dissertation is project, constructing and analyzing an autonomous robot with a differential system, with two wheels and a castor. The robot was projected to navigate safety at a known environment with obstacles. A sequence of movements to execute the trajectory is received through radio frequency communication. A personal computer do a path planning using Artificial Intelligence techniques, choosing the best path based on a map or an industrial floor plant, and through a starting position to a goal point previously established. The odometry error problem, that grows continuously, is verified and analyzed due the absence of external sensors. Advanced techniques of velocity closed loop control of the wheels were used.

The results showed that the robot is capable to follow the path with stability and precision.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

Os robôs móveis com rodas (WMR-Wheeled Mobile Robots) autônomos têm sido utilizados para realizar uma grande variedade de tarefas, permitindo aplicações que vão desde as associadas à substituição do homem em ambientes hostis até a busca de amostras de meteoritos na Antártica [1].

LEONARD *et al.* [2] resumiram o problema geral de navegação de robôs móveis em três questionamentos: Onde estou?, Para onde estou indo? e Como devo chegar lá? A navegação pode ser definida por três competências fundamentais: Autolocalização, Planejamento da trajetória e Construção e interpretação de mapas.

Autolocalização denota a capacidade de determinar/estabelecer sua própria posição em relação a um sistema de referência. Para aplicações externas (outdoors), é comum a utilização do Sistema de Navegação Global por Satélite (SNGS). Com equipamentos caros, é possível determinar uma posição na superfície da terra com precisão de poucos centímetros. Entretanto, receptores comerciais têm precisão na ordem de 5 a 100 metros. Além disso, é necessária uma linha de visada entre a posição na superfície da terra e o satélite. Em áreas florestais ou urbanas com prédios, o sinal recebido tende a sofrer degradação podendo haver o seu bloqueio, ou seja, a qualidade da estimativa de posição é prejudicada. Uma outra desvantagem é a baixa taxa de atualização que usualmente é de 1 segundo. Nas aplicações como corrida de automóveis, treinamento de direção, exercícios de direção, ou medidas de performance, uma estimativa de posição mais freqüente é exigida com o objetivo de registrar uma trajetória precisa. Somado a esta exigência, estão as informações desejadas de velocidade e atitude do veículo. O Sistema de Navegação Inercial (INS-Inertial Navigation System) pode fornecer a estimativa destas informações. Com base em acelerômetros e giroscópios, e nas leis de movimento de Newton, é possível determinar a posição, velocidade, e atitude se os valores iniciais são conhecidos. Os sensores podem ser amostrados com uma freqüência mais alta (tipicamente 100 Hz para os dispositivos comerciais). Este sistema é chamado de auto-contido [3].

Os sensores inerciais são classificados como sensores *dead reckoning* uma vez que a avaliação do estado atual é formado pelo incremento relativo dos estados conhecidos prévios. Devido a esta integração, erros como ruído em baixa freqüência e

imperfeições do sensor são acumulados, levando a um desvio na posição com erro crescente e ilimitado.

Uma combinação de ambos sistemas, SNGS com precisão de longo prazo e INS com precisão de curto prazo, pode fornecer uma boa precisão com taxas freqüentes para as aplicações externas. O Sistema de Posicionamento Global (GPS-Global Position System) faz com que o INS possua um erro limitado, e ao contrário, o INS pode ser usado para identificar e corrigir os escorregamentos de fase cíclica da portadora.

Em aplicações internas (indoors) como salas, prédios, galpões industriais, etc... com áreas de até 1000m^2 o sistema INS isoladamente, pode ser empregado apresentando resultados relativamente satisfatórios.

Existem diversas categorias de INS quanto à qualidade. Sistemas comerciais de navegação INS apresentam um erro de posição de aproximadamente 30m/min mas custam mais de US$ 100000. Sistemas INS de baixo custo por sua vez, apresentam um erro de posição de até 3000m/min e custam menos de US$ 1000,00. Recentemente os avanços nos sistemas Micro Mecânicos e Elétricos levam a sensores muito baratos, comparados com os sensores de navegação ou táticos. Entretanto, sua precisão, polarização, e características de ruído são de uma qualidade inferior em relação às outras categorias. Existe um esforço em se melhorar a precisão com as técnicas de processamento de sinal avançado [3].

O projeto do WMR deste trabalho utiliza sensores do tipo encoders adaptados diretamente ao eixo do motor das rodas esquerda e direita, para a medição da posição relativa (odometria). O projeto do software embarcado no dispositivo de controle, inclui o sistema Fundo (nível *tarefa)*/Frente (nível *de interrupção*) ou superlaços, que efetua o controle em malha fechada das duas rodas motoras e recebe via rádio freqüência a seqüência de trinca de números, que são trechos da trajetória em arcos de círculo a serem executados pelo robô.

Devido a disponibilidade e versatilidade do kit de desenvolvimento do fabricante Freescale Semiconductor, modelo DSK (Kit inicial de desenvolvimento) 13213 SRB, este foi utilizado como dispositivo de controle do robô. O kit é constituído de uma placa de circuito eletrônico que inclui o Microcontrolador MC13213, sensores, portas de entrada e saída digital e analógica, interface de comunicação com PC por cabo e por rádio freqüência.

O planejamento da trajetória requer a determinação da posição de origem e da posição de destino em relação a um sistema de referência. Essas duas competências requerem, normalmente, o uso de uma representação do ambiente (mapa) e a habilidade de interpretar esta representação. No presente trabalho, o planejamento da

trajetória é executado em um PC, em aplicativo com interface gráfica, que permite inserir obstáculos e as coordenadas de origem e destino do robô. A trajetória resultante, seqüência de trincas de números, é realizada através do algorítmo *A Estrela* e enviada via sinal de rádio freqüência para o robô.

É um tema relevante pois é esperado que possa ser utilizado como ponto de partida, para o desenvolvimento de robô autônomo para aplicações específicas em área industrial.

1.2. OBJETIVO

O objetivo do trabalho é descrever o método e os procedimentos para o projeto de um robô terrestre, com duas rodas motoras e uma roda livre, autônomo, assim como avaliar a precisão de execução de trajetória estabelecida, com base em um mapa representativo de um ambiente plano com obstáculos.

1.3. ORGANIZAÇÃO DO TRABALHO

O capítulo 2 é destinado a apresentação de uma revisão bibliográfica do modelo cinemático e dinâmico do robô móvel com rodas e algumas estratégias de controle comumente empregadas.

Os capítulos 3 e 4 descrevem as características construtivas de hardware e software, que são consideradas como pontos de partida para o desenvolvimento do robô.

O capítulo 5 descreve o Sistema de Navegação, abordando o algoritmo que permite realizar o planejamento de trajetória.

Finalizando o processo de avaliação do robô, no capítulo 6 foram apresentados os resultados dos erros odométricos obtidos. No capítulo 7 são descritas as conclusões e observações, tendo como anexo uma base de informações destinada como sugestão de melhorias e aperfeiçoamento futuros.

2. REVISÃO BIBLIOGRÁFICA

2.1. CLASSIFICAÇÃO DOS ROBÔS

Diversas metodologias têm sido utilizadas para classificar robôs, contudo, não há, ainda, nenhuma que seja definitiva. De uma forma geral, podemos agrupar os robôs existentes de acordo com três aspectos: anatomia, tipo de controle e funcionalidade [4].

2.1.1. Anatomia

Quanto à anatomia os robôs podem ser classificados em três grandes grupos: os robôs aéreos, os aquáticos e os terrestres.

Os robôs aéreos, geralmente são aeromodelos ou LTA's (Lighter-Than-Air) equipados com câmeras de vídeo e utilizados para inspeção de grandes áreas. Os aquáticos, em geral são plataformas equipadas com propulsores e balões de ar que permitem ao robô permanecer a alguns metros do fundo de rio, represa, mar ou tanque. Já os robôs terrestres são os mais populares e podem utilizar três tipos diferentes de atuadores: rodas, esteiras ou pernas.

Os robôs com rodas são os mais simples, pois não necessitam de um hardware tão complexo quanto os robôs com esteiras e pernas. A principal desvantagem no uso de rodas é que, em terrenos irregulares, o desempenho pode não ser satisfatório. Em geral, a roda do robô deve possuir raio igual ou maior que a dos obstáculos que ele irá transpor.

Robôs com esteiras são mais utilizados em ambientes irregulares, como por exemplo, em solo fofo e pedras. Sua grande desvantagem em relação aos robôs com rodas está na dissipação de energia causada pelo atrito das rodas com a esteira e desta com o terreno.

Robôs com pernas são utilizados em terrenos acidentados, com subidas íngremes ou em ambientes específicos, como por exemplo a mudança de um plano através de escadas. A grande dificuldade deste tipo de robô está no desenvolvimento do projeto para controle das pernas, apresentando no mínimo dois graus de liberdade associado a sua estabilidade física, através do controle do seu centro de gravidade. Há também o fator custo associado a este tipo de projeto, pois cada atuador utiliza pelo menos dois motores.

É importante ressaltar que estas são categorias difusas, uma vez que podem ser combinadas características de dois grupos na construção de um robô para uma tarefa específica.

2.1.2. Tipo de controle

Segundo o tipo de controle, os robôs podem ser classificados em três categorias: Teleoperados, Semi-Autônomos e Autônomos. Nos robôs Teleoperados, o operador comanda todos os movimentos do robô. Nos Semi-Autônomos, o operador indica o macro comando a ser executado e o robô o faz sozinho. Nos Autônomos o robô realiza a tarefa sozinho, tomando suas próprias decisões baseando-se nos dados obtidos do ambiente.

2.1.3. Funcionalidade

Ao agruparmos os robôs segundo sua funcionalidade, identifica-se quatro grupos: robôs industriais, robôs de serviço, robôs de campo e robôs pessoais. Contudo, há uma sobreposição entre os três primeiros, devido à diferença dos ambientes onde atuam e a necessidade de maior autonomia.

Robôs Industriais são utilizados em linhas de produção. Estes robôs recebem tarefas determinadas *a priori* na forma de uma seqüência explícita de ações e executam este programa automaticamente. O ambiente é completamente estruturado e ajustado, para a execução da tarefa, tendo o robô o conhecimento exato da sua posição e da posição dos objetos. Em geral, robôs industriais são plataformas móveis utilizadas para tarefas pesadas, como transporte de materiais e produtos finais em sistemas de manufatura.

Os robôs de serviço industrial possuem as características de um robô de serviço, porém atuam em um ambiente completamente estruturado.

Os robôs de serviço de campo atuam em ambientes externos que podem ser previamente modelados ou não. Em geral, caso haja um modelo, este é precário e há a necessidade do processamento sensorial para complementar o modelo existente. São utilizados na realização de tarefas agrícolas e para navegação em auto-estradas.
Os robôs de campo trabalham em ambientes não estruturados, pouco conhecidos e geralmente perigosos. Suas principais atividades são a exploração (espacial, de cavernas, vulcões), a mineração e a limpeza de acidentes nucleares.

Os robôs pessoais são aqueles vendidos em prateleiras, que não desenvolvem tarefas específicas, mas interagem com os humanos e aprendem a localizar-se no ambiente.

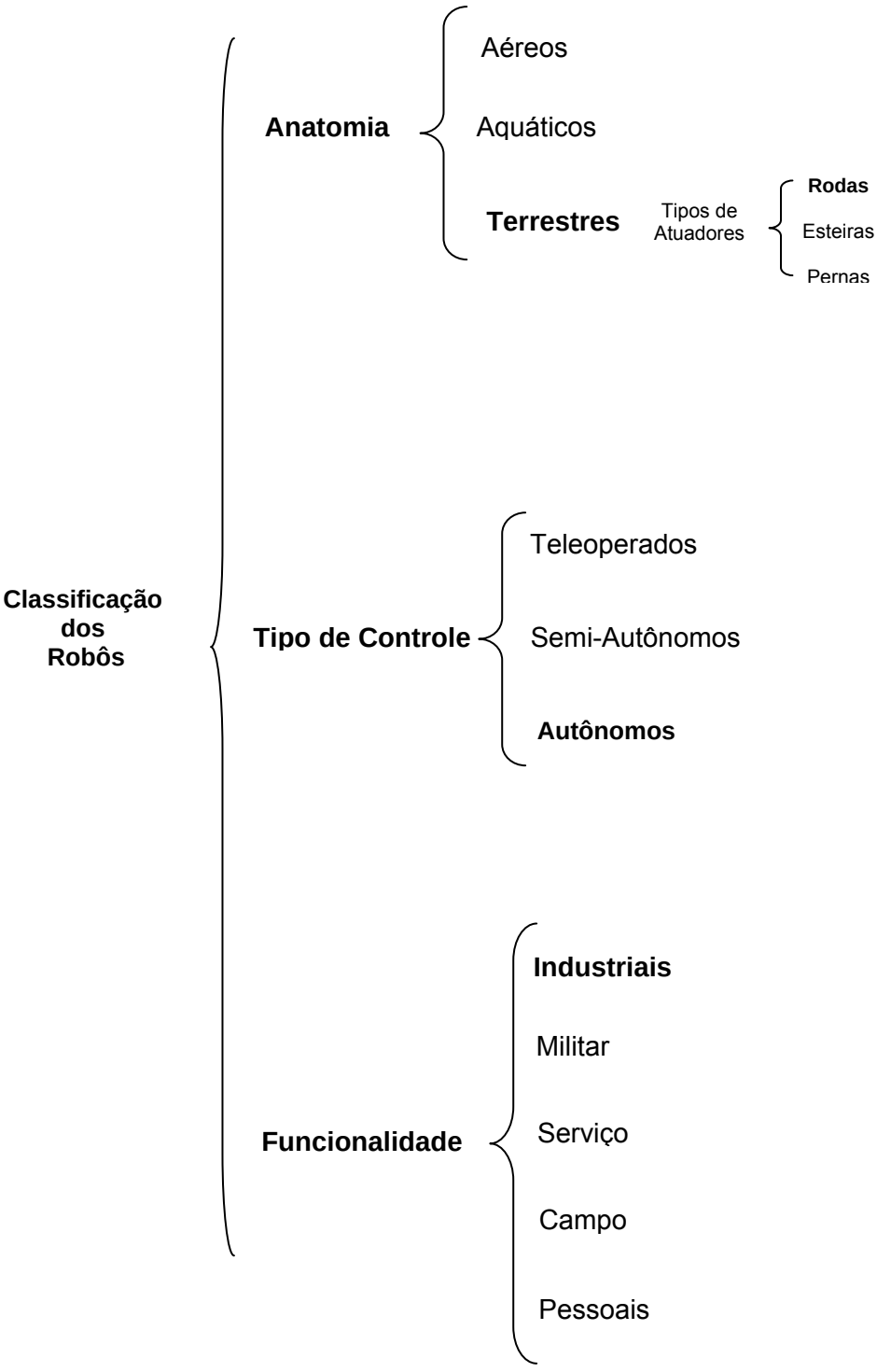

Figura 2.1.1 Classificação dos Robôs.

No presente trabalho o robô projetado e construído é classificado quanto à anatomia como terrestre, com atuadores do tipo rodas. Quanto ao tipo de controle, como autônomo e quanto a desejável funcionalidade como tipo industrial.

2.2. MODELO CINEMÁTICO

A cinemática do robô estuda o movimento deste com respeito a um sistema de referência [5]. A cinemática se interessa pela descrição analítica do movimento espacial do robô como uma função do tempo, em particular pelas relações entre a posição e a orientação do extremo do robô (a extremidade de um braço manipulador, por exemplo).

Existem basicamente dois problemas fundamentais a serem resolvidos na cinemática do Robô. O primeiro deles chamado **problema cinemático direto** consiste em determinar qual a posição e orientação do extremo final do robô em relação a um sistema de coordenadas de referência, conhecendo-se os valores das articulações e os parâmetros geométricos dos elementos dos robôs. O segundo denominado **problema cinemático inverso** deve resolver o caminho inverso, ou seja, dada uma posição e orientação do extremo do robô, deve-se descobrir o valor das coordenadas das articulações.

Além disso, a cinemática do robô tem como objetivo encontrar as relações entre a velocidade do movimento das articulações e dos seus extremos. Esta relação é dada pelo modelo diferencial, expresso através da matriz jacobiana.

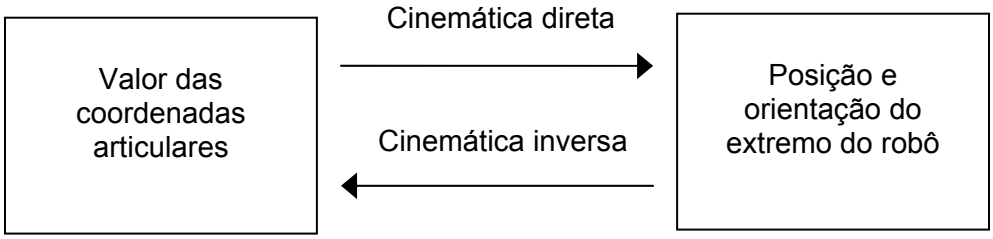

Figura 2.2.1 Diagrama de relação entre cinemática direta e inversa.

O SDD (Sistema de Direção Diferencial) talvez seja o sistema de direção mais simples para um robô autônomo terrestre [6]. Ele consiste em duas rodas montadas num eixo comum e controladas por motores independentes, um para cada roda. No SDD, para que cada roda tenha seu movimento, é necessário que o robô gire em torno de um ponto (ICC) localizado necessariamente na direção do eixo comum às duas rodas. Variando a velocidade relativa de duas rodas, o ponto de giro pode ser alterado, correspondendo a trajetórias diferentes. Em cada instante, o ponto de giro do robô tem

como propriedade o fato de que a roda esquerda e a roda direita seguem caminhos que se movem em torno de ICC à mesma velocidade angular ω.

Este sistema não leva em consideração a roda livre, que provê o equilíbrio do robô e estabelece o seu centro de gravidade em outro ponto diferente. Assumindo condições ideais, o contato entre a roda e o plano horizontal é reduzido a um único ponto durante o movimento. As rodas estão fixas, sendo que o plano de cada roda permanece na vertical e ocorre a rotação em torno do seu eixo horizontal. São assumidos também que não ocorrem derrapagem ou escorregamento das rodas e estas não se deformam e se movem em um plano horizontal não deformável. Isto significa que a velocidade do ponto de contato entre cada roda e o plano horizontal é igual a zero. Para velocidade de rolagem baixa este é um modelo de movimentação razoável. Com base na Figura 2.2.2, teremos então:

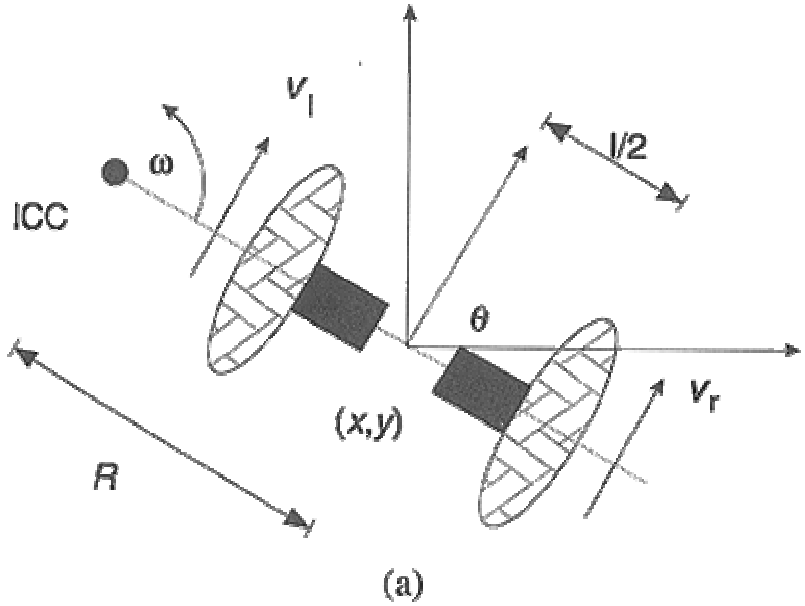

Figura 2.2.2 Sistema de Direção Diferencial.

$$V_r = \omega(R + \frac{l}{2}) \qquad (2.2.1) \qquad\qquad V_l = \omega(R - \frac{l}{2}) \qquad (2.2.2)$$

Onde:

V_l – Velocidade angular da roda esquerda;

V_r – Velocidade angular da roda direita;

ω – ângulo entre segmento de reta formado pelo eixo diferencial e o ponto de referência ICC;

R – raio entre (x , y) e ponto de referência ICC;

l/2 – distância metade entre as rodas;

(x , y) – Coordenadas relativa ao plano de orientação cartesiano, que coincide com a distância metade entre as rodas;
θ - ângulo que define a orientação no mapa.

Uma vez que V_l, V_r , ω e R são todas funções do tempo. Isolando ω e igualando as duas equações temos :

$$\omega = \frac{V_r}{(R + \frac{l}{2})} = \frac{V_l}{(R - \frac{l}{2})} \tag{2.2.3}$$

$$R = \frac{l}{2}\left(\frac{V_r + V_l}{V_r - V_l} \right) \tag{2.2.4}$$

Alguns casos especiais são de interesse:

- Vl = Vr neste caso o raio R é infinito e o robô se move em linha reta.
- Vl = -Vr neste caso o raio é zero e o robô gira em torno de um ponto que está à meia distância entre as duas rodas, girando portanto sobre si mesmo.

Para outros valores de V_l e V_r, o robô segue uma trajetória curvada em torno de um ponto à distância R do seu centro, mudando a sua posição e orientação.

O SDD é muito sensível à velocidade relativa das rodas. Pequenos erros na velocidade provocam trajetórias diferentes e não significam simplesmente um robô mais rápido ou mais lento.

É possível mostrar que para o SDD a solução geral do problema cinemático direto é:

$$x(t) = \frac{1}{2}\left\{ \int_0^t [V_r(t) + V_l(t)]\cos[\theta(t)]dt \right\} \tag{2.2.5}$$

$$y(t) = \frac{1}{2}\left\{ \int_0^t [V_r(t) + V_l(t)]sen[\theta(t)]dt \right\} \tag{2.2.6}$$

$$\theta(t) = \frac{1}{l}\left\{ \int_0^t [V_r(t) - V_l(t)]dt \right\} \tag{2.2.7}$$

O problema cinemático inverso é de solução extremamente difícil, a não ser nos casos especiais citados. Nestes casos, para o robô assumir uma posição (x, y, θ), basta que ele gire sobre si mesmo de modo a apontar para (x, y), andando em seguida

numa linha reta até atingir (x, y) e por último gire sobre si mesmo até que a orientação θ seja atingida.

Um modelo cinemático mais completo é proposto por ALBAGUL [1]. Neste modelo, assumem-se as mesmas condições ideais anteriores. As entradas do sistema são as velocidades angulares da roda esquerda e direita, ω_r e ω_l, respectivamente. O movimento do robô pode ser descrito pela cinemática de corpos rígidos modelado em sistema de três dimensões. Como no caso anterior, é importante definir a posição e orientação do robô assim como a localização e orientação do centro de gravidade, $p(X_o, Y_o, \theta_o)$, relativo ao quadro mundo fixo. A geometria e o sistema de coordenadas do robô são mostrados na Figura 2.2.3.

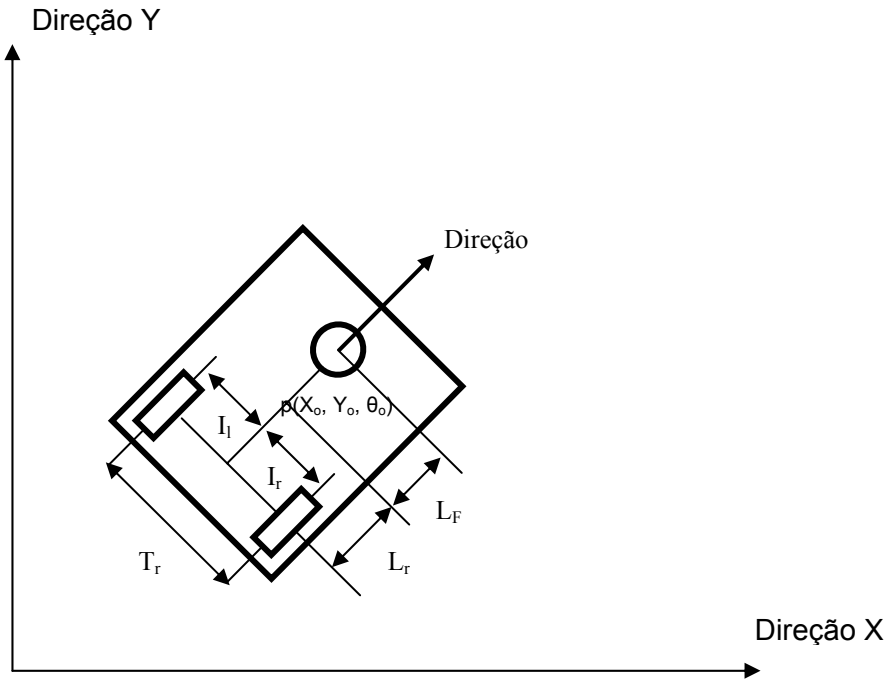

Figura 2.2.3 Geometria e coordenação do robô.

Assumindo que não existe escorregamento lateral ou longitudinal da roda, no sistema de três dimensões, as velocidades lineares das rodas direita e esquerda podem ser expressas por:

$$v_r = R_t \omega_r \hat{i} \tag{2.2.8}$$

$$v_l = R_t \omega_l \hat{i} \tag{2.2.9}$$

As equações de movimento da carta rígida com respeito a cada roda são:

$$V_r = V_g + r\hat{k}\left(-L_r\hat{i} - \frac{T_r}{2}\hat{j}\right) = u\hat{i} + v\hat{j} + r\frac{T_r}{2}\hat{i} - rL_r\hat{j} \tag{2.2.10}$$

$$V_l = V_g + r\hat{k}\left(-L_r\hat{i} + \frac{T_r}{2}\hat{j}\right) = u\hat{i} + v\hat{j} - r\frac{T_r}{2}\hat{i} - rL_r\hat{j} \qquad (2.2.11)$$

Onde:

V_g – vetor velocidade do centro de gravidade, (X_o, Y_o, θ_o);

L_r – distância do eixo traseiro ao centro de gravidade;

T_r – distância entre as duas rodas motoras;

L_F - distância da roda dianteira ao centro de gravidade;

u, v, r – velocidade frontal, velocidade lateral e velocidade em torno do centro de gravidade, respectivamente.

Usando as equações (2.2.8) a (2.2.11), o modelo cinemático para o robô pode ser obtido como:

$$u = (\omega_r + \omega_l)\frac{R_t}{2} \qquad (2.2.12)$$

$$r = (\omega_l - \omega_r)\frac{R_t}{T_r} \qquad (2.2.13)$$

$$v = (\omega_l - \omega_r)\frac{L_r R_t}{T_r} \qquad (2.2.14)$$

Então a direção, velocidades e posição do robô no sistema de coordenadas mundo podem ser obtidas.

$$\theta_o = \int r dt \qquad (2.2.15)$$

$$V_x = u\cos\theta_o - v sen\theta_o \qquad (2.2.16)$$

$$V_y = u\cos\theta_o + v sen\theta_o \qquad (2.2.17)$$

$$X_o = \int V_x dt \qquad (2.2.18)$$

$$Y_o = \int V_y dt \qquad (2.2.19)$$

Onde:

V_x e V_y – componentes velocidade do veículo relativo ao quadro mundo;

X_o e Y_o - posição do robô em relação ao quadro mundo;

θ_o - direção do robô em relação ao quadro mundo.

2.3. MODELO DINÂMICO

A dinâmica se preocupa com as relações entre as forças que atuam no corpo e o movimento originado. Portanto, o modelo dinâmico do robô tem por objetivo conhecer a relação entre o movimento do robô e as forças aplicadas no mesmo.

Esta relação é obtida mediante o chamado modelo dinâmico, tendo como base matemática:

1. A localização do robô definida pelas suas variáveis articulares e pelas coordenadas de localização de seu extremo, além de suas derivadas, velocidade e aceleração;
2. As forças aplicadas às articulações;
3. As dimensões do robô, como comprimento de suas hastes, massa e inércia de seus elementos.

A obtenção deste modelo para mecanismos de um ou dois graus de liberdade (GDL) não é excessivamente complexa, mas, à medida que o número de GDL aumenta, a obtenção deste modelo dinâmico se complica enormemente. Por este motivo, nem sempre é possível obter-se um modelo dinâmico expresso de forma fechada, isto é, mediante uma série de equações, normalmente do tipo diferencial de segunda ordem, cuja integração permita conhecer que movimento surge ao aplicar forças e que forças se devem aplicar para se obter o movimento desejado. O modelo dinâmico deve ser resultado então de iterações mediante utilização de algoritmos numéricos.

O problema da obtenção do modelo dinâmico de um robô é, portanto, um dos aspectos mais complexos da robótica, e que tem sido evitado em muitas ocasiões. Apesar disto, o modelo dinâmico é imprescindível se quisermos:

1. Simular o movimento do robô;
2. Projetar e avaliar a estrutura do robô;
3. Dimensionar os atuadores;
4. Projetar e avaliar o controle dinâmico do robô.

Este quarto item possui grande importância, posto que a qualidade do controle dinâmico do robô depende da precisão e velocidade de seus movimentos. A grande complexidade existente na obtenção do modelo dinâmico do robô tem motivado certas simplificações de modo que se possa utilizá-lo no projeto do controlador.

É importante notar que o modelo dinâmico completo do robô deve incluir não só a dinâmica de seus elementos, mas também o sistema de transmissão, dos atuadores

e equipamentos eletrônicos de comando. Estes elementos incorporam ao modelo dinâmico novas inércias, fricções e saturações dos circuitos eletrônicos, aumentando ainda mais sua complexidade.

Para se obter o modelo dinâmico do robô algumas simplificações são utilizadas, tais como, dentre outras, a consideração de que as hastes são rígidas e sem deformações. O modelo dinâmico da estrutura mecânica de um robô rígido pode ser elaborado utilizando a formulação Lagrange-Euler ou a Newton-Euler. A obtenção deste modelo se baseia fundamentalmente na obtenção do equilíbrio de forças através da utilização da segunda Lei de Newton, e do seu equivalente para movimentos de rotação denominada Lei de Euler.

$$\sum F = m\dot{v} \tag{2.3.1}$$

$$\sum T = I\dot{\omega} + \omega(I\omega) \tag{2.3.2}$$

Supondo-se que toda massa esteja concentrada no centro de gravidade do elemento e que não exista fricção e carga alguma, conclui-se que:

- O modelo dinâmico direto expressa a evolução temporal das coordenadas articulares do robô em função das forças sobre ele aplicadas;

- O modelo dinâmico inverso expressa as forças em função da evolução das coordenadas articulares e suas derivadas.

No caso de um robô com cinco ou seis GDL, pode ser muito complicado obter estas equações. Como alternativa, utiliza-se a formulação Lagrangiana, que se baseia em considerações energéticas.

Para se obter o modelo dinâmico do robô, forças devem ser aplicadas e analisadas. O momento deve ser aplicado em algum ponto do robô, neste caso o centro de gravidade. Sabendo que o robô possui três GDL, o qual permite somente o momento nas direções laterais e longitudinais junto com o deslocamento angular, as equações da força e do momento podem ser expressas por:

$$\sum F_x = m(\dot{u} - vr) \tag{2.3.3}$$

$$\sum F_y = m(\dot{v} - ur) \tag{2.3.4}$$

$$\sum M_z = I_z\dot{r} \tag{2.3.5}$$

As forças que agem sobre o robô são aquelas exercidas pelas rodas motora direita e esquerda. Estas forças são proporcionais ao torque aplicado menos a quantidade de torque exigido para acelerar as rodas, caixa de redução e rotor do motor. O torque aplicado é dividido de duas maneiras: o torque linear para acelerar o motor e o torque angular para acelerar as rodas, caixa de redução e rotor:

$$T_{app} = T_{lin} + T_{ang} \tag{2.3.6}$$

Onde T_{app}, T_{lin} e T_{ang} são os torques aplicado, linear e angular respectivamente.

O torque linear é convertido em força longitudinal na interface superfície/roda e expresso por:

$$Fx = \frac{T_{lin}}{Rt} \tag{2.3.7}$$

e o torque angular é:

$$T_{ang} = I_z \dot{\omega} = I_z \frac{\dot{u}}{R_t} \tag{2.3.8}$$

$$Fx = \frac{T_{lin} - T_{ang}}{Rt} = \frac{R_t T_{app} - I_z \dot{u}}{R_t^2} \tag{2.3.9}$$

Usando a equação (2.3.9), uma distinção pode ser feita para obter as forças exercidas pelas rodas motoras esquerda e direita, como abaixo:

$$Fx_r = \frac{R_t T_{appr} - I_z \dot{u}_r}{R_t^2} \tag{2.3.10}$$

$$Fx_l = \frac{R_t T_{appl} - I_z \dot{u}_l}{R_t^2} \tag{2.3.11}$$

As equações dinâmicas que descrevem o movimento do robô em termos de aceleração são:

$$\dot{V}_x = \frac{F_{Xr} - T_{Xl}}{m} + V_y \omega \tag{2.3.12}$$

$$\dot{V}_y = \frac{F_{Yr} + T_{Yl}}{m} - V_x \omega \tag{2.3.13}$$

$$\dot{r} = \frac{L_r F_{Xr} - L_l F_{Xl} - L_r (F_{Yr} + T_{Yl})}{I_z} \tag{2.3.14}$$

As velocidades longitudinal, lateral e em torno do eixo vertical que passa pelo centro de gravidade, são simplesmente a integral destas acelerações. Ainda, a posição e a direção do robô podem ser obtidas da mesma maneira que em 2.2.

2.4. ESTRATÉGIA DE CONTROLE

A tarefa do controlador é alcançar determinadas especificações de desempenho e características desejáveis para a movimentação do robô. O controlador deve ser projetado para executar corretamente a seqüência planejada de movimentos na presença de qualquer erro. As estratégias de controle são baseadas nas forças de tração que provocam o movimento, assim como a manutenção do percurso no caminho desejado, sem a ocorrência de desvio ou derrapagem. O controle do movimento da tração tem algumas características desejáveis como:

1. Manter a aceleração e a desaceleração a mais rápida e precisa possível;
2. Manter a precisão no percurso do caminho desejado;
3. Manter a estabilidade do robô durante as manobras;
4. Prevenir o robô da ocorrência de derrapagem ou escorregamento.

O diagrama em bloco do sistema de controle do robô móvel é mostrado na Figura 2.4.1. A proposta de ALBAGUL [1] é uma estratégia de controle constituída de duas partes. A primeira refere-se à estratégia de controle de “Nível Baixo” a qual lida com as mudanças na dinâmica do robô. A segunda parte é a estratégia de controle de “Nível Alto” a qual lida com as mudanças no ambiente no qual o robô opera. A combinação do controle de movimento de Nível Baixo e Alto para o robô é mostrada na Figura 2.4.2.

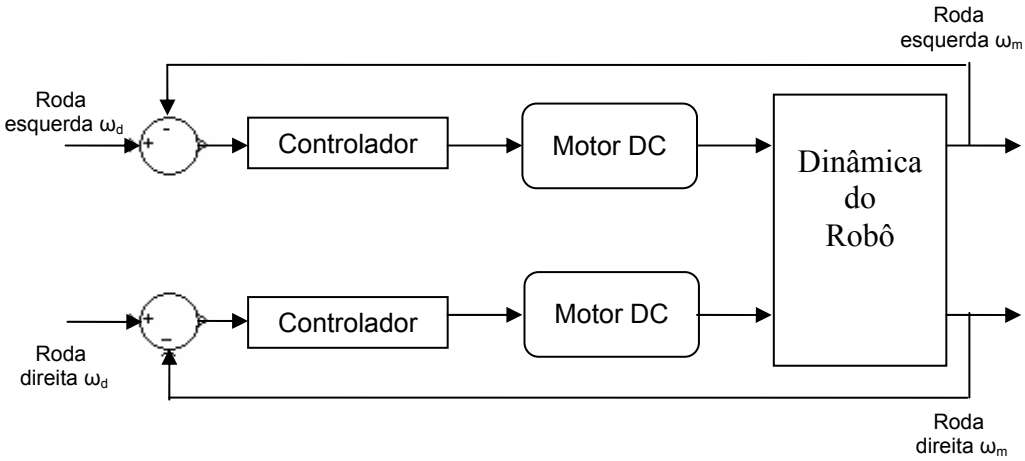

Figura 2.4.1 Diagrama em bloco do sistema de controle do robô móvel.

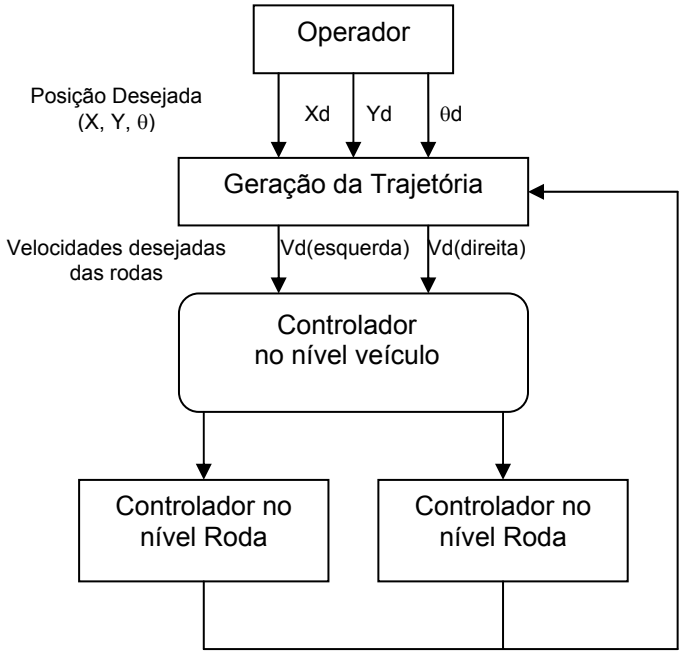

Figura 2.4.2 Controle combinado de movimento do robô móvel. Nível Baixo e Alto.

2.4.1. Estratégia de controle de Nível Baixo

Nesta estratégia, um algoritmo de controle de movimento adaptativo baseado no controlador adaptativo auto-sintonizado, pela localização do pólo, é considerado. Este controlador, constituído de uma estrutura PID, é projetado para estimar as mudanças nos parâmetros dinâmicos do sistema.

A estrutura do controle adaptativo auto-sintonizado oferece um bom quadro de trabalho para estimar os parâmetros do modelo. Isto pode ser alcançado através do mecanismo de estimativa. Então o controlador é implementado e está apto para interagir com as mudanças dinâmicas e tomar as devidas ações de controle do movimento. Entretanto, ele não pode estimar ou interagir com as mudanças nas condições da superfície. O diagrama em bloco básico para o controlador adaptativo auto-sintonizado é mostrado na Figura 2.4.3.

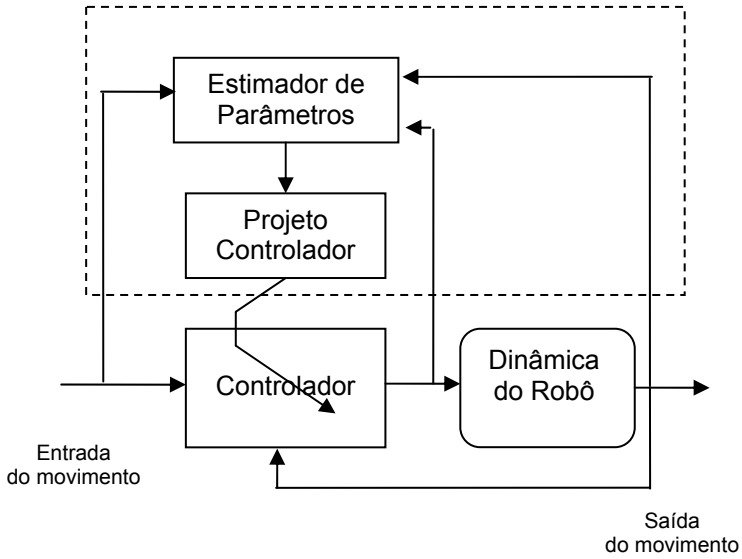

Figura 2.4.3 Diagrama em bloco básico do controlador adaptativo auto-sintonizado.

2.4.2. Estratégia de controle combinada

Esta estratégia adota uma combinação dos controles de Nível Baixo e Alto levando em consideração a estrutura do ambiente e a dinâmica do robô. Esta é a principal diferença entre o controle combinado e o controle de Nível Baixo na qual não inclui a estrutura do ambiente e suas condições como mostrado na Figura 2.4.4.

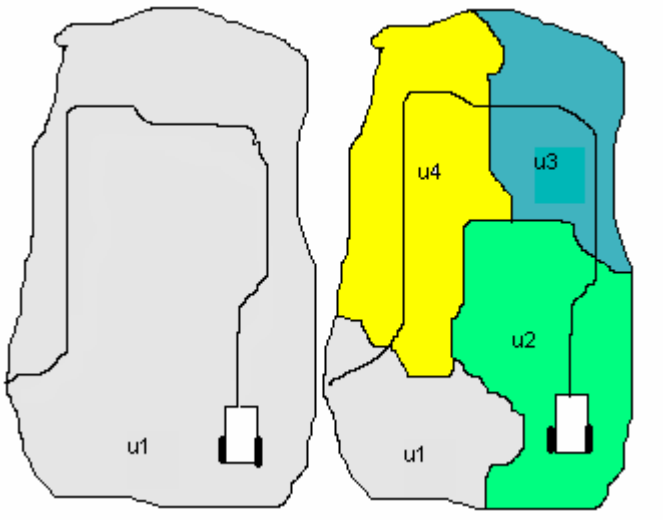

Figura 2.4.4 Movimento do robô sob diferentes condições de superfícies. Os coeficientes de atrito μ1 a μ4 são distintos.

2.4.3. Avaliação e resultados

Os resultados obtidos por ALBAGUL [1] mostraram que o controlador adaptativo combinado de "Nível Alto e Baixo" é adequado para compensar e interagir com as mudanças das condições do caminho e manter o percurso com melhor precisão. O veículo utilizado no experimento possuía os seguintes parâmetros:

Massa (m): 40 Kg

Inércia (J): 2,627 Kg/cm^2

Distância da roda dianteira ao centro de gravidade (L_F): 0,3 m

Distância do eixo traseiro ao centro de gravidade (L_R): 0,1 m

O veículo utilizado no experimento possuía as seguintes restrições:

Máxima velocidade linear (V_{max}): 2 m/s

Máxima aceleração linear (A_{max}): 1,5 m/s^2

Máxima velocidade angular (ω_{max}): 2 rad/s

Máxima aceleração angular ($A\omega_{Max}$): 1,5 rad/s^2

No percurso avaliado de aproximadamente 40 metros, constituído por 15 trechos alternados de retas e curvas, em uma área de avaliação de 180m^2 (9m x 20m), com diferentes regiões de coeficiente de atrito, foram obtidos os seguintes resultados:

- Com controlador único “Nível Baixo”

Maior erro de deslocamento: 4 metros (trecho constituído de reta de 8 metros)

Menor erro de deslocamento: 0,1 metro (trecho constituído de reta de 2,5 metros)

- Com controlador combinado “Nível Alto e Baixo”

Maior erro de deslocamento: 0,006 metros (trecho constituído de curva de 2 metros)

Menor erro de deslocamento: nulo (trecho constituído de reta de 1 metro)

Erro de deslocamento no trecho constituído de reta de 8 metros: 0,001m

Erro de deslocamento no trecho constituído de reta de 2,5 metros: 0,0015m

Como podemos observar com o controlador único “Nível Baixo”, o erro ou desvio do robô em relação ao caminho desejado é grande devido ao escorregamento. Este controlador mostrou-se incapaz de interagir com as mudanças nas condições da superfície. Entretanto, no caso do controlador combinado “Nível Alto e Baixo”, é observado que o robô segue o caminho desejado e alcança uma boa precisão durante o percurso. O erro de deslocamento é muito pequeno e o robô alcança a posição final com precisão.

A estratégia de controle com compensador antecipatório proposta por GYULA [7] adota um controlador para cada motor, incluindo o tradicional controlador realimentado do tipo PI e o compensador antecipatório com a dinâmica inversa do veículo. As entradas do compensador antecipatório são as velocidades angulares desejadas das rodas direita e esquerda (ω_r e ω_l), cuja as equações do controlador proposto são:

$$\tau_r = k_p(\omega_{rd} - \omega_r) + K_i \int_0^t (\omega_{rd} - \omega_r)dt + A\dot{\omega}_{rd} + B\dot{\omega}_{ld} + C\omega_{rd} \qquad (2.4.1)$$

$$\tau_l = k_p(\omega_{ld} - \omega_{lr}) + K_i \int_0^t (\omega_{ld} - \omega_l)dt + A\dot{\omega}_{ld} + B\dot{\omega}_{rd} + C\omega_{ld} \qquad (2.4.2)$$

Onde:

A,B e C são parâmetros relacionados ao movimento do veículo.

Kp – Ganho Proporcional.

Ki – Ganho Integral.

ω_{rd} – Valor desejado da velocidade angular da roda direita.

ω_{ld} - Valor desejado da velocidade angular da roda esquerda.

A estrutura do controlador proposto é apresentada na Figura 2.4.5.

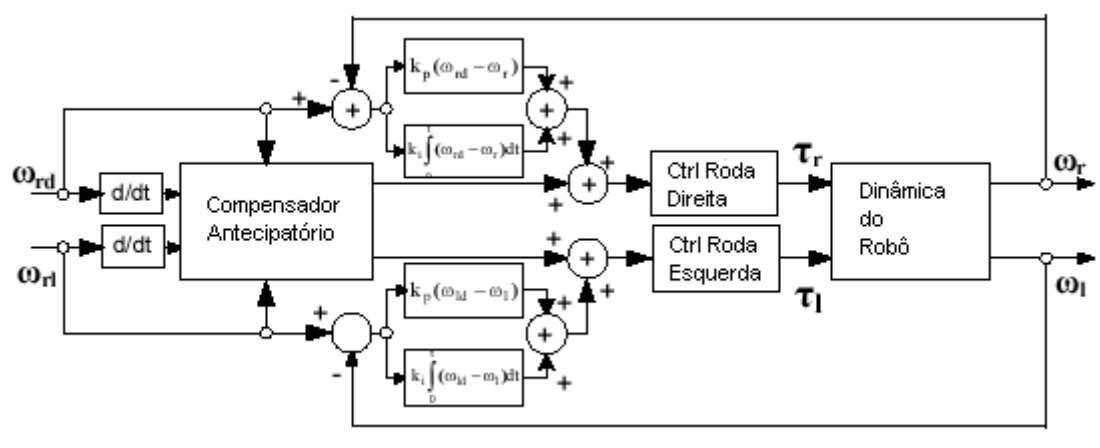

Figura 2.4.5 Diagrama em bloco do controlador com compensador antecipatório.

3. CARACTERÍSTICAS DO ROBÔ (HARDWARE)

3.1. INTRODUÇÃO

Considerando que o robô deverá ser capaz de navegar com segurança em um ambiente com obstáculos devidamente conhecido, torna-se necessário para o seu desenvolvimento e avaliação de desempenho o estabelecimento das seguintes premissas básicas, necessárias à definição dos critérios de projeto.

- Área de avaliação de até $25m^2$ (5m x 5m);
- Tamanho mínimo do obstáculo de $0,01m^2$ (0,01m x 0,01m);
- Piso da área de avaliação com coeficiente de atrito fixo;
- Geometria e tamanho do robô compatível com a área de avaliação e obstáculo.
- Comunicação bidirecional: o robô receberá de um computador portátil externo a seqüência de movimentos previamente estabelecida para a execução da trajetória, e fornecerá o seu estado operacional através de comunicação por rádio freqüência;
- Sistema embarcado de sensoriamento, acionamento e controle em tempo real. Este deverá incluir um sistema operacional que permita controlar no mínimo 4 tarefas: controle em malha fechada da roda direita, controle em malha fechada da roda esquerda, recepção da seqüência de movimentos e envio do estado operacional do robô, ambos via rádio;
- Consumo de energia baixo para permitir longos percursos sem a necessidade de recarga do módulo de alimentação (bateria);
- Peso mínimo de carga transportada de 3,5 kg, não considerando peso próprio do robô;
- Distribuição simétrica de carga dos módulos embarcados, a fim de evitar possível escorregamento nas rodas motoras;
- Velocidade linear e angular adequada para resultar em boa precisão no percurso da trajetória definida.

O desenho geométrico idealizado para o robô, apresentado na Figura 3.1.1, foi desenvolvido para atender as premissas básicas ou especificações previamente definidas, mostrando a localização dos módulos embarcados, as duas rodas motoras com os seus respectivos acionamentos e os dispositivos de medição de velocidade.

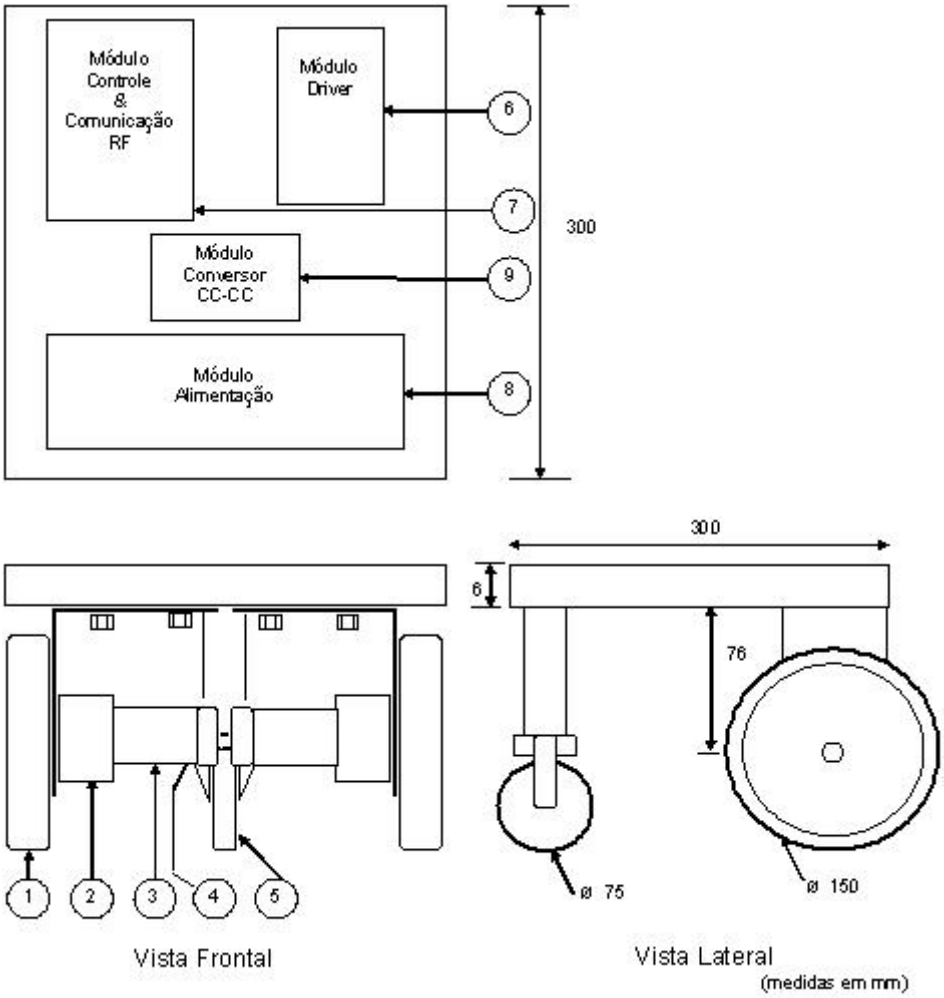

Figura 3.1.1 Desenho geométrico do robô, com a localização dos módulos.

embarcados.

A Tabela 3.1.1 descreve os principais componentes utilizados para a construção do robô.

Tabela 3.1.1 Lista dos principais componentes do robô.

Item	Descrição	Qtde	Obs
1	Roda motora	2	
2	Caixa de redução	2	
3	Motor DC	2	
4	Encoder HEDS 5500 J06 500 PPR	2	
5	Roda livre	1	
6	Módulo Driver MD22	1	
7	Módulo Controle & Comunicação RF SRB 13213 Freesaclae	2	Um dos módulos instalado no PC
8	Módulo Alimentação - Baterias de 12 Vdc e 7,4 Vdc	1	
9	Módulo Conversor CC-CC	1	

No próximo item são abordadas a criação da base mecânica e a especificação e agregação dos motores e rodas. Esta etapa é crucial na construção de robôs, já que o alinhamento dos motores na base é fundamental para o correto controle.

3.2. PROJETO MECÂNICO

A estrutura de locomoção do robô deverá ser composta de dois motores com caixa de redução e encoder acoplado ao extremo do eixo. Conforme as premissas básicas citadas, estes motores devem possuir torque suficiente para movimentar toda a estrutura do robô, com todos os módulos instalados, ou seja, módulo de controle e comunicação, módulo driver e módulo de alimentação. Devem também possuir velocidade que permita executar com precisão o percurso da trajetória definida. A escolha destes dar-se-á pelos fatores disponibilidade, facilidade e preço, uma vez que o robô apresentado visa executar com precisão o percurso da trajetória definida, não importando o tempo necessário para fazê-la ou capacidade de carga. O motor pode ser alimentado com até 12 Vdc (Volts corrente contínua), tendo nesta faixa, uma velocidade máxima sem carga, na saída do redutor de 39,6 rotações por minuto. A escolha do motor e da caixa de redução é feita com base nas premissas do item 3.1 e nos procedimentos abaixo :

a) Diâmetro da roda.

Considerando que o robô deverá transpor obstáculos de até 50 mm, tem-se:

D = 2 X raio = 2 x 50 = 100 mm

Diâmetro adotado: 150 mm.

b) Máxima velocidade do robô.

A máxima velocidade a ser alcançada pelo robô é de 0,3 m/s. Este valor arbitrado é inferior a 1 m/s para evitar instabilidade do movimento [9].

c) Relação da caixa de redução.

Considerando que a velocidade nominal sem carga, ω_n , de motores dc de potência inferior a 50 W, é de 5000 rpm, tem-se:

$$\omega_{ne} = \frac{\text{Vel.máx.}}{2 \times \pi \times r} \times 60 = \frac{0,3}{0,471} \times 60 = 38 \text{ rpm} \qquad (3.2.1)$$

$$\text{Razão de Redução} = \frac{\omega_n}{\omega_{ne}} = \frac{5000}{38} = 130,83 \qquad (3.2.2)$$

Onde:

ω_n – Velocidade do eixo do motor sem carga (rpm);

ω_{ne} – Velocidade do eixo da redução sem carga (rpm);

d) Potência do motor e torque.

Para a definição da potência do motor e torque são considerados que a velocidade do motor é constante e a aceleração é muito pequena e por isto pode ser ignorada. O regime de atuação é permanente e por não existir escorregamento entre a roda e o chão, não ocorrerá dissipação de energia na roda. Além do parâmetro velocidade sem carga do motor, ω_n, os parâmetros e constantes abaixo também são utilizados para esta definição.

$B\tau_s$ – Torque de stall (Nm ou K_gm) – Torque mínimo para parar completamente o eixo do motor;

$B\tau_L$ – Torque exercido pela carga (Nm ou K_gm);

$B\tau_m$ – Torque exercido pelo motor (Nm ou K_gm);

J_{eq} – Momento de inércia equivalente das engrenagens;

θ_m – Deslocamento angular do eixo do motor;

b_{eq} – Atrito viscoso equivalente das engrenagens;

n - Relação equivalente ao número de dentes das engrenagens;

μ - Coeficiente de atrito estático (entre borracha e concreto) – 0,4 a 0,6;

N – Força normal (N ou K_g) – Reação normal ao peso próprio do sistema (3,5 Kg +3 Kg)/2 = 3,25 Kg.

A equação que relaciona o torque exercido pelo motor com o torque exercido pela carga é definida por:

$$J_{eq}\ddot{\theta}_m + b_{eq}\dot{\theta}_m + nT_L = T_m \qquad (3.2.3)$$

$$T_L = \text{sen}\phi \times r \times \mu \times N = 0,075 \times 0,5 \times 3,25 = 0,1218\ K_g m \qquad (3.2.4)$$

$$T_m = T_L \times \frac{\omega_{ne}}{\omega_n} \times 0,66 = 0,1218 \times \frac{38}{5000} \times 0,66 = 0,00061 K_g m \qquad (3.2.5)$$

A potência mínima do motor, para vencer o torque exercido pela carga é dada por:

$$P = \tau_m \omega_n - \tau_m^2 \frac{\omega_n}{\tau_s} = 0,00061 \times 5000 - (0,00061)^2 \times \frac{5000}{0,0061} = 2,74W \qquad (3.2.6)$$

Em anexo estão disponíveis as folhas de dados do motor dc escolhido, com caixa de redução escolhido. A seguir estão resumidas algumas das especificações obtidas da folha de dados.

3.2.1. Motor com caixa de redução

Especificações

Fabricante: Globe Motors;
Modelo: IM – 15 Gearmotor;
Fonte DC: 12 Vdc;
Corrente sem carga: 220 mA;
Corrente de Stall: 6 A;
Velocidade máxima sem carga: 5060,08 rpm;
Velocidade máxima sem carga, no eixo após caixa de redução: 39,6 rpm;
Relação da Redução: 127.78:1;
Torque de Saída: 300 oz – in = 0,216 Kg x m;
Torque de atrito: 1,2589 Kg x m;
Peso: 453.6 g;
Dimensões máximas: 98 mm (comprimento) e 50,8 mm (diâmetro).

Cálculo da velocidade máxima do robô sem carga

Dados:
Diâmetro da roda **:** 150 mm
Velocidade angular máxima: 39,6 rpm = 0.66 rps (1,515s para um giro completo).

Conforme MCU (Movimento Circular Uniforme) temos:

$$L = \alpha r \tag{3.2.7}$$

$$v = \frac{dd}{dt} = \frac{Lr}{t} = \frac{2\pi r}{t} = \frac{2\pi 75}{1{,}515} = 0{,}311 \text{(m/s)} \tag{3.2.8}$$

As variáveis “L” e “r” são o comprimento e o raio do arco da circunferência.

Donde se obtém as seguintes restrições :

Máxima velocidade linear (V_{max}): 0,311 m/s;

Máxima aceleração linear (A_{max}): 0,272 m/s^2;
Máxima velocidade angular (ω_{max}): 0,311 rad/s;
Máxima aceleração angular ($A\omega_{max}$): 5,51 rad/s^2.

Da geometria do robô e dos componentes utilizados, obtém-se os seguintes parâmetros:

Massa (m): 3,8 Kg;
Momento de Inércia (J): 1,14 Kgxm^2;
Distância entre rodas motora direita e esquerda (T_R): 0,30 m;
Distância da roda dianteira ao centro de gravidade (L_F): 0,21 m;
Distância do eixo traseiro ao centro de gravidade (L_R): 0,045 m.

3.3. MÓDULO DE CONTROLE DE MOVIMENTO E COMUNICAÇÃO POR RF

Este módulo é constituído pelo kit de desenvolvimento da empresa Freesacle Semiconductor, ZigBee Evaluation Kit 6.0 [9]. O hardware do kit utilizado contém:

- 2 x 13213 – Placa Referenciada com Sensor (SRB-Sensor Reference Board);
- 1 Cabo de comunicação USB.

As placas de sensor de referência são idênticas, sendo que uma é conectada ao PC através do cabo USB e a outra é instalada no robô. Estas placas incluem os seguintes componentes e interfaces principais:

- MC13213 – Este circuito integrado inclui um módulo Modem padrão 802.15.4 e um microcontrolador (MCU-Micro Controller Unit) baseado na família HCS08 de microcontroladores de 8 bit's com sinal de clock de até 40MHz. Possui 60 Kb de memória flash e 4 Kb de memória RAM;
- MMA7260Q - Sensor do tipo acelerômetro de 3 eixos X, Y e Z;
- 1 Porta USB 2.0;
- 1 Conexão Serial RS232;
- 4 Teclas do tipo push button (S1, S2, S3, S4);
- 1 Tecla de Reset ;
- 4 LEDs, (LED1, LED2, LED3, LED4);
- 1 Chave de Liga/Desliga (S100);

- 1 Antena construída no circuito impresso;
- 1 conexão BDM (Back Ground Debuging Module) que permite a programação da memória Flash e depuração no circuito através do cabo Multilink USB;
- 1 Conector de alimentação 5 a 9 Vdc;
- 1 Porta pilhas 2 x AA;
- 1 Conector de 26 pinos para acesso a pinos específicos do MCU e módulo de RF.

3.3.1. Descrição da placa 13213 SRB

Nas Figuras 3.3.1 e 3.3.2 são mostradas a placa 13213 SRB e seu diagrama em blocos respectivamente.

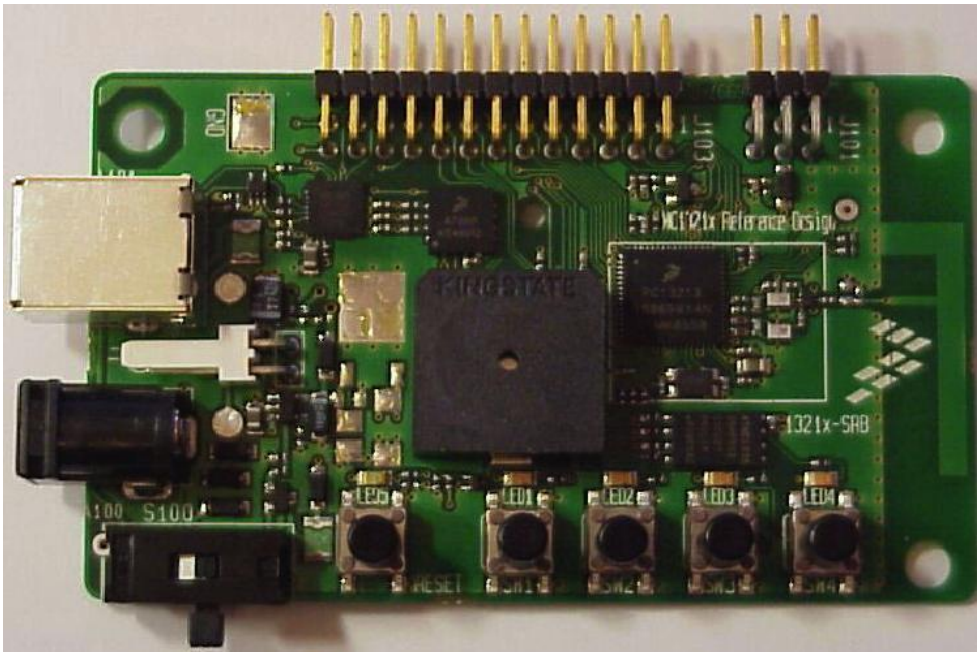

Figura 3.3.1 Placa da Freescale Semiconductor 13213 SRB.

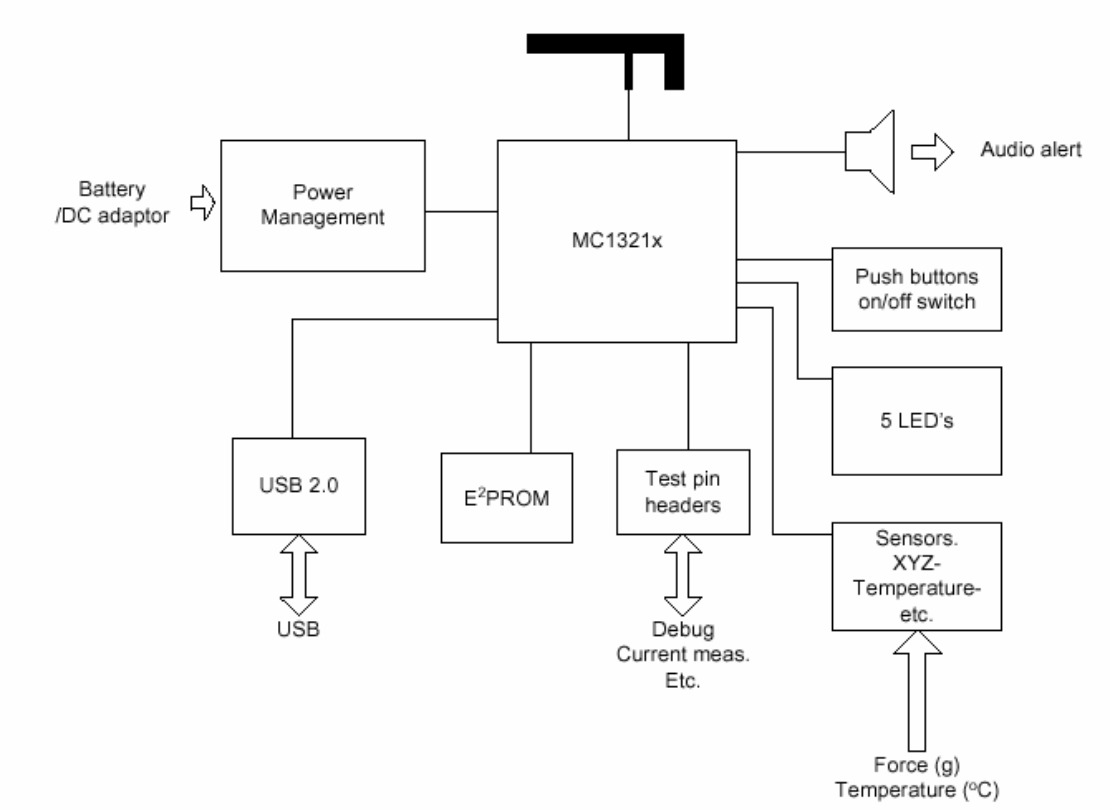

Figura 3.3.2 Diagrama em Bloco da Placa 13213 SRB.

Na Figura 3.3.3 é mostrado o diagrama em blocos do CI MC13213, constituído pelo módulo do Modem padrão 802.15.4 e pelo microcontrolador (MCU) HCS08. O CI MC 13213 é encapsulado em um invólucro de 71 pinos do tipo LGA/QFN (Quad Flat Non-lead), quadrado plano com dimensões de 9 x 9 mm e 1 mm de altura [10].

A placa 13213 SRB possui um cristal de referência de clock de 16 Mhz dedicada ao módulo do Modem padrão 802.15.4. Apesar do microcontrolador permitir um oscilador externo de até 40 MHz, o lay out da placa não possui esta opção. O clock de referência do microcontrolador é obtido da saída de clock, CLKO, pino 10 do módulo do Modem padrão 802.15.4. Portanto, a freqüência máxima programável para o clock do microcontrolador é a própria freqüência de clock do cristal dedicado ao módulo do Modem padrão 802.15.4, ou seja, 16 MHz.

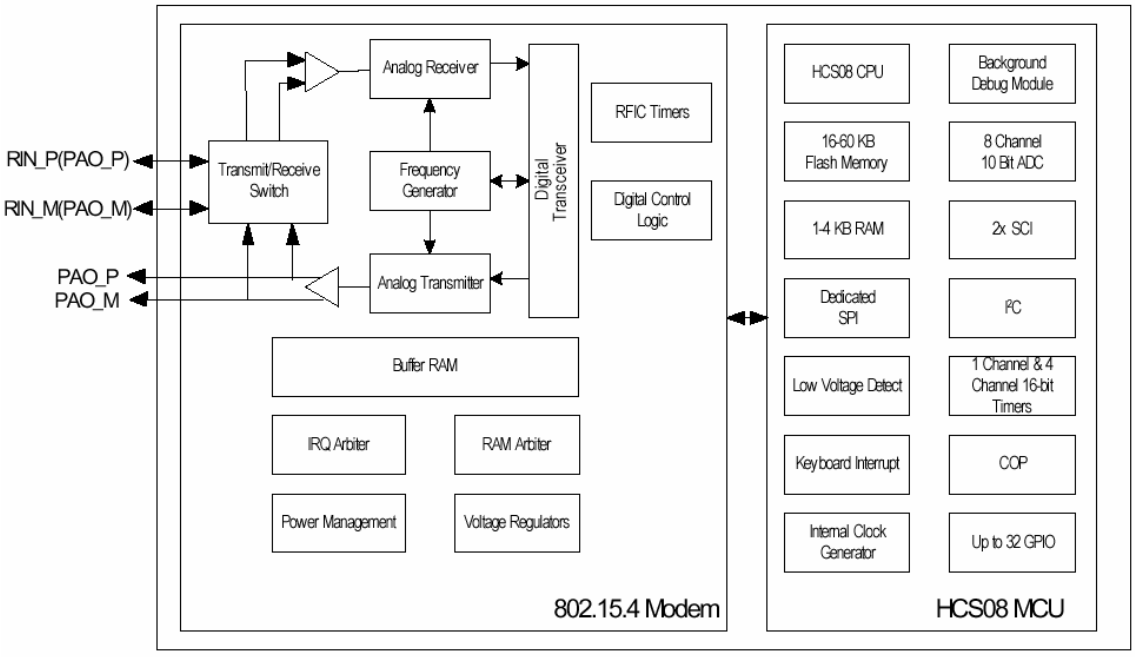

Figura 3.3.3 Diagrama em Blocos do CI - MC13213.

3.3.2. Descrição do microcontrolador MCU HCS08

O microcontrolador (MCU) HCS 08, interno ao CI MC 13213, inclui dois módulos de comunicação serial independentes (SCI-Serial Comunication Interface). Estas interfaces de comunicação serial são conectadas a uma interface RS 232 e uma interface USB. Contém ainda um módulo conversor analógico para digital (ATD-Analog to Digital Controller) de 8 canais, com resolução de 10 bits e freqüência de conversão de 2 MHz. Uma interface periférica serial (SPI-Serial Peripheral Interface) controla a comunicação entre o microcontrolador (MCU) HCS 08 e o transceiver (Modem 802.15.4). Possui dois temporizadores de 16 bit's sendo que o TPM1 possui somente um canal que é utilizado pela placa e o TPM2 possui 4 canais disponíveis para utilização.

Existem 38 pinos que são compartilhados entre as funções de entrada e saída de propósito geral (GPIO –General Propouse Input Output) e os periféricos internos ao chip. Estas funções estão agrupadas em 6 portas, no qual incluem um total de 38 pinos do tipo GPIO, sendo que 6 destes são utilizados internamente pelo CI, mas estão disponíveis para monitoração e verificação. As seguintes portas e pinos foram utilizados para o sistema de controle em malha fechada das rodas motora esquerda e direita, Tabela 3.3.1.

Tabela 3.3.1 Definição das portas e dos pinos de entrada e saída utilizados.

Porta	Descrição	Função	J103
PTD4	TPM2CH1 – Canal 1 do 2º Módulo Temporizador e PWM	PWM RD (saída)	Pino 2
PTD5	TPM2CH2 – Canal 2 do 2º Módulo Temporizador e PWM	PWM RE (saída)	Pino 1
PTD6	TPM2CH3 – Canal 3 do 2º Módulo Temporizador e PWM	Captura vel. digital RD	Pino 4
PTD7	TPM2CH4 – Canal 4 do 2º Módulo Temporizador e PWM	Captura vel. digital RE	Pino 3

O diagrama em bloco do MCU HCS08 interno ao CI MC13213 está mostrado na Figura 3.3.4.

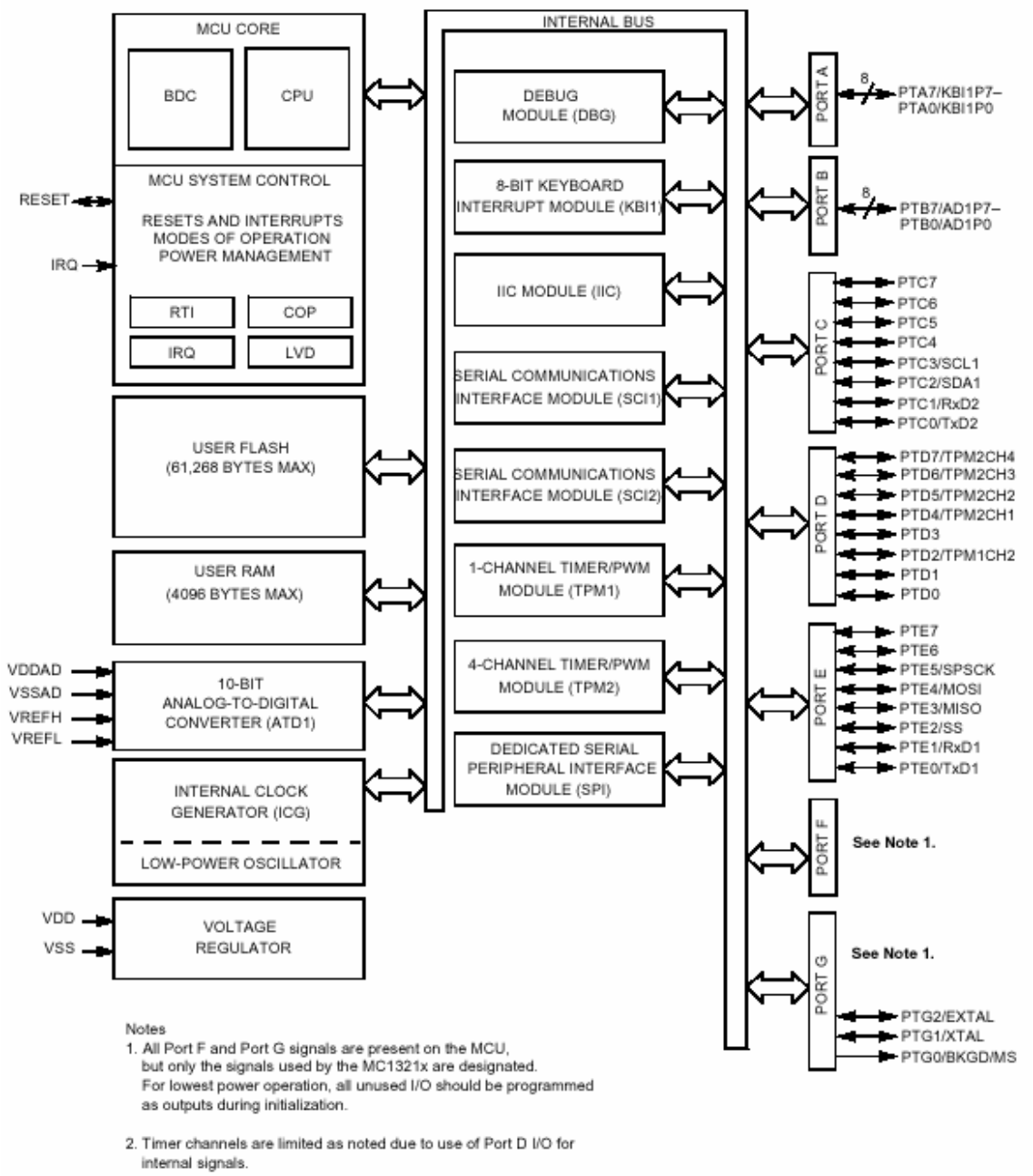

Figura 3.3.4 Diagrama em Bloco do microcontrolador (MCU) HCS08.

3.3.3. Descrição do módulo Modem padrão 802.15.4

O Modem padrão 802.15.4 é um transmissor e receptor de Radio Freqüência operando na freqüência de 2.4 GHz, compatível com o protocolo de comunicação definido pelo padrão IEEE 802.15.4. Possui ainda as seguintes características:

- Suporta modulação digital por desvio de fase em quadratura ortogonal (OQPSK-OffSet Quadrature Phase Shift Keying) com taxa de dados de 250 Kbps em canais de 5 MHz e codificação e decodificação em espalhamento de espectro;

- Opera em um dos 16 canais selecionáveis na banda de 2.4GHz ISM (Médica, Científica e Industrial);
- Potência de saída nominal de -1 dBm a 0 dBm, com programação típica de -27 dBm a + 3dBm;
- Sensibilidade de recepção de até -92 dBm em uma taxa de erro de pacote (PER) de 1%;
- Chave de transmissão/recepção integrada;
- Dois amplificadores de potência de saída em paralelo, podendo ser programado para operação diferencial de porta única ou porta dupla que suportam um amplificador de baixo ruído (LNA-Low Noise Amplifier) e/ou amplificador de potência externo;
- Três modos de baixa potência para aumentar o tempo de vida da bateria;
- Saída de freqüência de clock programável para uso do MCU;
- Quatro comparadores de tempo interno disponível para suplementar os recursos de temporizador para o MCU;
- Suporta o modo de transferência de dados em pacote (os dados são armazenados na memória RAM do chip) e em streaming (os dados são processados palavra por palavra);
- Oscilador controlado por tensão (VCO-Voltage Controller Oscilator).

3.4. MÓDULO DRIVER

Como visto no item 3.1, o robô possui dois motores DC para acionamento das rodas. Para obter-se um controle eficiente, torna-se necessário interfacear esse motores a um microcontrolador através do desenvolvimento de vários circuitos capazes de realizar esta tarefa.

A ponte H, mostrada na Figura 3.4.1, é uma topologia de circuito muito utilizada para interfacear um motor DC e um microcontrolador,. É constituída basicamente de quatro chaves controladas pelo microcontrolador que determinam a polaridade da tensão no motor, permitindo a mudança no sentido da corrente deste, tendo como efeito a inversão do sentido de rotação das rodas. Quando o microcontrolador comanda as chaves S1 e S4 para a posição fechada, mantendo as demais na posição aberta, a corrente circula pelo motor no sentido positivo. Quando as chaves S2 e S3 são fechadas e as demais abertas, a corrente muda de sentido e o motor gira no sentido oposto. Deixando as quatro chaves simultaneamente abertas, o motor perde a capacidade de acionamento.

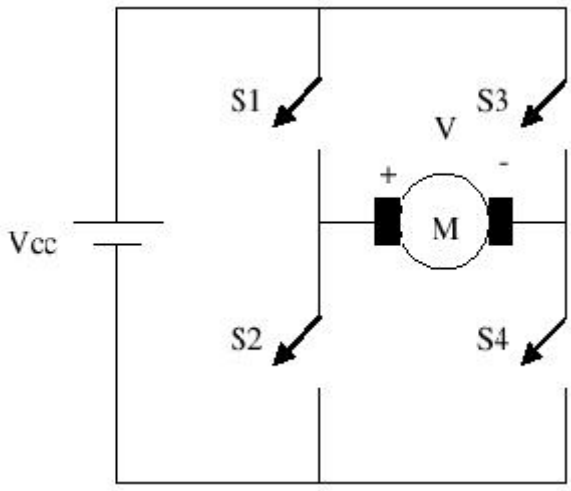

Figura 3.4.1 Circuito ilustrativo da ponte H.

Utilizando a técnica de modulação da largura de pulso (PWM - *Pulse-Width Modulation*) para controlar a abertura e fechamento das chaves que compõem a ponte H, é possível controlar a velocidade do motor, uma vez que ele responde ao valor RMS do sinal PWM. Este sinal consiste simplesmente em uma onda quadrada, cujo ciclo de trabalho pode ser variado, resultando em diferentes valores médios (e RMS) de tensão.

Uma vez que o microcontrolador não tem uma unidade eletrônica de potência incorporada, faz-se necessária a utilização da ponte H para o acionamento do motor. Desta forma, uma ponte H com transistores de potência pode perfeitamente desempenhar o papel das chaves eletrônicas, fornecendo a corrente necessária para o acionamento do motor, ficando o microcontrolador responsável apenas pelos sinais de controle. As referências SGS Thompson L293D e Motorola MPC1710A, são dois exemplos típicos de CIs baseados em pontes H.

3.4.1. Características do Módulo Driver

O módulo de driver utilizado possui as seguintes características:

- Permite excitar dois motores com controle independente;
- Facilidade de uso e flexibilidade;
- A tensão de 15 Vdc necessária para polarização do MOSFET é gerada internamente com um conversor DC-DC. O módulo requer somente duas fontes de energia, sendo que uma fonte padrão de 5 Vdc, com consumo máximo de 50 mA, para a lógica de controle e a outra fonte, em 12 Vdc com consumo máximo de 2 A, para alimentação do motor conforme especificação;

- Os motores podem ser comandados pelo registrador I2C (Interface de Comunicação entre Circuito Integrado) ou através de entrada analógica, 0 a 5 Vdc ou Servo RC (Radio Controle).

As seguintes opções estão disponíveis, para o controle do motor:

a) Através do barramento I2C, até 8 módulos MD22 podem ser controlados. O endereço do módulo é selecionado por micro chave e permite 4 modos de operação.

b) Duas entradas analógicas independentes de 0 a 5 Vdc, sendo que em 0 VDC a velocidade é máxima em um sentido de rotação, em 2,5 Vdc o motor para e em 5 Vdc a velocidade é máxima no sentido oposto de rotação.

- Capaz de excitar MOSFETs com corrente de até 5 A. Estes são capazes de conduzir correntes contínuas de até 27 A e pulsos breves de corrente de até 290 A.

O modo analógico 0 - 2,5 – 5 Volts é o escolhido para o controle do motor. Neste modo, os motores são controlados independentemente por dois sinais de 0 a 5 Vdc, através dos terminais SCL (motor da RD) e SDA (motor da RE). Em 0 Vdc a potência é máxima e reversa, em 2,5 Vdc o motor para e em 5 Vdc a potência é máxima no sentido direto. Existe uma pequena banda morta de 2,7% em torno de 2,5 Vdc para que o motor possa estabilizar-se na posição de parado.

Uma segunda opção para o controle do motor dc é o modo servo RC (Radio Control). Neste modo, os pulsos de controle proveniente do MCU são também conectados aos terminais SCL (motor da RD) e SDA (motor da RE). O módulo de driver M22 fornece um controle total no range de 1,0 a 2,0 ms. Em 1,5 ms, o motor está desligado (parado). Existe uma zona morta de 7 μs centrada em 1,5 ms para garantir a parada do motor.

A Figura 3.4.2 mostra a placa do módulo de Driver M22 utilizado no interfaceamento dos motores com o microcontrolador do robô.

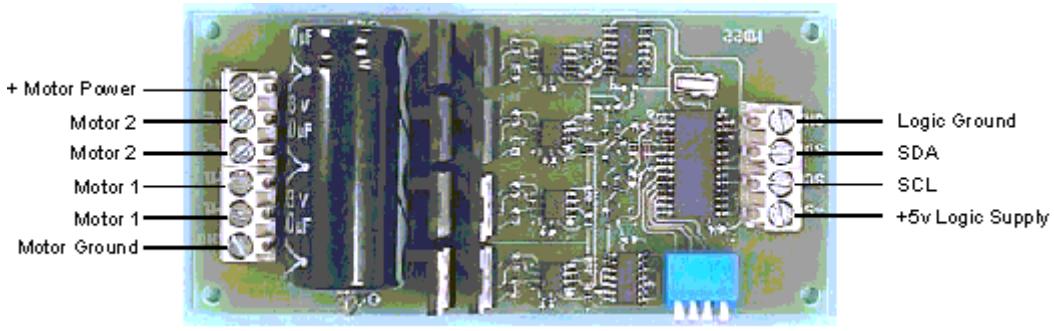

Figura 3.4.2 Placa do módulo de Driver M22.

3.5. MÓDULO DE ALIMENTAÇÃO

3.5.1. Baterias

Uma bateria é um dispositivo capaz de armazenar energia química e liberá-la na forma de energia elétrica. As características de uma bateria são: possibilidade de recarga, densidade de energia (máxima quantidade de energia por unidade de massa ou volume), quantidade de energia armazenada na célula (geralmente em A.h, ou mA.h), tensão de operação (em função dos elementos químicos utilizados), resistência interna, taxa de descarga (varia com a resistência interna), “shelf life”, ou seja, a medida de quão rapidamente a bateria perde a carga sem ser utilizada, e dependência da temperatura.

A análise de todas essas características são importantes para a escolha do tipo e o dimensionamento da bateria a ser utilizada no projeto. Dos principais tipos de baterias comerciais adequadas para esta aplicação (Níquel – Cádmio, Hidreto de Níquel Metal e Chumbo Ácido), as de NiCd, são indicadas para a maioria dos usos devido a sua diversidade de tamanhos, sua baixa resistência interna, sua selagem (não havendo risco de derramamento de material contaminante no interior do robô) e são relativamente baratas em relação as NiMH (que apresentam melhores características), e mais leves do que as de chumbo-ácido. Em contrapartida, estas baterias têm restrições técnicas para operação em grupo, apresentando uma baixa densidade de energia (comparável com as de chumbo ácido). Além disto, as baterias de NiCd possuem um “efeito de memória”, isto é, se forem recarregadas sem estarem completamente descarregadas, a nova carga não será total. Por exemplo, se uma bateria com 50% da carga for recarregada, a nova carga adquirida será de 50%. Por possuir baixa resistência interna, as baterias de NiCd possuem valores elevados de corrente de curto circuito (corrente de descarga), levando em determinadas situações a causar danos ao isolamento da bateria e até incêndio. A tabela 3.5.1 apresenta

algumas características das baterias de ácido-chumbo, lítio-Ion, Lítio-Ion Polímero, níquel-cádmio e hidreto de níquel-metal [11].

Tabela 3.5.1 Características de Baterias Diversas.

	NiCd	NiMh	Li-Ion	Li-Íon Polímero	Chumbo-Ácido
Densidade de Energia (Wh/kg)	45-80	60-120	110-160	100-130	30-50
Resistência Interna (miliOhm)	100-200 Pack 6V *(1)	200-300 Pack 6V *(1)	150-250 Pack 7,2V *(1)	200-300 Pack 7,2V *(1)	<100 Pack 12 V *(1)
Ciclo de Vida (80% da capacidade inicial)	1500 *(2)	500-1000 *(2)(3)	500-1000 *(3)	300-500	200-300 *(2)
Tempo para Carga Rápida	1 hora	2 a 4 hs	2 a 4 hs	2 a 4 hs	8 a 16 hs
Tolerância para Sobrecarga	Moderada	Baixa	Muito Baixa	Baixa	Alta
Auto-Descarga Mensal (na temperatura ambiente)	20% *(4)	30% *(4)	10% *(5)	10% *(5)	5%
Tensão da Célula	1,25V *(6)	1,25V *(6)	3,6V	3,6V	2V
Corrente de Carga -Pico - Melhor Resultado	20C 1C	5C 0,5C	>2C 1C	>2C 1C	5C - *(7) 0.2C
Temperatura de operação (somente descarga) *(8)	-40 a 60 ^{0}C	-20 a 60 ^{0}C	-20 a 60 ^{0}C	0 a 60 ^{0}C	-20 a 60 ^{0}C
Manutenção	30 a 60 dias	60 a 90 dias	Não é necessário	Não é necessário	3 a 6 meses *(9)
Comparação de Custo Pack 7,2V – U.S.A. *(10)	$ 50	$60	$100	$100	$25
Custo por ciclos *(11)	$0,04	$0,12	$0,14	$0,29	$0,10
Usada comercialmente desde	1950	1990	1991	1999	1970

(01) A resistência interna de uma bateria varia com a capacidade da célula, tipo de proteção e numero de células. Os circuitos de proteção para Li-Íon e Li-Íon Polímero adicionam 100 mili Ohms de resistência.

(02) O ciclo de vida é baseado na consideração que a bateria recebe o ciclo adequado de manutenção. A falha na aplicação de ciclos profundos de descarga pode reduzir a vida útil por três vezes.

(03) O ciclo de vida útil é baseado na profundeza da descarga. Descargas curtas permitem ciclos de vida mais longos.

(04) A descarga é imediatamente maior após a carga. A bateria NiCd descarrega aproximadamente 10% nas primeiras 24 horas e após descarrega 10% cada 30 dias. A autodescarga aumenta com a elevação da temperatura.

(05) Circuitos internos de proteção tipicamente consomem 3% da energia armazenada por mês.

(06) 1,25 V é a tensão de célula sem carga. 1,2 V é a tensão mais comum.

(07) Capaz de altas correntes pulsadas.

(08) Aplicado apenas à descarga; a temperatura de carga é mais restrita.

(09) A manutenção pode ser na forma de carga de equalização ou de pico.

(10) Custo das baterias para aplicações portáteis.

(11) Derivado do preço da bateria dividido pelo numero de ciclos.

Uma boa característica da bateria de lítio é a de manter praticamente constante o valor da tensão à medida que diminui sua capacidade (carga) com o tempo. No caso das baterias Alcalinas, Chumbo-Ácido e NiCd, o valor da tensão é mais afetado em relação ao estado de carga, diminuindo significativamente com o tempo.

A Tabela 3.5.2 mostra um resumo das capacidades e tempo de retenção de carga de diferentes baterias.

Tabela 3.5.2 Tipo de Bateria por Tempo de Retenção de Carga.

Tipo de Bateria	Tempo de Retenção de Carga	Curva de Descarga
Lítio	Alto tempo de retenção de carga em vazio (± 10 anos)	Plana
NiMH	Médio tempo de retenção em vazio	Inclinada
Chumbo-Ácido	Médio tempo de retenção em vazio	Inclinada
NiCd	Médio tempo de retenção em vazio	Inclinada

3.5.2. Reguladores de Tensão

Os reguladores de tensão são dispositivos que servem para fornecer uma tensão constante para o circuito, mesmo que a tensão de entrada varie sobre uma ampla faixa, compensando o efeito da redução da tensão fornecida pela bateria à medida que esta se descarrega.

Um regulador linear é um dispositivo constituído normalmente por 3 terminais: alimentação (cujo valor deve ser superior em alguns volts a tensão de saída desejada), terra e tensão de saída. Alimentando o regulador linear com uma tensão dentro da faixa de operação, a tensão de saída permanecerá sempre constante.

Um regulador linear como o LM7805, mantém a tensão de saída em 5 V para tensões de entrada variando na faixa de 7 a 35 V. Se a tensão de entrada for menor que 7 V, a tensão de saída não se mantém regulada.

Um outro regulador, o LM2940CT-5.0 opera com tensões de entrada na faixa entre 5.5 V e 26 V, mantendo a saída regulada em 5 V. Sua vantagem em relação ao LM7805 é a necessidade de precisar de apenas um acréscimo de 0,5 V na tensão de entrada em relação a saída para operar satisfatoriamente. Esta característica acarreta uma menor dissipação de potência pelo regulador [12].

3.5.3. Conversores CC-CC

Em um robô móvel são necessários vários níveis de tensão (+5 V, -12 V, +12 V, +24 V), enquanto que uma bateria só fornece um nível de tensão. Utilizando-se um conversor CC-CC, todos os níveis de tensão e polaridades necessárias podem ser obtidos, inclusive valores maiores que o nível de tensão fornecido pela bateria. Os conversores CC-CC podem ser construídos usando dois princípios: o primeiro é um conversor tipo fonte de carga, que funciona com a carga de capacitores em paralelo e a descarga destes em série, para gerar uma tensão maior, ou conectando-os com polaridade invertida para produzir uma tensão negativa a partir de uma fonte positiva. O segundo é um regulador de chaveamento que funciona através do princípio da indução de tensão provocada pela interrupção instantânea da corrente num indutor, gerando a indução de tensões elevadas (inclusive com valores maiores que a tensão da bateria). Desta forma, através de um esquema de chaveamento e filtragem adequado, é possível se obter os níveis de tensão desejados. Um regulador de chaveamento é um tipo de regulador de tensão mais eficiente do que os reguladores lineares, pois possui uma maior eficiência (normalmente acima de 80%). Sua principal desvantagem é seu custo elevado.

3.5.4. Proteção de Circuitos, Isolamento e Redução de Ruídos

Uma vez que os motores e determinados sensores (ultra-som) produzem muita interferência, torna-se necessário no projeto do robô o isolamento da parte lógica em relação aos sensores e circuitos de potência.

Os motores usados nos robôs móveis causam picos de tensão quando passam por uma seção do comutador. Para evitar esse efeito indesejável, uma solução é a colocação da bateria entre o motor e a parte lógica, bastando somente regular a tensão para alimentar os circuitos lógicos. Caso isto não seja suficiente para evitar problemas de funcionamento no circuito, outra solução é a colocação de um capacitor em paralelo com o circuito lógico e um diodo entre a bateria e este circuito. Outra medida, considerada mais sofisticada e eficiente seria a utilização de fontes de alimentação distintas. Uma última alternativa é a utilização de um isolador ótico, o que permitiria um verdadeiro isolamento entre o circuito lógico e os motores. Referências de chips comerciais para esta finalidade são: 4N25, 4N28 e 4N30.

3.5.5. Dimensionamento do módulo de alimentação utilizado no robô

Os picos de tensão, gerados pelos MOSFETs de chaveamento de corrente para os motores, provocam ruídos interferentes no módulo de Controle e

Comunicação por RF. Como forma de se evitar estas interferências, foi adotada a configuração com fontes de alimentação independentes.

Para alimentação dos motores dc, optou-se pela bateria do tipo Chumbo-Ácido, 12 Vdc, por ter baixa resistência interna e alta corrente de descarga. Um dos requisitos do planejamento da trajetória é evitar mudanças de velocidade em degrau que possam gerar um aumento súbito da demanda de corrente do motor e perda de precisão do percurso devido à patinação das rodas. Em caso de travamento do eixo a bateria deverá ser capaz de fornecer altas correntes em pequenos intervalos de tempo.

Para a alimentação do módulo de Controle e Comunicação por RF, o Encoder e o Módulo VCF optou-se por utilizar uma bateria de Lítio Polímero, 7,4 Vdc, por apresentar uma curva de descarga plana. Para manter a tensão regulada em 5 Vdc, foi utilizado o regulador de tensão, LM2940CT-5.0.

Com base nos dados anteriores e supondo que o robô desenvolve uma velocidade constante de 0,6 m/s e o coeficiente de atrito μ é 0,5, a potência instantânea exigida é:

$$P_i = \mu * m * g * v = 0{,}5 * 3{,}8\ (Kg) * 9{,}8\ (m/s^2) * 0.6\ (m/s) = 11{,}172\ Watts$$

$$Watts = \frac{Joules}{seg} = \frac{Volts * Coulombs}{seg} = \frac{Volts * Amps * seg}{seg} = Volts * Amps$$

Entretanto temos:

$$Volts * Amps * hrs = Watts * hrs$$

A expectativa de vida da bateria é dada por:

$$T_{bateria} = \frac{Volts * AmpHora}{Potência}$$

Temos então neste caso:

$$T_{bateria} = \frac{12 * 7}{11{,}172} = 7{,}51 horas$$

Existem imprecisões nesta análise porque não é desejável que o robô se movimente em velocidade constante durante todo o tempo de vida da bateria e a tensão da bateria tende a diminuir com o tempo.

Resumindo, a Tabela 3.5.3 apresenta as fontes, os reguladores e os valores estimados de carga e sua autonomia.

Tabela 3.5.3 Fontes, reguladores, cargas e autonomia para o módulo de alimentação.

Fonte	Regulador	Carga	Autonomia
Bateria 12 Vdc/ 7 Ah Chumbo-Ácido	-	Motores DC (2A) através do Módulo Driver	451 min
Bateria 7.4 Vdc/ 800 mAh Lítium Polímero	LM2940CT-5.0 5 Vdc	Módulo de Controle e Comunicação por RF PS (80 mA), Módulo Driver (60 mA), Encoder (60 mA)	240 min

3.6. SENSORES

Os sensores são dispositivos projetados para quantificar ou detectar parâmetros específicos por meio de elementos transdutores. Em um sensor, os transdutores são os elementos que desenvolvem a função de transformação de uma magnitude física em outra.

Os sensores podem ser de contato (interruptores, palpadores, provadores, etc.) ou não-contato (através de campos magnéticos, ondas sonoras, luz, raios-X, luz infra-vermelho, etc.) e podem ser divididos em duas classes básicas:

1. Sensores Internos - Dispositivos utilizados para medir posição, velocidade ou aceleração das juntas e/ou da extremidade de um robô manipulador ou das rodas de um robô móvel.

Os sensores internos são geralmente designados para as tarefas conhecidas como *"Dead Reckoning"*. A expressão *"Dead Reckoning"* deriva de *"Deduced Reckoning"* (Contagem Deduzida). *"Dead Reckoning"* é um procedimento para determinação da localização atual de um veículo através da análise das informações sobre sua velocidade e curso conhecido. Baseia-se na idéia de que se o curso é conhecido, é possível determinar a posição do veículo neste curso através da observação da velocidade de seu deslocamento. A implementação mais simplista de *"Dead Reckoning"* é conhecida como *odometri*a, onde a posição do veículo ao longo do caminho é derivada diretamente de algum tipo de odômetro embarcado. Outra forma de odometria bastante comum envolve codificadores ópticos diretamente acoplados à armadura do motor ou aos eixos das rodas [13].

São exemplos de sensores internos:

- Potenciômetros (pots)
- Sincros (synchros)
- Resolvers
- Escala Indutiva Linear
- Transformadores Diferenciais (LVDT e RVDT)
- Interruptores Óticos
- Encoders Óticos (absoluto e incremental)
- Tacômetros
- Acelerômetros

2. Sensores Externos - Utilizados para monitorar o próprio robô (auto-proteção) e/ou sua relação dinâmica com sua tarefa, ambiente ou objetos manipulados pelo mesmo. Podem ser visuais ou não-visuais.

Os sensores não-visuais podem ser aplicados para monitorar:

a) Distância a um objeto ou a um obstáculo

b) Toque/Escorregamento

c) Força/Torque

São exemplos de sensorers não visuais:

- Strain Gages
- Transdutores de Pressão
- Sensores de Proximidade
- Sensores de Ultra-Som
- Sensores Eletromagnéticos
- Materiais Elastométricos

Entre os sensores visuais estão os sistemas de visão artificial constituídos por câmaras de vídeo e dispositivos de processamento de imagens. Devido à gradativa redução de preço e peso das câmaras de vídeo, estas estão sendo cada vez mais utilizadas em robótica. A maior complexidade continua sendo o processamento da imagem, que exige computação intensiva, implicando em processadores mais caros e pesados a bordo do robô móvel. Uma alternativa é a transmissão da imagem para seu processamento fora do robô, por exemplo, através de cabos, embora o ideal seja um transmissor de vídeo, disponível no mercado com baixo preço e pequenas dimensões.

O desempenho no controle dos robôs depende da capacidade de obtenção de informação sobre a junta e/ou extremidade de um robô manipulador, ou sobre as rodas de um robô móvel. Para isso, torna-se necessário à utilização de dispositivos (transdutores) que forneçam tal informação que possa ser utilizada pelo robô. Assim, posição, velocidade e/ou aceleração (ou pelo menos uma representação analógica ou digital das mesmas) devem ser medidas para verificação da movimentação do robô da maneira desejada (por exemplo, em linha reta).

Embora seja possível utilizar um robô sem qualquer sensoriamento externo, muitas aplicações necessitam de sensores. Por exemplo, no manuseio de diferentes objetos, dos quais alguns podem ser frágeis, é importante fazer a medição da força exercida pela garra para ajustá-la num valor adequado que proporcione manusear o objeto sem quebrá-lo.

O robô objeto deste trabalho utiliza sensor interno, não-contato, do tipo encoder ótico incremental para medição da velocidade e distância percorrida.

3.6.1. Princípio de Funcionamento do Encoder Ótico Incremental

Os encoders ótico incrementais são amplamente utilizados para monitorar velocidade e posição em robôs, máquinas ferramentas, tornos, etc, por proporcionarem alta resolução e menor custo que os encoders absolutos. Entretanto, apresentam como desvantagem a necessidade de calibração prévia na ocasião de sua energização.

Tal como o encoder absoluto, o encoder ótico é constituído por um disco, uma fonte de luz (LED) e um conjunto de receptores de luz (fototransistores). Normalmente são utilizados um único LED, quatro fotodetectores e um disco para um único setor com n linhas radiais [14], conforme mostrado na Figura 3.6.1.

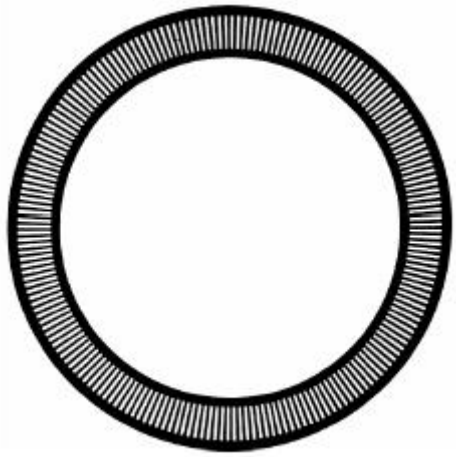

Figura 3.6.1 Disco com frestas simétricas.

A resolução do encoder está associada ao número de linhas. Desta forma, a resolução de posição angular é definida por 360º / n. Normalmente, são considerados como satisfatórios valores de resoluções menores que 0,175º. Em robótica, são geralmente aplicados valores entre 1,8º e 0,36º .

A forma de onda de saída é senoidal quando o encoder é montado sobre o eixo do motor, a luz que chega ao receptor é interrompida por cada linha do disco. Neste caso usa-se um comparador para converter esses sinais em pulsos TTL. São identificados dois problemas com a configuração de um emissor e um receptor: a impossibilidade determinar a direção de rotação e a dependência da largura dos pulsos gerados em relação à velocidade da rotação. Por isso, para que não ocorram problemas de detecção são empregados múltiplos receptores. Por exemplo, um segundo fotodetector afastado eletricamente em 90º produzirá pulsos idênticos ao primeiro com um determinado atraso, cujo valor servirá de calculado para se determinar a direção do movimento do disco [14]. A Figura 3.6.2 mostra a forma de onda de saída de um encoder ótico com 2 canais A e B. O canal A está adiantado em relação a B, o sentido de giro é horário.

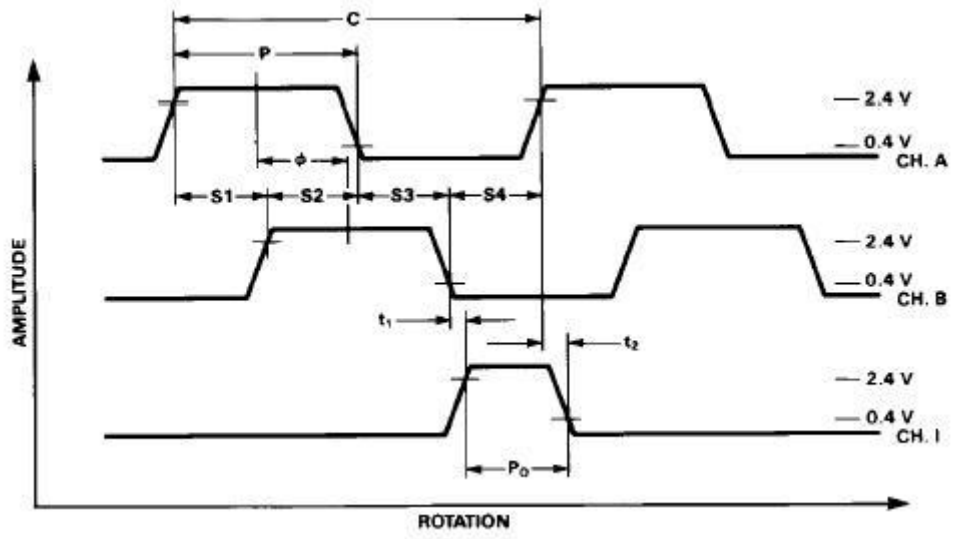

Figura 3.6.2 Formas de Onda de Saída do Encoder. Canal A e B.

Para evitar o problema da perda dos pulsos (quando operando em alta velocidade), utilizam-se dois fotosensores adicionais e duas retículas fixas (ou disco fixo) na frente do sensor de luz, conforme mostrado na Figura 3.6.3, [14].

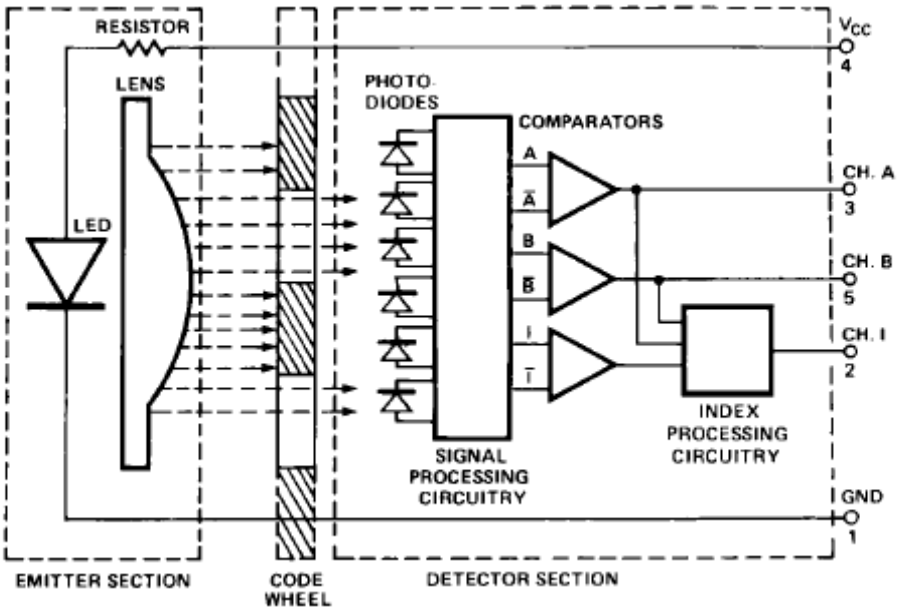

Figura 3.6.3 Diagrama em bloco do Encoder com os fotosensores adicionais.

Se as saídas dos dois fotodetectores são subtraídas, teremos uma forma triangular centrada em zero com aproximadamente duas vezes o valor de pico do sinal gerado.

Nesta , o encoder ótico é muito menos sensível as variações da fonte de luz (problemas no LED) ou a sensibilidade do fotodetector (temperatura elevada ou operação em alta-frequência) .

O uso de dois canais através de circuitos eletrônicos, além de permitir a determinação da direção de rotação e compensações para variações na fonte de luz e fotodetectores, permite um aumento na resolução do sensor em até 20 vezes.

Para determinar o zero de referência do encoder incremental, normalmente se utiliza um canal adicional para detectar uma linha de referência no disco e assim produzir um pulso a cada rotação do mesmo.

3.6.2. Medição da Velocidade

O encoder ótico além de ser utilizado como sensor de posição, também pode ser usado para medir velocidade; existem duas técnicas: a primeira usa o encoder e um conversor de freqüência para tensão (FVC-Frequency to Voltage Convert), gerando uma tensão analógica que é proporcional à velocidade. A segunda técnica utiliza o encoder e um software adequado para proporcionar uma representação digital da velocidade.

a) Encoder e Conversor Freqüência para Tensão (FVC)

Os pulsos produzidos pelo encoder podem ser contados periodicamente e este número de pulsos é convertido pelo FVC para um nível CC proporcional à velocidade do eixo do motor. O chip AD451 é utilizado para produzir tensões de 0 a 5 V nas freqüências de até 10kHz. O AD453 vai até 100kHz.

b) Encoder e Software

A amostra da velocidade pode ser obtida de duas maneiras distintas:

- Através da contagem de frestas ou pulsos durante um período de tempo fixo.
- Através da medida de tempo entre duas frestas ou pulsos consecutivos.

A obtenção da velocidade do motor é calculada através do seguinte algoritmo:

1. Ler e armazenar o valor atual do encoder: *P(kT)*
2. Recuperar o valor prévio do encoder: *P((k-1)T)*
3. Obter a velocidade aproximada

$$V(kT) = \frac{P(kT) - P((k-1)T)}{P(kT)}$$

4. Incrementar k e repetir os passos de 1 a 3.

Este algoritmo é de fácil implementação e rápida execução, mas trata-se uma aproximação de velocidade que pode não ter precisão adequada para determinadas aplicações. A escolha do valor de T não deve ser muito elevado (devido ao teorema da amostragem) e não muito pequeno devido ao baixo desempenho em baixas velocidades.

3.6.3. Encoder Ótico Incremental HEDS 5540 A06

O encoder utilizado, apresentado no ítem 4 da Tabela 3.1.1, fornece 500 pulsos por revolução, apresentando uma resolução de 360º / 500, ou seja, 0,72º. A partir do valor de velocidade máxima sem carga no eixo do motor – 5060,08 rpm (item 3.2.1), calcula-se o tempo entre frestas ou pulsos - TEP, número de pulsos por segundo - PPS e o número de pulsos a cada 42 ms - PPms gerados pelo encoder, conforme

Tabela 3.6.1. A partir da tabela é verificado que para um movimento linear da roda de 0,0001 mm/s, o encoder fornecerá um TEP de 1,518 mS, PPS de 658,86 e PPmS de 27,5.

Detalhes sobre as características elétricas e mecânicas do encoder HEDS 5545 A06 , inclusive os erros de Largura de Pulso, Fase, Posição e Ciclo, são obtidas no data sheet anexo a este trabalho.

Tabela 3.6.1 Determinação do tempo entre frestas do encoder.

Velocidade					**ENCODER**		
Vm - Velocidade máxima angular					**500 PPR**		
		Ang Motor	Ang Cx Red	Linear	**TEP**	**PPS**	**PPms**
		(rpm)	**(rpm)**	**(mm/s)**	**(ms)**		
v_m	1	5060,08	39,60	311,0172	0,024	42167,33	1697
v_m/	2	2530,04	19,80	155,5086	0,047	21083,67	880
v_m/	4	1265,02	9,90	38,8771	0,095	10541,83	440
v_m/	8	632,51	4,95	4,8596	0,190	5270,92	220
v_m/	16	316,26	2,47	0,3037	0,379	2635,46	110
v_m/	32	158,13	1,24	0,0095	0,759	1317,73	55
v_m/	64	79,06	0,62	0,0001	1,518	658,86	27,5
v_m/	128	39,53	0,31	0,0000	3,036	329,43	13,75
v_m/	256	19,77	0,15	0,0000	6,071	164,72	6,875

O MCU 13213 fornece através do temporizador TPM2, canais 1 e 2, os sinais PWM para as rodas direita e esquerda, RD e RE, respectivamente. Tendo como objetivo manter a velocidade desejada, o valor da velocidade medido é comparado com a velocidade desejada. Caso seja verificado erro, o ciclo de trabalho de saída do módulo PWM é modificado.

Os canais 3 e 4 do temporizador TPM2 são configurados para entrada, modo captura de sinal, recebendo os sinais dos sensores óticos (encoder incremental), canal A, das rodas direita e esquerda, respectivamente. Estes canais serão utilizados para medição da velocidade através da técnica descrita em 3.6.3, Encoder e Software.

4. CARACTERÍSTICAS DO ROBÔ (SOFTWARE)

4.1. SISTEMA OPERACIONAL EM TEMPO-REAL EMBUTIDO

Sistemas em tempo-real realizam tarefas de controle e processamento de informações, mas com um diferencial: suas respostas ao ambiente devem ser retornadas em tempo hábil ou o sistema irá entrar em um estado inconsistente de funcionamento.

Os sistemas em tempo-real podem ser encontrados em várias outras áreas de aplicação, como:

- Sistemas de navegação para veículos;
- Controle de tráfego aéreo;
- Sistemas de monitoramento de vida;
- Controle de processos industriais;
- Sistemas distribuídos de multimídia.

O fato de fornecerem resposta em tempo hábil requer que os sistemas em tempo-real tenham o comportamento determinístico. Além disso, é necessário que outros aspectos entrem na especificação de projetos em tempo-real. São eles:

- Previsibilidade - em vez de ser rápido (que é um termo relativo), a propriedade mais importante de um sistema em tempo-real deve ser sua previsibilidade, que se define como: seu comportamento funcional e temporal deve ser tão determinístico quanto impõe as especificações do sistema;
- Confiabilidade – está relacionada à exatidão no funcionamento do sistema, ou seja, a falha do sistema é que pode gerar uma resposta fora do tempo esperado;
- Desempenho – o sistema deve ser eficiente o suficiente para lidar com a complexidade inerente ao ambiente de comportamento em tempo-real;
- Compactação – geralmente as soluções de sistemas em tempo-real são embarcadas, o que implica recursos reduzidos de processamento, memória e fontes de alimentação.

Os sistemas em tempo-real podem ser classificados em relação ao cumprimento de seus prazos de resposta ao ambiente, os deadlines. Deadline é o tempo limite que o sistema deve responder e/ou atuar para garantir sua consistência, sendo uma meta que deve ser perseguida e alcançada sempre.

Alguns sistemas são mais tolerantes às perdas de deadlines, ou seja, eles continuam em funcionamento mesmo não retornando suas respostas em tempo

correto. É claro que o acúmulo de faltas pode levar o sistema para um estado inconsistente. Geralmente, esses sistemas contam o número de faltas ocorridas e, a partir de certo valor, acionam uma rotina de proteção e/ou correção. “Esses sistemas são chamados de soft real-time, em que a perda ocasional de um deadline é aceitável” [15].

Por outro lado, existem sistemas que um deadline não alcançado significa grande prejuízo econômico e/ou humano. Esses sistemas são do tipo hard real-time; um sistema de navegação de mísseis teleguiados é um bom exemplo do mesmo. O sistema deve ser constantemente realimentado com novas posições, de modo que possa desviar dos obstáculos ambientais e ainda atingir seu alvo. Caso o comando de reposicionamento não seja processado no tempo desejado, o míssel poderá colidir com uma montanha, ou pior, com habitações civis. “Se a ação deve acontecer absolutamente em um certo momento (ou em certo intervalo), tem-se um sistema em tempo-real hard ” [15].

Não se tem como caracterizar, de modo preciso, um sistema em tempo-real como hard ou soft. A maioria dos sistemas se encontra em algum lugar entre essas duas definições. Um sistema em tempo-real seria uma composição de tarefas hard e soft.

4.1.1. Conceitos de Sistemas em Tempo-Real

Sistemas pequenos de baixa complexidade são geralmente projetados conforme Figura 4.1.1. Estes sistemas são chamados de sistemas de Fundo/Frente ou super-laços. Uma aplicação consiste de um laço infinito que chama módulos ou funções para executar determinadas operações (Fundo). Rotinas de serviço de interrupções (ISR-Interruption Service Routine) controlam eventos assíncronos (Frente). O nível Frente é também chamado de *nível de interrupção* e o nível Fundo é chamado de *nível tarefa*. Operações críticas devem ser executadas por ISRs para garantir que estão lidando com o modo oportunidade. Devido a isto, as ISRs tem uma tendência de levar mais tempo do que elas deveriam. A informação que uma ISR torna disponível para uma função de Fundo não é processada até que a rotina de fundo obtenha sua vez de executá-la, no qual é chamada de resposta do *nível tarefa*. O pior caso para o tempo de resposta do nível tarefa depende de quanto tempo leva para que o laço de Fundo seja completado. Devido ao tempo de execução de códigos típicos não ser constante, o tempo para sucessivos passos através de uma porção do laço é não determinístico. Ademais, se é feito uma mudança no código, a temporização do laço é afetada.

A maior parte das aplicações de grande volume baseadas em microcontroladores (forno microondas, telefones, etc...) são projetadas com sistemas de Fundo/Frente. Também, nas aplicações baseadas em microcontroladores, seria melhor, do ponto de vista do consumo de energia, parar o processador e executar todo o processamento dentro de ISRs.

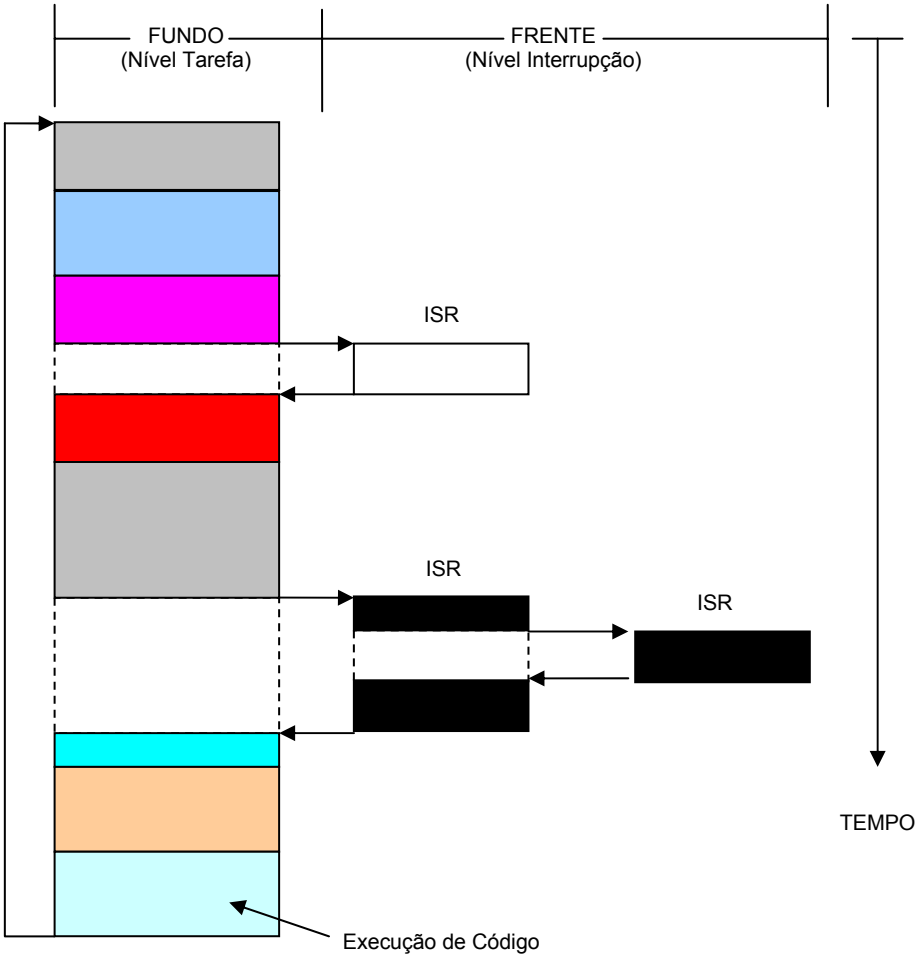

Figura 4.1.1 Sistemas de Fundo/Frente ou Super-Laços.

As definições abaixo são utilizadas em sistemas de tempo real.

4.1.1.1. Seções críticas de código

Uma seção crítica de código, também chamada de região crítica, é o código que precisa ser tratado indivisivelmente. Após a seção de código iniciar a execução, ela não deve ser interrompida. Para garantir que a execução não é interrompida,

interrupções são tipicamente desabilitadas antes do código crítico ser executado e habilitadas quando o código crítico é terminado.

4.1.1.2. Recursos

Um recurso é qualquer entidade utilizada por uma tarefa. Um recurso pode ainda ser um dispositivo de Entrada/Saída com teclado, display, uma variável, uma estrutura ou um array.

4.1.1.3. Recursos Compartilhados

Um recurso compartilhado é um recurso pode ser usado por mais do que uma tarefa. Cada tarefa deve ganhar acesso exclusivo a um recurso compartilhado para prevenir a corrupção de dados. Este processo é chamado de exclusão mútua.

4.1.1.4. Multitarefa

É o processo de planejamento e chaveamento da unidade central de processamento (CPU) entre várias tarefas; uma única chave de CPU atende várias tarefas seqüenciais. Multitarefa é como o sistema *Fundo/Frente* com múltiplos Fundos. Ela maximiza o uso da CPU e provê construção modular de aplicações.

4.1.1.5. Tarefa

Uma tarefa, também chamada de *thread,* é um simples programa que pensa que tem toda a CPU para si mesma. O processo de desenvolvimento de uma aplicação em *tempo-real* envolve a divisão do trabalho a ser feito dentro de tarefas responsável por uma porção do problema. A cada tarefa é designada uma prioridade, seu próprio conjunto de registros da CPU, e sua própria área da pilha, como mostrado na Figura 4.1.2.

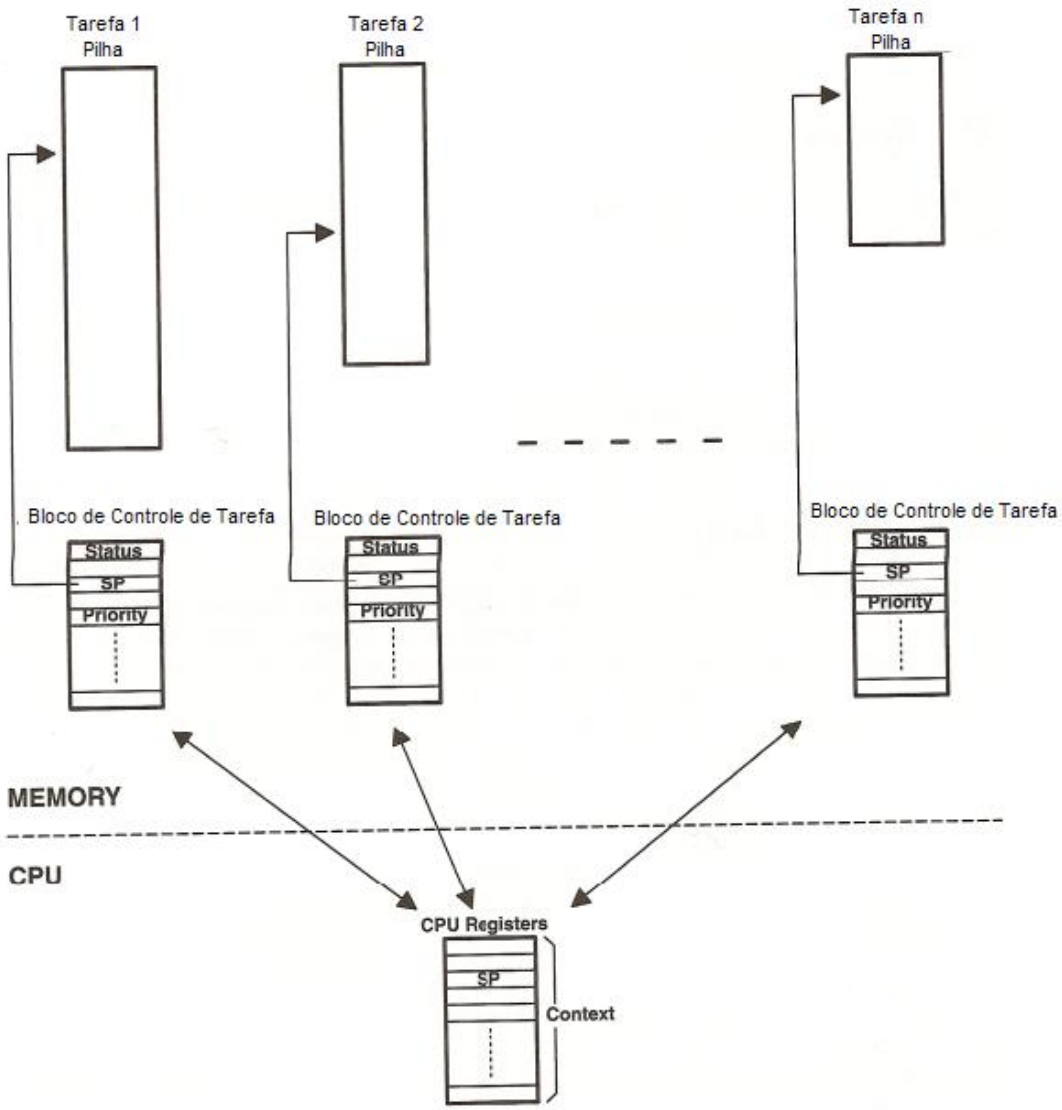

Figura 4.1.2 Divisão do trabalho em Tarefas.

Cada tarefa tipicamente é um laço infinito que pode estar em um de cinco estados: inativo, pronto, rodando, esperando, ou ISR (interrupção). O estado inativo corresponde a uma tarefa que reside na memória mas não tem sido feita disponível para o *kernel* multitarefa. Um tarefa está rodando quando ela tem controle da CPU. Uma tarefa está esperando quando ela requer a ocorrência de um evento (por exemplo, esperando que uma operação de Entrada/Saída seja completada). Finalmente, uma tarefa está no estado ISR quando uma interrupção ocorreu e a CPU está no processo de atendimento a interrupção. A Figura 4.1.3 mostra as funções fornecidas pelo MicroC/OS para fazer uma tarefa se mover de um estado para outro.

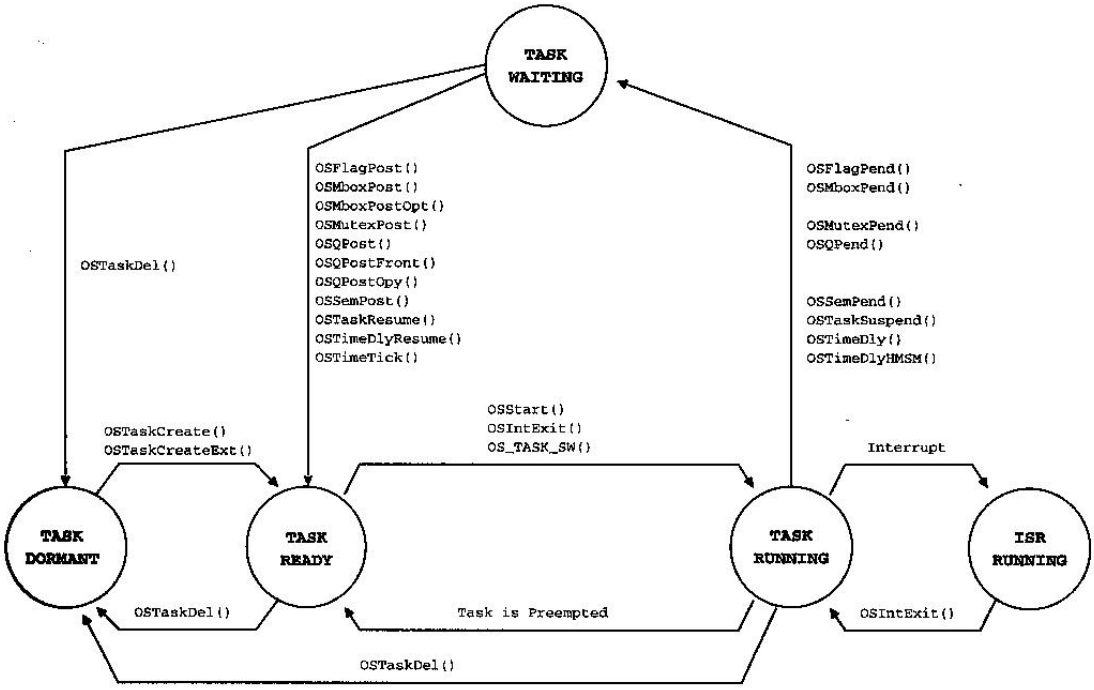

Figura 4.1.3 Funções do MicroC/OS.

4.1.1.6. Chaves de Contexto (Chaves de Tarefas)

Quando um *kernel* multitarefa decide rodar uma tarefa diferente, ele guarda o contexto da tarefa corrente na área de armazenamento de contexto da tarefa corrente – sua pilha. Após esta operação ser realizada, o novo contexto de tarefa é restaurado da sua área de armazenamento e então resume a execução do novo código de tarefa. Este processo é chamado de uma *chave de contexto* ou uma *chave de tarefa.* Chaves de contexto adicionam cabeçalho à aplicação. Quanto mais registradores uma CPU tem, maior é o cabeçalho. O tempo exigido para executar uma chave de contexto é determinado pela quantidade de registradores que tem que ser salvos e restaurados pela CPU. O desempenho de um *kernel tempo-real* não deve ser julgado pela quantidade de *chaves de contexto* que o kernel é capaz de executar por segundo.

4.1.1.7. Kernels

O kernel é a parte do sistema multitarefa responsável pelo gerenciamento de tarefas (gerenciamento de tempo da CPU) e comunicação entre tarefas. O serviço fundamental fornecido pelo *kernel* é a chaveamento de contexto. O uso de um kernel

de tempo-real geralmente simplifica o projeto de sistemas por permitir que a aplicação seja dividida dentro de tarefas múltiplas que o *kernel* gerencia.

Em uma aplicação bem projetada, um *kernel* utiliza entre 2 e 5% do tempo da CPU. Devido ao *kernel* ser um software que é adicionado a aplicação, ele exige memória ROM (espaço de código) extra e RAM adicional (espaço de dado) para as estruturas de dado, e cada tarefa exige seu próprio espaço de pilha, o qual se alimenta rapidamente da RAM.

Microcontroladores de chip simples não estão geralmente aptos a rodar um *kernel de tempo-real* porque eles têm pouca RAM. Um *kernel* permite fazer um melhor uso da CPU devido ao fornecimento de serviços indispensáveis, como o gerenciamento de semáforos, caixas de correio, filas, e atraso de tempo.

4.1.1.8. Schedulers

O scheduler, também chamado de despachador, é a parte do kernel responsável por determinar qual a próxima tarefa a ser executada. Muitos *kernel de tempo-real* são baseados em prioridades. A cada tarefa é designada uma prioridade baseada em sua importância e isto é específico para cada aplicação. O controle da CPU é sempre dado a tarefa de mais alta prioridade pronta para rodar. Existem dois tipos de kernel baseado em prioridade: *não-preemptivo e preemptivo.*

4.1.1.9. Kernels Não-Preemptivos

São aqueles que exigem que cada tarefa faça alguma coisa para explicitar a liberação da CPU. Um *kernel não-preemptivo* permite que cada tarefa rode até que voluntariamente libere o controle da CPU. Uma interrupção suspende uma tarefa. Após o término da ISR, a ISR retorna a tarefa interrompida. A resposta ao *nível-tarefa* é muito melhor do que no sistema de Fundo/Frente mas é ainda não determinístico.

4.1.1.10. Kernels Preemptivos

São utilizados quando a responsividade do sistema é importante; Por esta razão, MicroC/OS e a maioria dos *kernel de tempo-real* são preemptivos. A tarefa de prioridade mais alta pronta para rodar está sempre com o controle da CPU. Quando uma tarefa faz uma tarefa de prioridade mais alta pronta para rodar, a tarefa corrente é suspensa, e é imediatamente dado o controle da CPU a tarefa de prioridade mais alta. Se uma ISR faz uma tarefa de prioridade mais alta pronta, quando a ISR é terminada,

a tarefa interrompida é suspensa, e a nova tarefa de prioridade mais alta é reassumida.

4.1.1.11. Funções Reentrantes

Uma função reentrante pode ser usada por mais de uma tarefa sem o medo da corrupção do dado. Uma função reentrante pode ser interrompida a qualquer momento e reassumida mais tarde sem a perda do dado. Funções reentrantes também utilizam variáveis locais (registradores da CPU ou variáveis na pilha) ou protegem dados quando variáveis globais são utilizadas.

4.1.1.12. Fatiamento de Tempo (Time Slicing)

Quando duas ou mais tarefas tem a mesma prioridade, o kernel permite que uma tarefa rode por um predeterminado tempo, chamado de *quantum*, e então seleciona outra tarefa. Este processo é chamado de Fatiamento de Tempo. O kernel fornece o controle à próxima tarefa na linha se:

- A tarefa corrente não tem trabalho a executar durante a sua fatia de tempo;
- A tarefa corrente é terminada antes do fim da sua fatia de tempo ou
- A fatia de tempo termina.

MicroC/OS não suporta o fatiamento de tempo. Cada tarefa deve ter uma única prioridade na aplicação.

4.1.1.13. Prioridade das Tarefas

Uma prioridade é determinada para cada tarefa. A tarefa de maior importância é dada a prioridade mais alta. O desenvolvedor é responsável por determinar a prioridade de cada tarefa.

4.1.1.14. Prioridades Estáticas

Tarefas prioritárias são estáticas quando não mudam durante a execução da aplicação. Todas as tarefas e suas restrições de temporização são conhecidas no momento da compilação em um sistema onde as prioridades são estáticas.

4.1.1.15. Prioridades Dinâmicas

Tarefas prioritárias são dinâmicas se a prioridade da tarefa pode ser alterada durante a execução da aplicação. Esta é uma característica desejável de um *kernel de tempo-real* pois evita inversões de prioridade. MicroC/OS possui esta característica.

4.1.1.16. Inversões de Prioridade

É um problema em sistemas de *tempo-real* e ocorre quando se utiliza um kernel de *tempo-real*. Um kernel multitarefa deve permitir que as prioridades das tarefas mudem dinamicamente para ajudar a prevenir a inversão de tarefas. Uma forma automática de mudar a prioridade de uma tarefa é a utilização do método chamado *herança de prioridade*, obtido por Semáforos de Exclusão Mútua.

4.1.1.17. Determinação da Prioridade da Tarefa

A determinação da prioridade de uma tarefa não é algo trivial devido à natureza complexa de sistemas de *tempo-real*. Em muitos sistemas nem todas as tarefas são consideradas críticas. Tarefas não críticas obviamente devem ter prioridades mais baixas.

Uma técnica interessante chamada de planejamento de taxa monotonica (RMS) tem sido estabelecida para determinar a prioridade das tarefas baseado na taxa de execução da tarefa no tempo. Tarefas com altas taxas de execução são definidas com de alta prioridade.

4.1.1.18. Exclusão Mútua

O caminho mais fácil para as tarefas se comunicarem é através de estruturas de dados compartilhadas. Este processo é especialmente fácil quando todas as tarefas existem em um espaço de endereço simples e podem referenciar elementos , como varáveis globais, ponteiros, buffers e listas encadeadas. Embora o compartilhamento de dados simplifica a troca de informações, é necessário assegurar que cada tarefa tem acesso exclusivo ao dado para evitar contenção e corrupção de dado. Os métodos mais comuns de se obter acesso exclusivo a recursos compartilhados são:

- Desabilitar e Habilitar Interrupções – A maneira mais fácil e rápida de ganhar acesso exclusivo a um recurso compartilhado é através da habilitação e desabilitação de interrupções. MicrC/OS utiliza esta técnica para acessar varáveis internas e estruturas de dados. Duas macros estão disponíveis para este fim: OS_ENTER_CRITICAL() e OS_EXIT_CRITICAL(). Não se deve desabilitar interrupções por muito tempo pois pode afetar o tempo de resposta do sistema a interrupções. É aconselhável a utilização deste método somente quando poucas variáveis são trocadas ou copiadas. Também, este método é a única maneira de uma tarefa compartilhar varáveis e estruturas de dados com uma ISR.
- Executar operações de Teste e Set – Quando não se utiliza um Kernel, duas funções podem concorrer a um mesmo recurso. Para permitir o acesso por uma única tarefa, primeiramente é checado se uma variável global é 0, e caso seja verdadeiro a função tem acesso ao recurso. Para prevenir que outra tarefa tenha acesso ao recurso, a primeira função que obtém acesso ao recurso, seta a variável para 1. Esta operação é chamada de Teste e Set.
- Desabilitando e Habilitando o Planejador - Se a tarefa não estiver compartilhando varáveis ou estruturas de dados com um ISR, é possível desabilitar e habilitar o planejador. Neste caso, duas ou mais tarefas podem compartilhar dados sem a possibilidade de contenção. Enquanto o planejador estiver habilitado, as interrupções estarão habilitadas, e , se uma interrupção ocorrer enquanto estiver em uma região crítica, a ISR é executada imediatamente. No término da ISR, o kernel sempre retorna para a tarefa interrompida, mesmo se a ISR tiver feito uma tarefa de prioridade mais alta pronta para rodar. Embora este método funcione bem, é melhor evitar desabilitar o planejador porque ele afeta o propósito de se ter um kernel em primeiro lugar.

- Utilização de Semáforos – O semáforo foi inventado por Edgser Dijkstra em meados de 1960. É um mecanismo de protocolo oferecido pela maior parte dos kernels multitarefas. Semáforos são utilizados para controlar o acesso a um recurso compartilhado (exclusão mútua), sinalizar a ocorrência de um evento e permitir que duas tarefas sincronizem suas atividades. Um semáforo é uma chave que o código adquire com o objetivo de continuar sua execução. Se o semáforo já estiver em uso, a tarefa solicitante é suspensa até que o semáforo é liberado pelo detentor corrente. Existem dois tipos de semáforo: o Binário e o de Contagem. Semáforo Binário utiliza dois valores, 0 ou 1. Um Semáforo de Contagem permite valores entre 0 e 255, 65535 ou 4292967295, dependendo se o semáforo é empregado usando 8, 16 ou 32 bits, respectivamente. Geralmente, somente três operações podem ser executadas pr semáforos: Inicialização (também chamado de Criar), Espera (também chamado de Pendente) e Sinal (também chamado de Post). O valor inicial do semáforo deve ser fornecido quando o semáforo é inicializado.

4.2. COMUNICAÇÃO WIRELESS

Existem três protocolos Wireless (conexão sem fio) disponíveis para utilização no kit MC13213 SRB da empresa Freescale Semiconductors:

a) SMAC (Controle de Acesso ao Meio Simples)

Ideal para o desenvolvimento de comunicação ponto a ponto é o que possui a pilha mais simples e permite enviar e receber pacotes via Wireless. Possui pequena ocupação de memória (aproximadamente 3 Kb).

b) IEEE 802.15.4 – Compatível com MAC (Controle de Acesso ao Meio)

Suporta as topologias físicas de rede estrela, totalmente ligada e árvore. Suporta também redes GTS e demarcada. Permite modo de economia de energia múltipla (hibernação , sono ocioso).

c) ZigBee – Compatível com a pilha da rede

Suporta as topologias físicas de rede estrela, totalmente ligada e árvore. Permite o padrão de segurança com criptografia avançado (AES-Advanced Encripted Security) 128-bits.

Para a comunicação entre as duas placas sensoras MC13213 (SRB) são utilizados dois códigos de projetos distintos:

- "SMAC_Snifer" - Responsável por realizar uma ponte entre a porta USB e o SMAC. Este código é carregado na placa sensora MC13213 (SRB) e conectada ao PC via cabo USB. O PC possui o aplicativo, algoritmo A Estrela, responsável por gerar o caminho, com base no mapa representativo do ambiente com obstáculos e pontos de origem e destino. Este aplicativo deverá ainda enviar pela porta serial USB, COM 4, a seqüência de até no máximo 85 trincas de números por vez, conforme protocolo de comunicação definido em 4.2.1
- "SimpleWirelessRxTx" – Responsável por colocar o MODEM no modo recepção durante todo o tempo e enviar o caracter “K” cada vez que um quadro é recebido via Wireless, indiferentemente se o caracter tenha sido recebido. Este código é adicionado à aplicação do sistema operacional em tempo real e carregado na placa sensora MC13213 (SRB) instalada no robô.

4.2.1. SMAC – Controlador de Acesso ao Meio Simples

O SMAC é um código simples baseado na linguagem ANSI C disponível como código fonte exemplo, para desenvolvimento de aplicações proprietárias de transmissão e recepção em RF, usando o circuito integrado MC13213 [16]. O SMAC foi construído para trabalhar com qualquer HCS08 MCU com uma SPI (Interface Serial entre Periféricos), mas pode ser adaptado para quase todos os tipos de núcleo processadores.

As características do SMAC são:

- Código Reduzido – Utiliza 2Kb da memória flash e 10 bytes (mais o comprimento máximo do pacote) de memória RAM;
- Enlace de comunicação por RF bidirecional proprietário e de baixa potência;
- IRQ de baixa prioridade;
- Transferência de pacote sem utilizar os recursos do MCU. O pacote único permite dado útil de até 123 bytes e possui detecção de erro automática (FCS-Frame Check Sequence);
- A estrutura do arquivo suporta a portabilidade do SMAC para outras arquiteturas.

A estrutura do pacote SMAC é mostrada na Figura 4.2.1.

.4 bytes	1 byte	1 byte	2 bytes	Até 123 bytes	2 bytes
Preâmbulo	SFD	FLI	Código	Payload Data	FCS

Figura 4.2.1 Estrutura do pacote SMAC.

Onde:

SFD – Detecção de início de quadro.

FLI - Indicador de tamanho do quadro.

FCS - Cheque da Seqüência do Quadro

O diagrama em bloco do SMAC é mostrado na Figura 4.2.2.

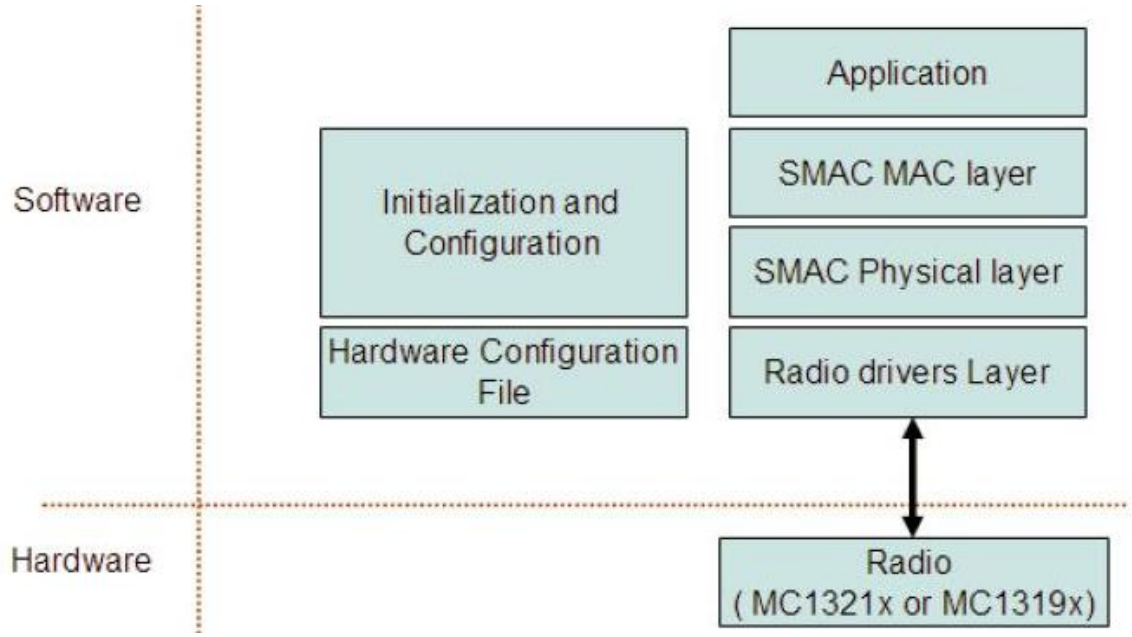

Figura 4.2.2 Diagrama em bloco do SMAC.

O SMAC é dividido em quatro camadas básicas:

1- Drivers

2- Física

3- MAC

4- Aplicação

4.2.1.1. Camada Drivers Rádio

É a acamada mais baixa que comunica diretamente com o hardware do rádio. Ela contém o código que habilita a funcionalidade do rádio como no manual de referência do MC 13213. O driver implementa uma funcionalidade de leitura e escrita da SPI para ler e escrever nos diversos registradores definidos no hardware do radio. Adicionalmente, a Camada Drivers Rádio um modo de escrita/ leitura em rajada no qual permite ao SMAC preencher um buffer Tx para transmissão, e ler o buffer Rx quando um pacote é recebido. A Camada Drivers Rádio também controla e processa todas as interrupções geradas pelo rádio. Alguns exemplos das funções da Camada Drivers Rádio são:

- Controlador da IRQ do Rádio que controla:

TX_DONE IRQ

RX_DONE IRQ
ATTN IRQ
TIMER1 IRQ
CCA_DONE IRQ
LO_LOCK IRQ
DOZE IRQ

- Requisição de acordar no qual ativa a linha ATTN no radio.
- Habilita e desabilita as interrupções do Rádio.
- Leitura e escrita na SPI.

4.2.1.2. Camada Física

Esta camada fornece uma API para a camada MAC e implementa comandos que agrupam chamadas a driver relativas as primitivas físicas. Estas primitivas estão relacionadas aos modos de hardware e suas funcionalidades, mas não comunicam diretamente com o rádio mas sim via chamadas do driver. Exemplos de primitivas para a Camada Física são:

- PLMEPhyReset
- PDDataRequest
- PDDataIndication
- PLMEHibernateRequest
- PLMESetChannelRequest

4.2.1.3. Camada MAC

Esta camada serve como um lugar de retenção para usuários adicionarem características de gerenciamento de um canal. Por exemplo, como selecionar um canal ou como adicionar um TX com um ACK. A maioria das primitivas da Camada MAC simplesmente chamam uma primitiva da Camada Física sem gerenciar o canal. Se o custo é o objetivo primário, então um pequeno e simples esquema de gerenciamento de canal pode ser empregado. Se a latência é um parâmetro chave, então algum esquema de gerenciamento de divisão de tempo seria apropriado. O SMAC fornece aos desenvolvedores acesso direto ao rádio em um ambiente com todo o código fonte pronto. Exemplos de primitivas para a Camada MAC são:

- MCPSDataRequest
- MLMEHibernateRequest
- MLMESetChannelRequest
- MLMELinkQuality

4.2.1.4. Camada Aplicação

Esta camada chama as primitivas da Camada MAC para habilitar uma específica aplicação do usuário. A camada de aplicação também controla qualquer rede outra característica não fornecida pelo SMAC. A camada de aplicação deve implementar funções de chamada de retorno que são exigidas pelo SMAC. Estas funções de chamada de retorno permitem que as camadas mais baixas notifiquem a Camada Aplicação que um evento ocorreu.

4.2.2. Protocolo de comunicação

Este protocolo faz com que o robô execute um trecho de cada vez, constituído de trincas de números, gerado pela aplicação no PC, algoritmo A Estrela. A execução deste protocolo de comunicação constitui a segunda tarefa do sistema de controle Fundo(*nível tarefa)*/Frente(*nível de interrupção*).

Para teste e verificação, as trincas de números podem ser geradas manualmente no aplicativo Hyperterminal do Windows através de transferência de arquivo texto (.txt) com 10 linhas, seguindo a regra abaixo. A cada caracter enviado pelo arquivo texto, o MCU envia o caracter "K" de volta para o aplicativo hyperterminal do PC, indicando que o caracter foi recebido. A aplicação deverá reenviar o caracter caso não receba do MCU o caracter "K". O Hyperterminal deverá ser configurado da seguinte forma:

- Porta Serial: COM4
- Bps: 38400
- Data bits: 8
- Parity: none
- Stop bits: 1
- Flow Control: none
- Character delay: 80ms
- Line dealy: 120 ms

O arquivo texto (.txt) deverá ter a seguinte seqüência de trinca de números:

#*tempo*vrd*vre*$ // Primeira trinca de números
#*tempo*vrd*vre*$ // Segunda trinca de números
...
#*tempo*vrd*vre*$@ // Décima trinca de números

Onde:

- # - caracter que indica o início da trinca;
- * - caracter que indica separação entre números;
- $ - caracter que indica o fim da trinca;
- @ - caracter que faz o robô executar o trecho;
- tempo - Tempo de execução do trecho. Variável inteiro sem sinal com ate 5 dígitos, mínimo valor de 1 (1 x 20ms = 20 ms) e máximo valor de 65535(65535 x 20 ms = 1310 s);
- vrd - Velocidade da roda direita em rotações por minuto (rpm). Variável do tipo inteiro sinalizada com até 4 dígitos, -1700(SAH) < vrd (Parado = 0) < +1700 (SH);
- vre - Velocidade da roda esquerda em rotações por minuto (rpm). Variável do tipo inteiro sinalizada com até 4 dígitos, -1700(SAH) < vrd (Parado = 0) < +1700 (SH).

A faixa de valores de SP desejado de vrd e vre (0 a 1700) foram obtidos da coluna PPms (quantidade de pulsos ou frestas a cada 42 ms) da Tabela 3.6.1.

Os valores de dutyrd e dutyre foram obtidos através de normalização com base no valor carregado no registrador contador de módulo (TPM2MOD) e na freqüência de clock do MCU 13213, no caso 4 MHz. O valor necessário para gerar um período de 20 ms, ou seja 50 Hz, é definido pela equação 4.2.1.

$$T = \frac{\text{Prescaler} * (\text{TPM2MOD} + 1)}{F_{FONTE}} = \frac{2 * (40000 + 1)}{4000000} = 0{,}02s \qquad (4.2.1)$$

Para gerar o pulso de largura variável entre 1 ms e 2 ms necessários para o driver M22, que fornecerá a máxima velocidade no sentido horário e anti-horário do motor, basta fazer com que o contador de módulo conte até o valor que reproduza este tempo de pulso em nível alto. Para isto é necessário carregar o registrador de comparação TPM2CXV, ou seja:

$$\text{TPM2CXV} = \frac{T * F_{FONTE}}{\text{Prescaler}} - 1 = \frac{0.001 * 4000000}{2} - 1 = 1999 \qquad (4.2.2)$$

Para o pulso de 2 ms o valor de TPM2CXV é 3999. A equação de conversão é definida por 4.2.3, válida também para RE (Roda Esquerda).

$$\text{TPM2CXV} = 2900 + \text{dutyrd} * \frac{10}{17} \qquad (4.2.3)$$

O fluxograma do protocolo de comunicação embarcado no kit da Freescale MC13213 instalado no robô é mostrado na Figura 4.2.3.

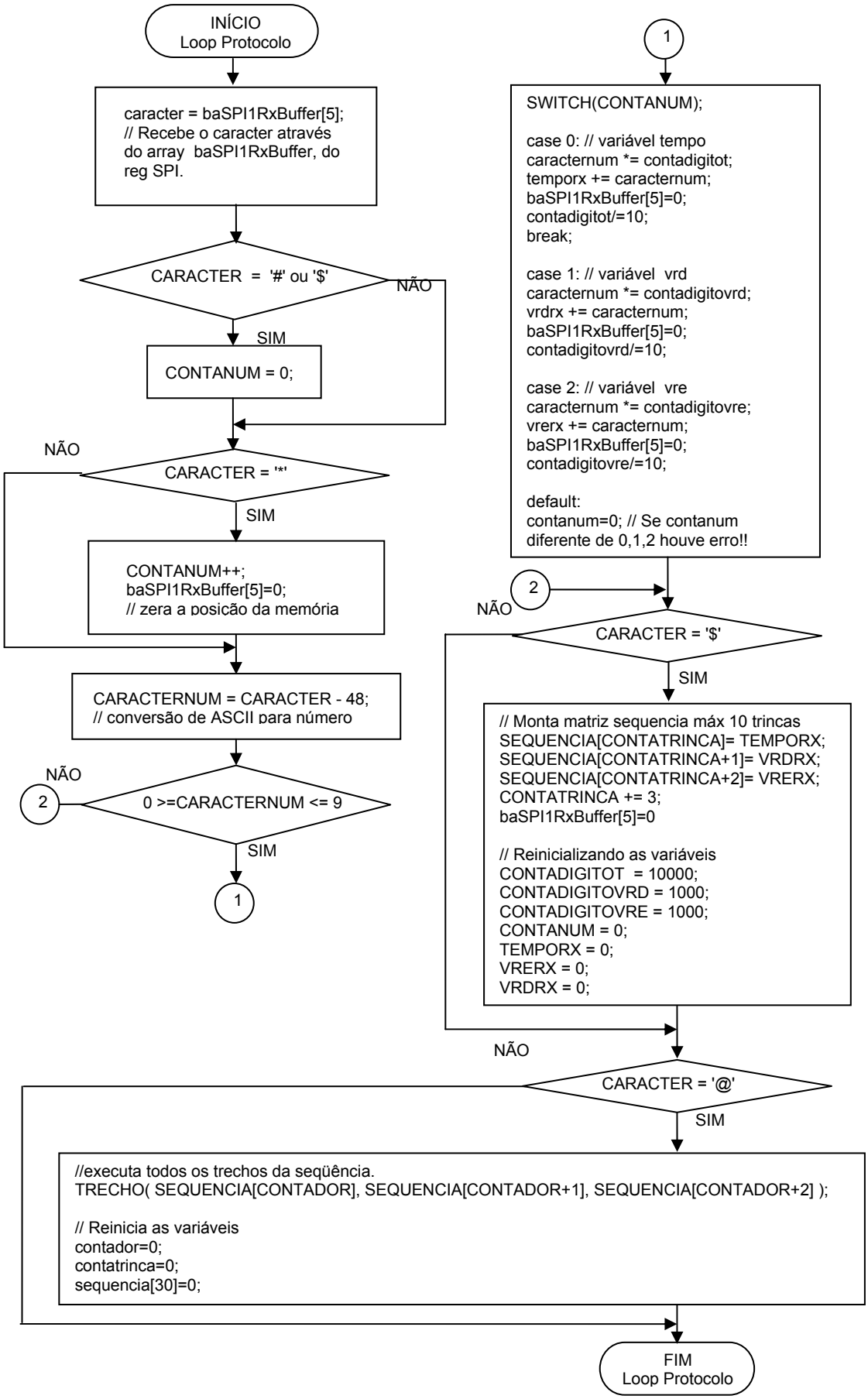

Figura 4.2.3 Fluxograma do protocolo de comunicação.

4.3. CONTROLE DE VELOCIDADE DE MOTOR DC COM ESCOVAS.

4.3.1. Visão geral do sistema de controle de velocidade

O sistema tem quatro elementos básicos, como mostrado na Figura 4.3.1.

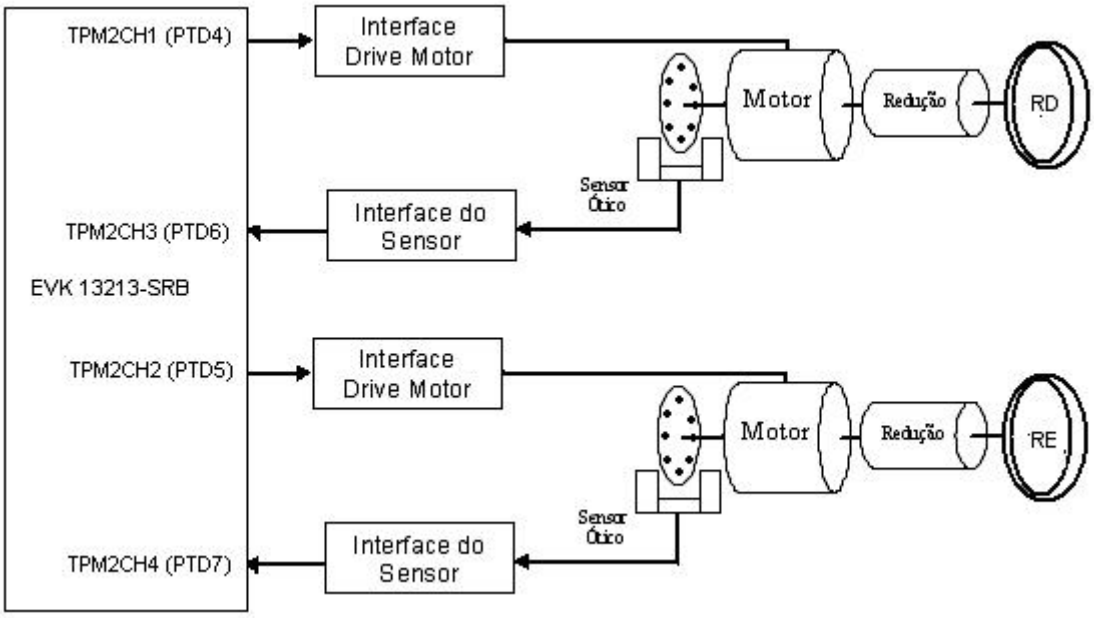

Figura 4.3.1 Diagrama em Blocos do Sistema de Controle de Velocidade.

O primeiro bloco representa a placa de avaliação EVK 13213- SRB que provê as funções de controle e processamento. O temporizador TPM2 no MCU gera um sinal PWM no qual é conectado ao segundo elemento, ou seja, módulo de interface drive do motor. A lógica do módulo de interface do motor recebe o sinal do PWM que realiza o chaveamento dos transistores de potência, conectados em ponte-H, para excitar o motor em 12 Vdc e Imáx 2 A. O motor juntamente com a caixa de redução é o terceiro elemento do sistema. O quarto elemento consiste de um sensor ótico e uma placa sensora que condiciona a saída do sensor de tal maneira que o período de rotação do eixo possa ser medido pelo software, para fechar a realimentação da malha de controle. A Figura 4.3.2 mostra o circuito de condicionamento do sinal do sensor.

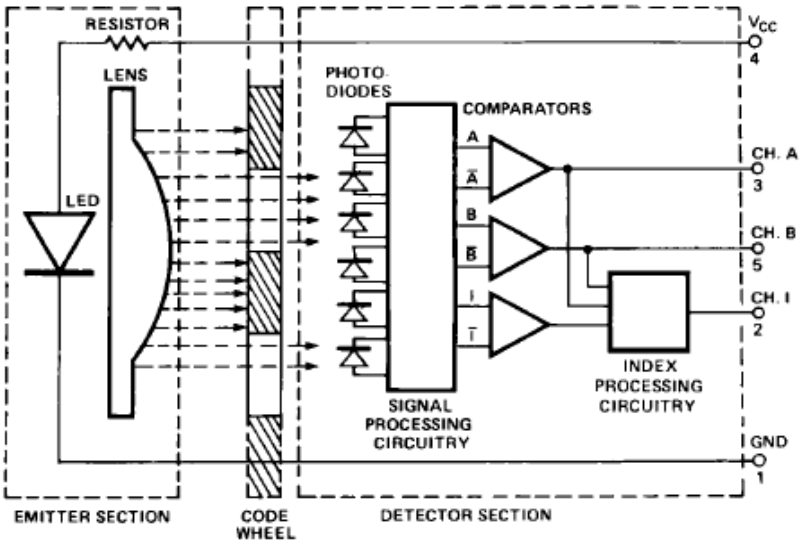

Figura 4.3.2 Circuito de condicionamento do sinal do sensor (ENCODER HEDS 5500 A06).

4.3.2. Modelo elétrico do motor DC

A modelagem matemática de um motor DC com engrenagens de redução pode ser realizada a partir de um circuito elétrico equivalente simplificado como o mostrado na Figura 4.3.3. Os seguintes passos são seguidos para realizar a modelagem e simulação em ferramenta de software própria para este fim (Matlab):

1. Descrição das equações do sistema;
2. Determinação da Função de Transferência;
3. Escolha do controlador e sintonia para o objetivo específico;
4. Análise dos resultados do simulador.

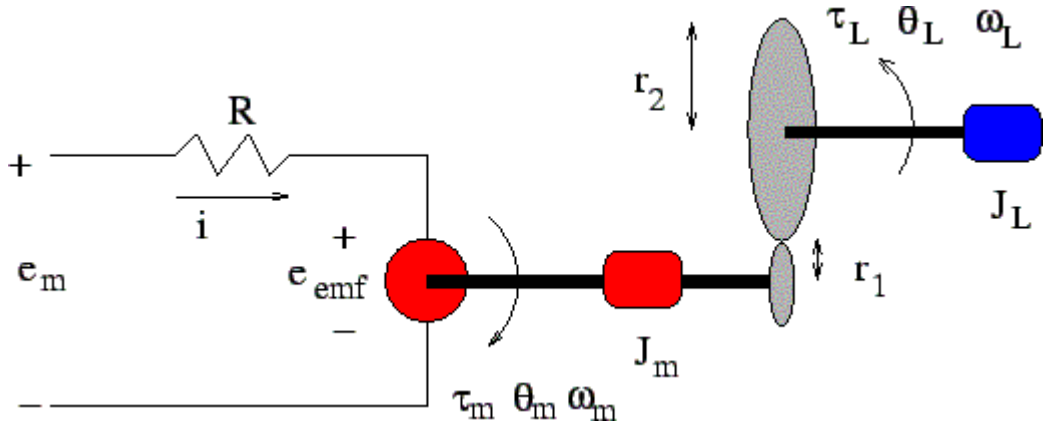

Figura 4.3.3 Modelo simplificado do Motor DC com engrenagens de redução.

4.3.2.1. Descrição das equações do sistema

A indutância de armadura foi considerada desprezível (μH) e por isto o modelo resultante é de primeira ordem. As equações do sistema são descritas a seguir, onde:

e_m – tensão de entrada;

i - corrente do motor;

R – resistência do motor;

e_{emf} – força contra eletromotriz devido ao giro do eixo do motor;

$K_g = r_2/r_1$ – razão de redução;

J_m – inércia do motor;

J_L – inércia da carga conectada ao eixo;

T_m e T_L – torque exercido pelo motor e torque exercido pela carga;

K_i - constante de torque do motor;

K_s - constante de velocidade;

ω_m e ω_L – velocidades do eixo do motor e do eixo da redução;

θ_m e θ_L – ângulos do eixo do motor e do eixo da redução.

.

A relação entre T_m e T_L é dada por:

$$\tau_L = K_g \tau_m \tag{4.3.1}$$

A relação entre ω_m e ω_L é dada por:

$$\omega_L = \frac{1}{K_g} \omega_m \tag{4.3.2}$$

O torque exercido pelo motor é proporcional a corrente do motor:

$$\tau_m = K_i i \tag{4.3.3}$$

A força contra eletromotriz é proporcional a velocidade angular do eixo do motor:

$$e_{emf} = K_s \omega_m \tag{4.3.4}$$

Aplicando a lei das tensões de Kirchoff ao circuito mostrado na Figura 4.3.1 temos:

$$e_m = iR + e_{emf}$$

Usando 4.3.3 e 4.3.4 obtém-se:

$$e_m = \frac{R}{K_i}\tau_m + K_s\omega_m$$

Usando 4.3.1 e 4.3.2 para refletir a carga obtém-se:

$$e_m = \frac{R}{K_g K_i}\tau_L + K_g K_s \omega_L \tag{4.3.5}$$

A inércia total vista pela carga é dada por:

$$J = J_L + K_g^2 J_m \tag{4.3.6}$$

Pela segunda Lei de Newton temos:

$$J\dot{\omega}_L = \tau_L \tag{4.3.7}$$

Ao utilizar 4.3.5 obtém-se:

$$\dot{\omega}_L = -\frac{K_g^2 K_i K_s}{JR}\omega_L + \frac{K_g K_i}{JR}e_m = -\frac{K_g^2 K_i K_s}{R(J_L + K_g^2 J_m)}\omega_L + \frac{K_g K_i}{R(J_L + K_g^2 J_m)}e_m$$

Definindo K_1 e K_2 como:

$$K_1 = \frac{K_g K_i}{R(J_L + K_g^2 J_m)}$$

$$K_2 = \frac{K_g^2 K_i K_s}{R(J_L + K_g^2 J_m)}$$

A dinâmica do motor, incluindo caixa de redução e carga, pode ser escrita como:

$$\dot{\omega}_L = -K_2\omega_L + K_1 e_m \tag{4.3.8}$$

Onde e_m é a tensão nos terminais do motor, e ω_L é a velocidade angular do eixo na carga. Como a velocidade angular é obtida através do encoder que está acoplado ao eixo do motor, a equação anterior pode ser escrita como:

$$\dot{\omega}_m = -K_2\omega_m + K_3 e_m \tag{4.3.9}$$

Onde:

$$K_3 = K_1 K_g$$

As constantes K_1 e K_2 podem ser determinadas pela identificação do sistema ou pela leitura dos valores das folhas de especificação e computando uma razoável estimativa de J_L.

Alternativamente, se considerarmos a corrente ***i*** como entrada, a partir de 4.3.7, 4.3.3 e 4.3.1 obtém-se:

$$J\dot{\omega}_L = K_g K_i i$$

No qual utilizando 4.3.6 fornece:

$$\dot{\omega}_L = \frac{K_g K_i}{J_L + K_g^2 J_m} i \tag{4.3.10}$$

O desafio é derivar uma razoável estimativa de J_L para o robô. O conjunto, motor, caixa de redução e roda do robô, é mostrado na Figura 4.3.4.

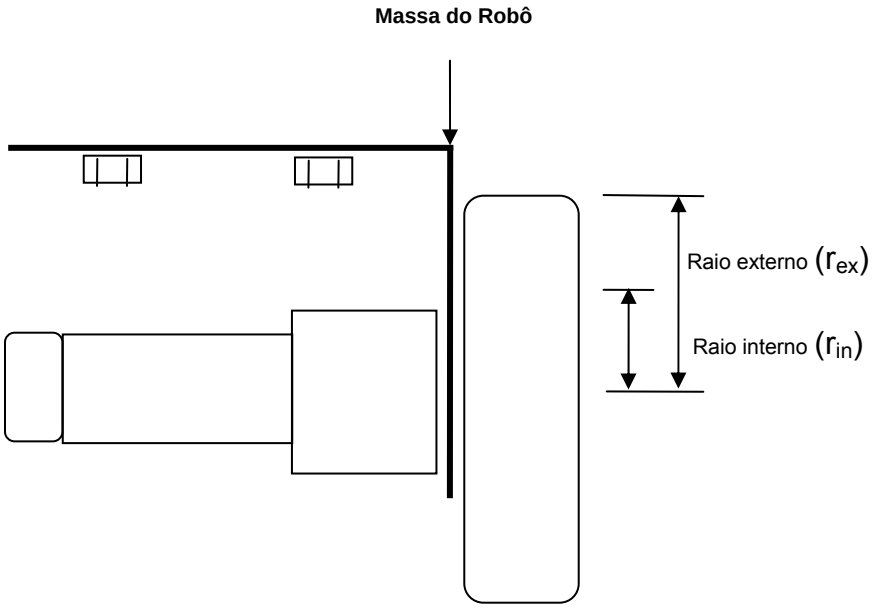

Figura 4.3.4 Conjunto, motor, caixa de redução e roda do robô.

A inércia do cubo da roda e da roda pode ser calculada da seguinte fórmula padrão:

$$J_{cubo} = \frac{M_{cubo} r_{in}^2}{2} \tag{4.3.11}$$

$$J_{roda} = \frac{M_{roda}}{2}(r_{in}^2 + r_{out}^2) \tag{4.3.12}$$

A massa do robô exerce uma força igual à metade do peso do robô, ao qual é sentida pelo raio da roda. Entretanto a inércia devido à massa do robô é:

$$J_{robô} = \frac{M_{robô}}{2} r_{externo}^2 \tag{4.3.13}$$

A inércia da carga é dada por:

$$J_L = J_{cubo} + J_{roda} + J_{robô} \tag{4.3.14}$$

4.3.2.2. Determinação da Função de Transferência

Utilizando a Transformada de Laplace, a equação 4.3.9 pode ser escrita como:

$$s\omega_{m(s)} = -K_2\omega_{m(s)} + K_3 e_m \tag{4.3.15}$$

A Função de Transferência da tensão de entrada $e_{m(s)}$, para a velocidade angular de saída é:

$$\frac{\omega_{m(s)}}{e_{m(s)}} = \frac{K_3}{s + K_2} \tag{4.3.16}$$

Na Tabela 4.3.1 são mostrados os valores dos parâmetros do motor dc, caixa de redução e componentes do robô. O motor escolhido é do tipo rotor bobinado com escovas e de imã permanente.

Tabela 4.3.1 Parâmetros do Motor DC, caixa de redução e componentes do robô.

Parâmetro	Nome	Valor	Unidade
R	Resistência do motor	2,2500	Ω
K_g	Razão de redução	127,7800	-
K_i	Constante de torque do motor	0,0021	Kgf*m/A
K_s	Constante de velocidade $K_s = e_m(máx)/ \omega_m(máx)$	0,0220	rad/s
J_L	Inércia da carga conectada ao eixo $J_c + J_{rd} + J_{robô}$	0,0190	$Kg*m^2$
J_m	Inércia do motor	2,1800E-06	$Kg*m^2$
K1	Constante 1	2,1845	-
K2	Constante 2	6,1410	-
K3	Constante 3	279,1349	-

Substituindo K2 e K3 pelos seus respectivos valores na Função de Transferência, temos:

$$\frac{\omega_{m(s)}}{e_{m(s)}} = \frac{279{,}13}{s + 6{,}14} = \frac{45{,}46}{\frac{1}{6{,}14}s + 1} = \frac{K}{\tau s + 1}$$

Com o objetivo de se obter um valor de constante de tempo (Tau) e de ganho (K) que mais se aproxime da dinâmica do robô, foi implementado uma rotina em linguagem C no microcontrolador para:

- Gerar um degrau de velocidade, ou seja, um degrau de tensão aplicado ao enrolamento do rotor do motor da roda esquerda e direita do robô;
- Medir a velocidade da roda esquerda e direita do robô, através do encoder,

Para registro dos resultados obtidos no tempo, foi utilizado o gráfico disponível pela ferramenta de visualização (Visualization Tool) do Simulador com Tempo Exato e Depurador em Tempo Real, do ambiente de programação CodeWarrior (Freescale).

A Figura 4.3.5 mostra o gráfico com os resultados obtidos desta implementação. As curvas de cor verde e azul mostram o comportamento da velocidade no tempo para as rodas direita e esquerda, respectivamente do robô. A curva de cor azul mostra o degrau de tensão aplicado no atuador, ou seja, motor da roda esquerda e direita do robô. A menor unidade de marcação da escala de tempo é de 0,375 segundos.

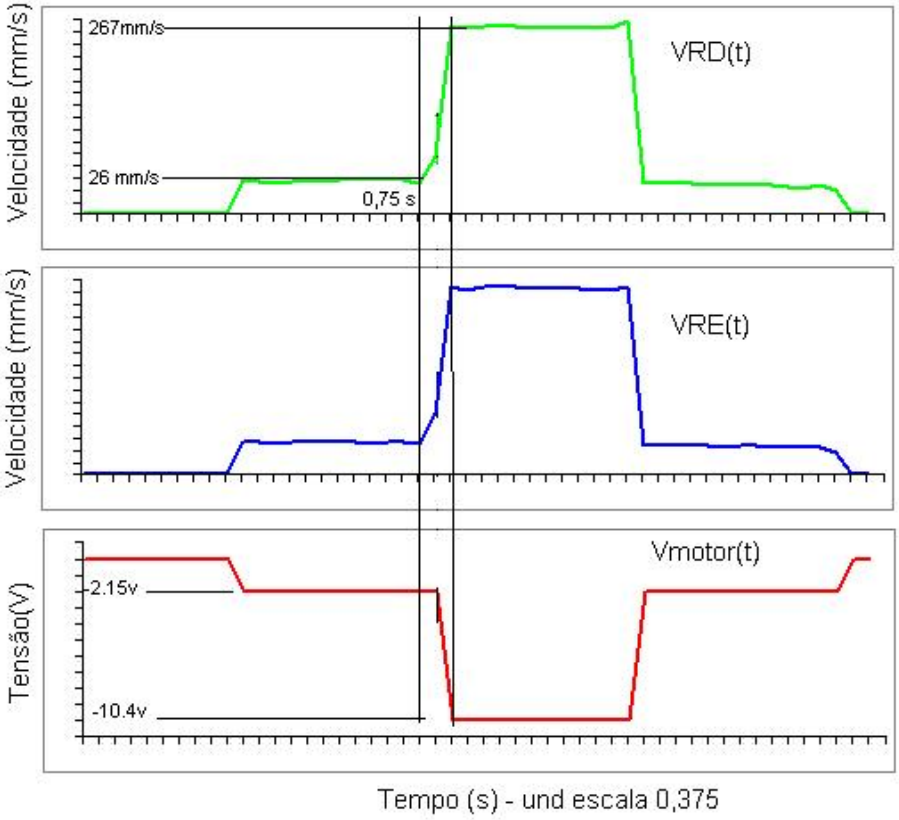

Figura 4.3.5 Gráfico com o teste para obter a dinâmica do robô.

Do gráfico obtém-se os seguintes valores:

$$K = \frac{\Delta\omega_m}{\Delta e_m} = \frac{\omega_{m2} - \omega_{m1}}{e_{m2} - e_{m1}} = \frac{267 - 26}{2{,}15 - 10{,}4} = -29{,}21$$

$$\tau = 0{,}63(t_2 - t_1) = 0{,}63(0{,}75) = 0{,}472s$$

$$\frac{\omega_{m(s)}}{e_{m(s)}} = \frac{K}{\tau s + 1} = \frac{29{,}21}{0{,}472s + 1}$$

Estes valores estão próximos dos valores calculados e serão utilizados para determinação da melhor sintonia do controlador.

4.3.2.3. Escolha do controlador e sintonia para o objetivo específico.

Os problemas de controle em malha fechada podem ser classificados em dois principais tipos de aplicações:

a) Regulatório – Onde o ponto de operação "setpoint" é fixo e se deseja manter o processo o mais próximo possível deste valor, apesar das perturbações. O objetivo deste controle é rejeitar os efeitos das perturbações.

b)Servo – Onde o ponto de operação deve seguir uma trajetória. O objetivo destes controles é seguir com o mínimo erro o "setpoint" desejado.

Na prática, os sistemas de controle devem ser capazes de resolver ambos os problemas na presença de incertezas e perturbações no processo.

O controlador do tipo Proporcional-integral-Derivativo é, sem dúvida, o mais usado em sistemas de malha fechada em diversas áreas. As vantagens deste controlador são:

- Bom desempenho em muitos processos;
- Estrutura versátil;
- Poucos parâmetros a serem sintonizados ou ajustados;
- Fácil associação entre os parâmetros de sintonia e o desempenho.

Antes de se obter a sintonia do controlador PID, para um processo com dinâmica conhecida, deve-se definir o critério de desempenho desejado para a malha. Os desempenhos desejados para o controle do robô são:

- Velocidade linear e angular que resulte em boa precisão no percurso da trajetória definida;
- A mudança de um ponto operacional para outro, mudança de “setpoint” de velocidade em degrau, deve ser a mais rápida possível e sem sobrevalor (isto é, sem ultrapassar o novo “setpoint”);
- Manter a aceleração e a desaceleração a mais rápida possível;
- Prevenir o robô da ocorrência de derrapagem ou escorregamento.

Para atender a estes critérios de desempenho são realizadas simulações no programa matlab conforme diagrama em blocos do sistema em malha fechada mostrado na Figura 4.3.6. São avaliados os controladores do tipo PI e PID e o método de sintonia utilizado é o ITAE (Integral do produto do Tempo pelo valor Absoluto do Erro) por ser o método de sintonia que produz a resposta mais rápida sem sobrevalor para uma entrada em degrau. A Tabela 4.3.2 mostra os valores das constantes e fórmulas para o cálculo da sintonia do controlador PI. A constante θ, tempo morto do sistema, não é aplicável ao sistema em estudo.

Tabela 4.3.2 Constantes e fórmulas para o cálculo da sintonia do controlador PI.

Controlador	Critério	A	B	C	D
PI	ITAE	0.859	-0.977	0.674	-0.68
$Kp = 1/K[A\,(\theta/Tau)^{B}]$		$Ti = Tau/[C(\theta/Tau)^{D}]$			

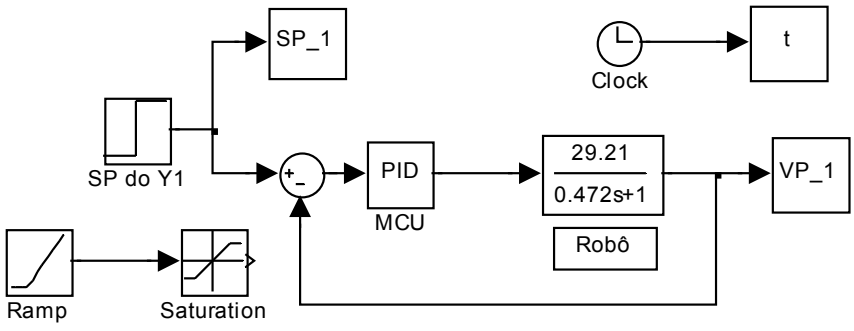

Figura 4.3.6 Sistema utilizado para simulação do controlador e F.T. do robô.

4.3.2.4. Análise dos resultados do Simulador

Os valores de sintonia obtidos pelo método ITAE se mostraram satisfatórios para uma entrada em degrau. Para uma entrada em rampa a saída não seguiu a entrada sem erro. Foi necessário aumentar o valor de Kp e de Ti para corrigir este erro sem afetar a resposta em degrau. Após as devidas correções, realizadas de forma empírica, chegou-se aos valores de ganho Kp = 0.1 e tempo integral Ti = 0.2. Estes valores foram utilizados no simulink para análise da resposta a uma entrada em degrau e rampa, apresentando resultados satisfatórios conforme Figuras 4.3.7 e 4.3.8.

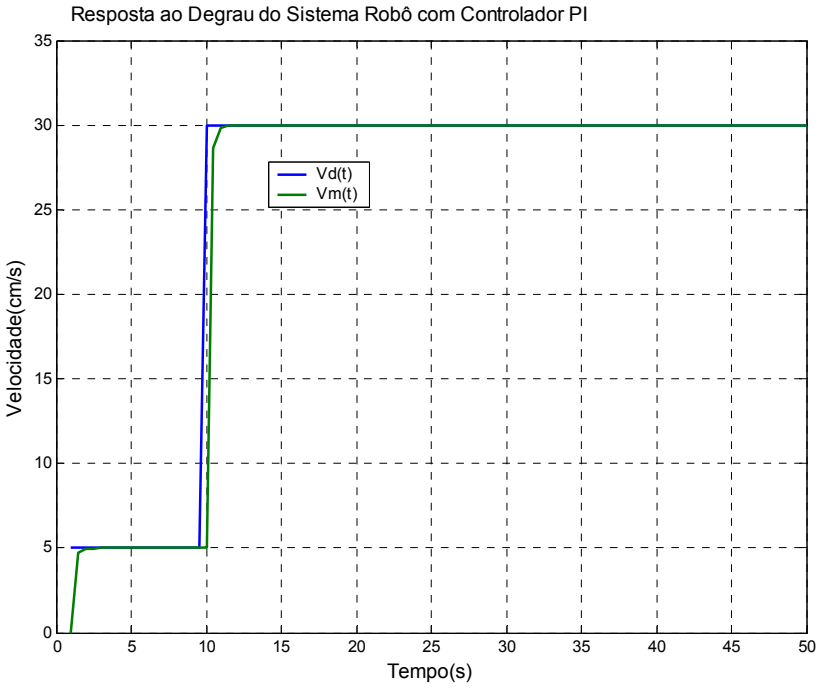

Figura 4.3.7 Resposta do sistema controlador e robô a uma entrada em degrau.

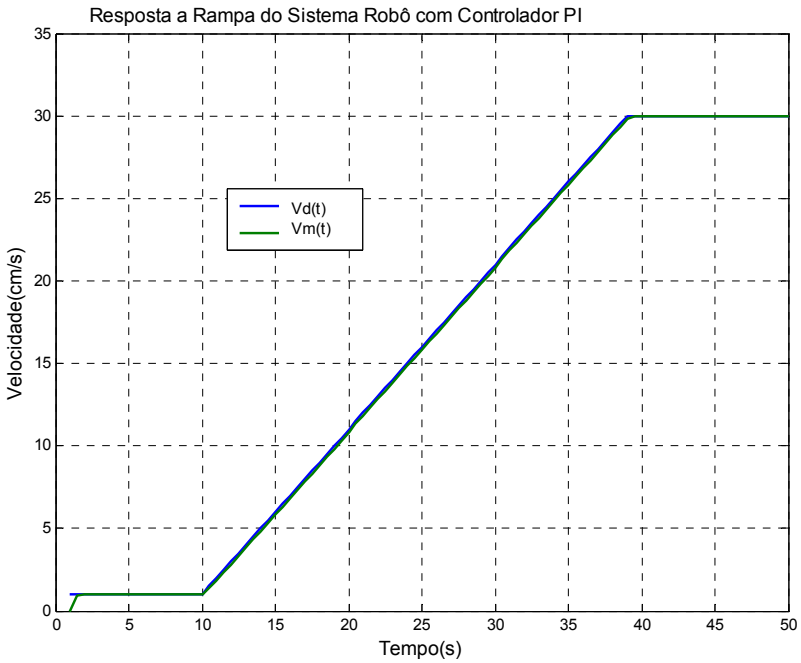

Figura 4.3.8 Resposta do sistema controlador e robô a uma entrada em rampa.

4.4. ALGORITMO DE CONTROLE

Nos sistemas de controle de malha fechada do tipo PID, embarcados em microcontroladores, o sinal analógico deve ser convertido para amostras digitais discretas antes de ser processado. A performance da malha determina a taxa de amostragem, e os cálculos devem ser terminados antes que o próximo tempo de amostra se inicie. Estas restrições relacionadas à malha e a exigência da freqüência de Nyquist colocam um limite superior nos sistemas fechados de controle digital com o erro de realimentação. Se o sistema controlado tem uma ressonância ou outro comportamento como uma constante de tempo mais curta do que o tempo de cálculo e amostra, resultados indesejáveis são obtidos. Apesar destas limitações, o aumento na taxa de clock do microcontrolador e a adição de hardware orientado ao controle, interno ao chip, estão expandindo o número de aplicações de controle de média performance em microcontroladores de 8 bits [17]. Os processadores da classe DSP (Processador de Sinal Digital) ou DSC (Controlador e Processador de Sinal Digital), é a escolha correta para a maioria das aplicações, mas o custo em relação à família de microcontroladores de 8 bits é bem superior.

4.4.1. O Algoritmo PID

O diagrama de blocos de um sistema controlado por um controlador PID paralelo alternativo é mostrado na Figura 4.4.1. Neste controlador o ganho proporcional K_P não afeta nem o termo integral, nem o termo derivativo. Ele é dito paralelo porque as suas ações são calculadas em paralelo e em seguida somadas.

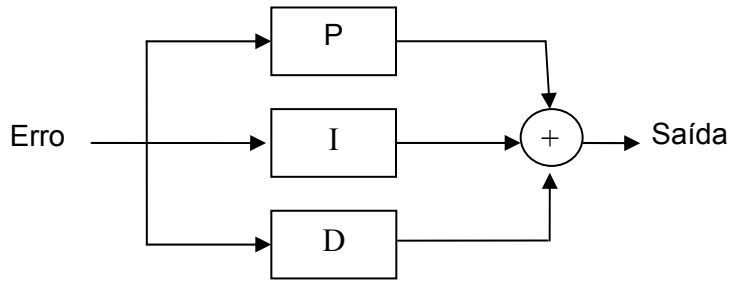

Figura 4.4.1 Diagrama de blocos do controlador PID paralelo alternativo.

Na Figura 4.4.2 é mostrada a realização em Z obtida da equação 4.4.2, do controlador PID no domínio do tempo.

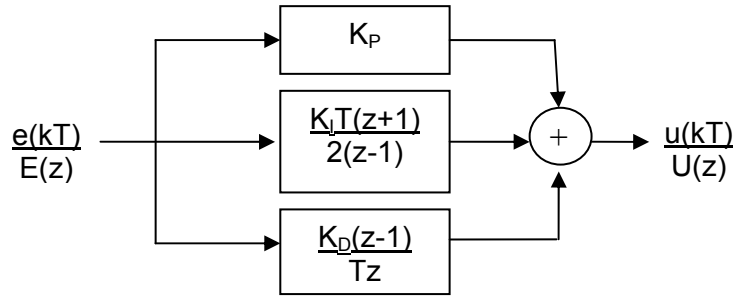

Figura 4.4.2 Realização em Z do controlador PID paralelo alternativo.

O sinal de erro e(t) é obtido da diferença entre o “setpoint” de velocidade desejado ω_D e a velocidade atual ω_A no tempo.

$$e(t) = \omega_D - \omega_A(t) \qquad 4.4.1$$

A correção da saída saida(t) para o controlador PID é:

$$saida(t) = K_P\, e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \perp t = T \qquad 4.4.2$$

Onde K_P, K_I e K_D são as constantes, ganho proporcional, ganho integral e ganho derivativo.

Reescrevendo a integral temos:

$$saida(t) = K_P\ e(t) + K_I \int_{t=0}^{t} (\omega_D - \omega_A)dt + K_D \frac{de(t)}{dt} \perp t = T \qquad 4.4.3$$

Para introduzir o tempo discreto, fazemos t = nT onde n = 1,2,3,...,n e T é o período de atualização do controle e amostragem. Ainda temos que t_O = (n-1)T. A integral avaliada de (n-1)T até nT pode ser aproximada utilizando a regra de integração trapezoidal. A derivada do erro é simplesmente a taxa de mudança do erro, mas esta pode ser ruidosa durante um único período.

A forma na qual pode ser executada diretamente pelo microcontrolador é:

$$saida(n) = K_P\ e(n) + T \frac{\sum e(n)}{T_i} + T_d \frac{[e(n) - e(n-1)]}{T} \qquad 4.4.4$$

Este termo é adicionado a saída corrente e colocada dentro do registrador de controle do PWM no início do próximo ciclo de cálculo. Substituindo as variáveis e constantes da Equação 4.4.4 temos:

```
errord = (AMORD - vrd*sinalrd);
somard += errord*ki;
saidard = somard + errord*kp + (errord - erro_antrd)*kd ;
erro_antrd = errord;
```

Onde:
rd – roda direita;
saidard – Valor a ser carregado no registrador de controle do PWM;
vrd – Velocidade desejada da roda direita;
AMORD – Velocidade atual amostrada da roda direita;
errord – Sinal de erro;
sinalrd – Sinal que define sentido de rotação horário ou anti-horário;
erro_antrd – Valor anterior carregado no registrador de controle do PWM;

A mesma equação é utilizada no controle de malha fechada da roda esquerda, sendo que as letras RD das variáveis são substituídas por RE. (ex: vre, errore,...)

A função do termo proporcional é clara, mas os termos derivativo e integral necessitam de uma breve explicação. Quando a variável regulada está fora do “setpoint” especificado, um controlador operando somente com controle proporcional, tenderá a aumentar a tensão de saída até que o sinal de erro seja zero. O sistema retorna ao “setpoint” com mais tensão de saída do que é exigido para manter o equilíbrio. Isto provoca um sobre-sinal na variável regulada e em seguida tende a oscilar em valor abaixo do “setpoint”. O termo derivativo contribui proporcionalmente a taxa de variação do erro, mas com sinal oposto ao termo proporcional. O objetivo do termo integral é eliminar o erro em regime estacionário.

4.4.2. Descrição da tarefa implementada em linguagem C

A tarefa implementada, controle da velocidade em malha fechada da roda direita e da roda esquerda, constitui as terceiras e quartas tarefas do sistema de controle Fundo(*nível tarefa)*/Frente(*nível de interrupção*).

O programa principal inicializa as constantes e variáveis, os registros do temporizador 2 (TPM2), a porta D, os registros do Clock do Sistema (ICG) e a rotina do protocolo de comunicação é executada. Quando esta rotina detecta a recepção do caracter “@”, significa que a matriz de seqüência de trinas numéricas, SEQUENCIA[ ], está pronta para ser executada. As trincas numéricas desta matriz são passadas uma a uma para serem executadas pela função trecho.

A função trecho (unsigned int tempo, signed int vrd, signed int vre) é a que realiza as principais subtarefas. Esta função realiza os cálculos do PID, com base no valor da amostra da velocidade, (AMORD e AMORE), e no SP desejado de velocidade (vrd e vre). Em seguida é verificado se o sinal de saída está dentro dos limites estabelecidos e após esta verificação é feito o ajuste de escala através da equação de conversão (4.2.3). O valor de saída resultante é armazenado no registrador de controle do PWM.

A função trecho é executada em um período T, inferior a 21 ms (47,6 Hz). Este tempo é estimado com base na quantidade de instruções necessárias para executar o algoritmo PID e atualizar o registrador do PWM. É possível medir este tempo através do osciloscópio, ativando uma saída do MCU no início da função e desativando no término da mesma. Conforme teorema de amostragem de Nyquist, a taxa de execução deve ser superior a duas vezes a taxa de amostra da velocidade. A taxa de amostra da velocidade é realizada em 42 ms (23,8 Hz).

As Figuras 4.4.3, 4.4.4, 4.4.5 e 4.4.6 mostram o fluxograma do controle de malha fechada para as rodas direita e esquerda.

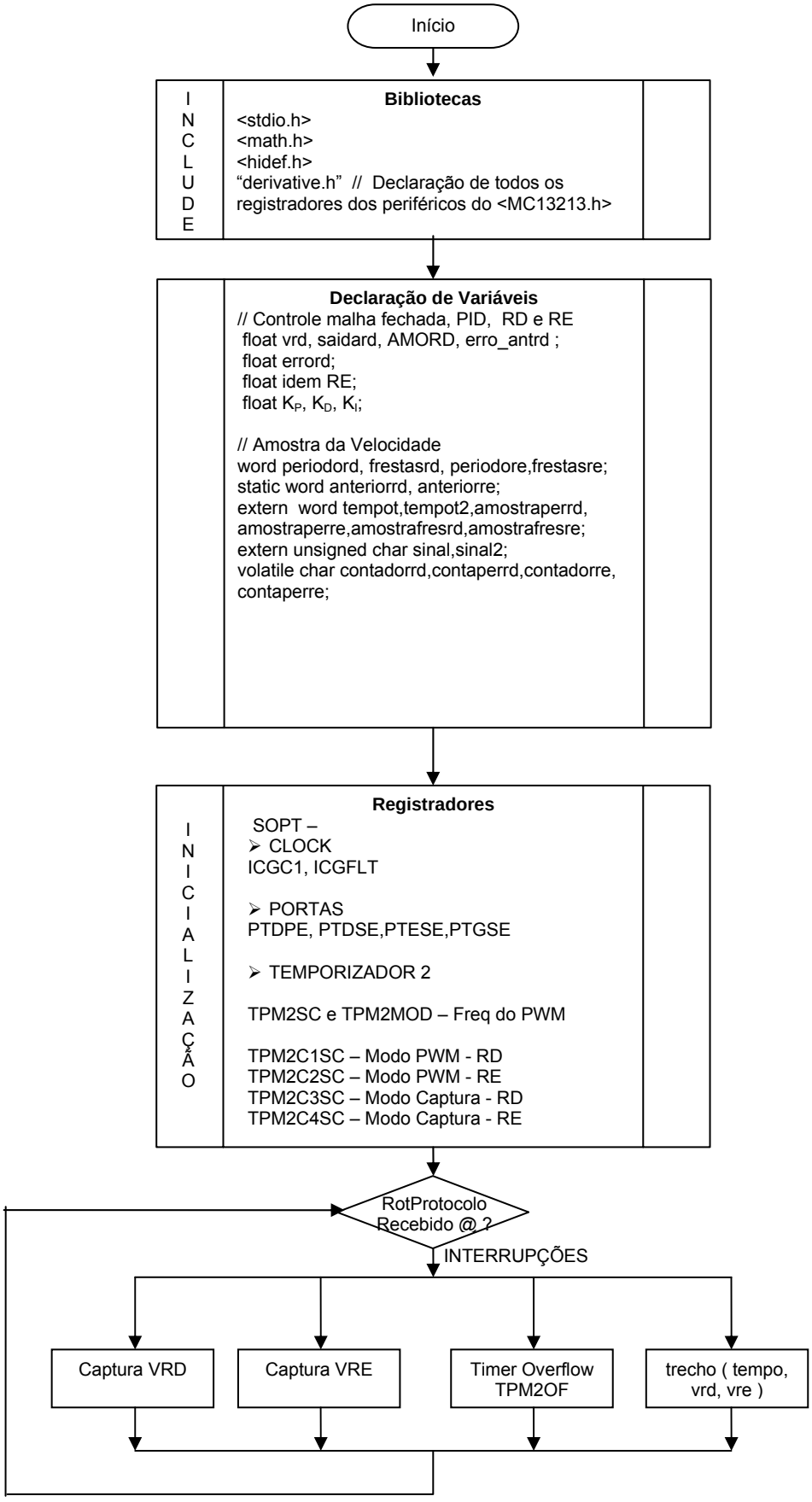

Figura 4.4.3 Visão geral do fluxograma de controle de malha fechada da RD e RE.

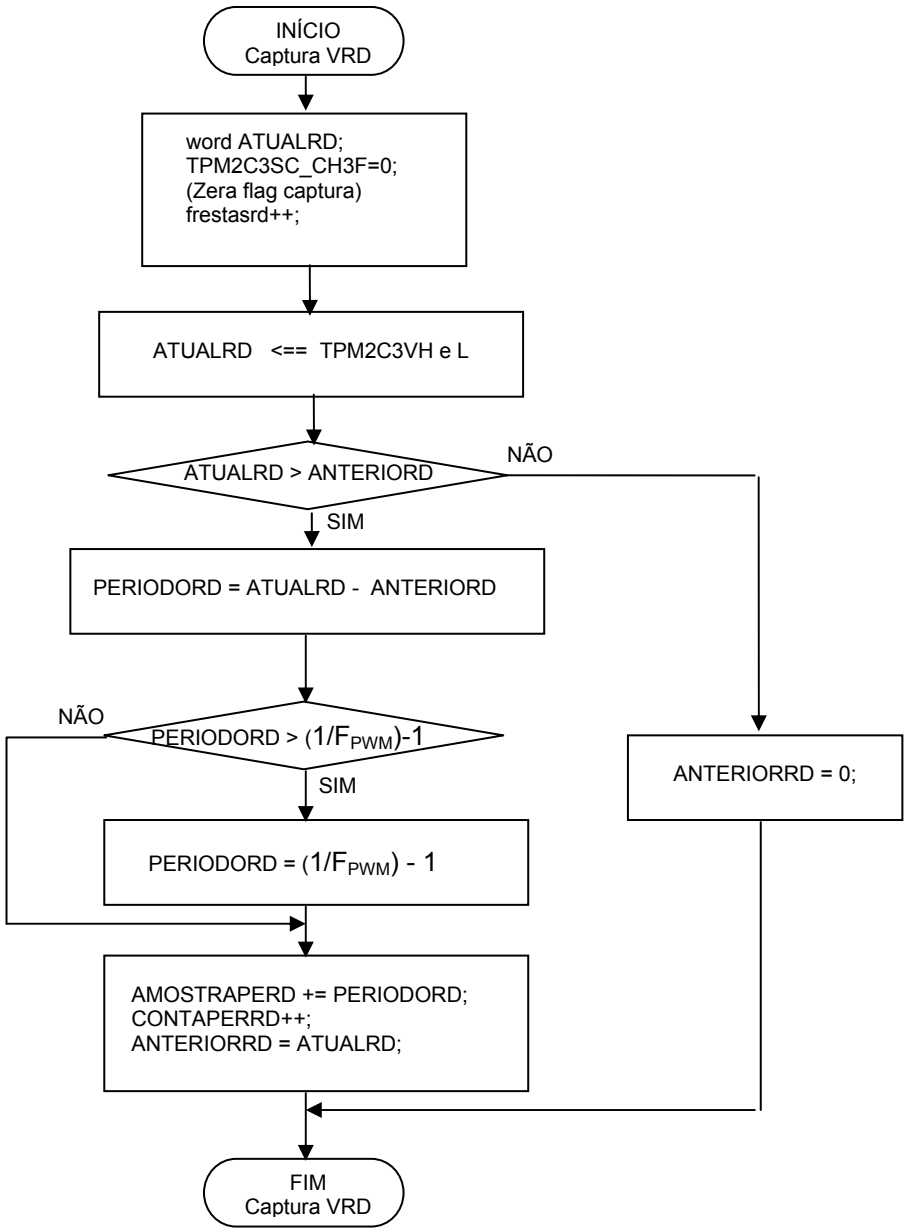

Figura 4.4.4 Fluxograma da rotina de interrupção da captura da velocidade da RD.

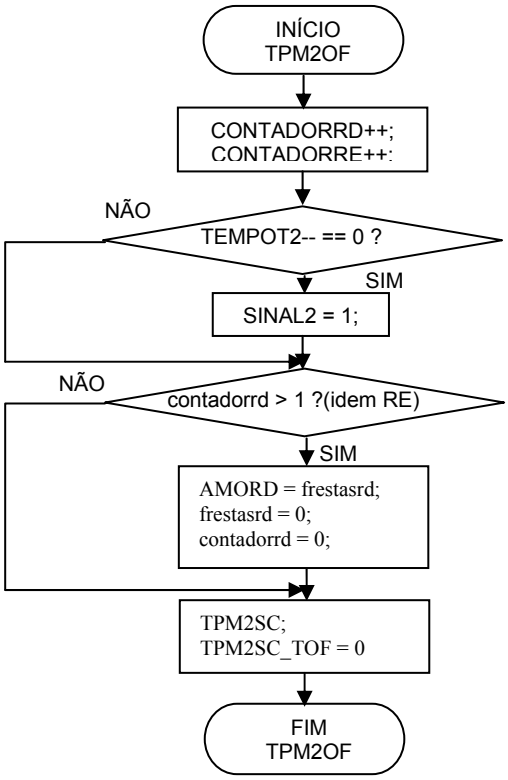

Figura 4.4.5 Fluxograma da rotina de interrupção Timer Overflow (TPM2OF).

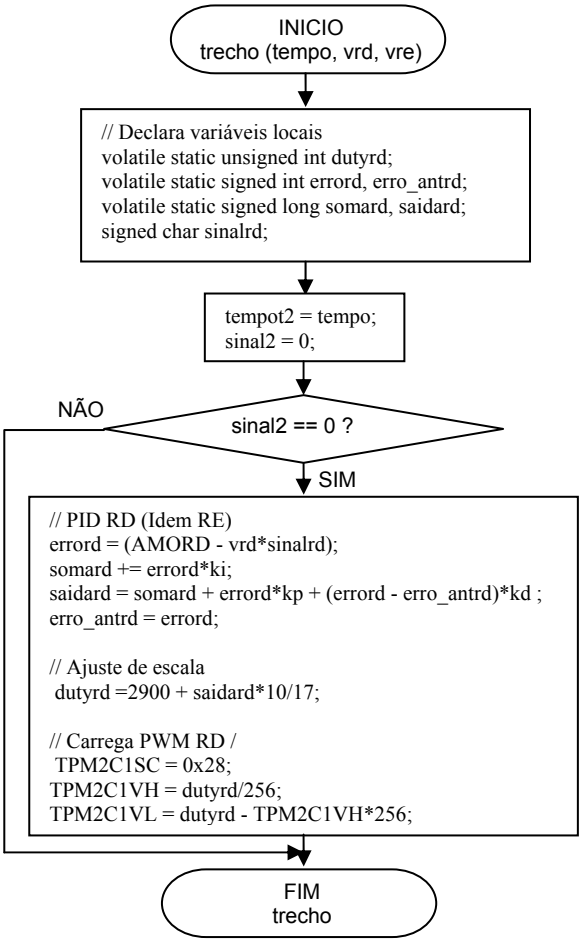

Figura 4.4.6 Fluxograma da função trecho (int tempo, sign int vrd, sign int vre).

5. SISTEMA DE NAVEGAÇÃO

O planejamento da trajetória é executado em um PC, em aplicativo com interface gráfica de usuário, criada através do Matlab v7.1. Apesar do Matlab ser uma linguagem interpretada ao invés de compilada, resultando em um programa de execução lenta, é possível utilizar o comando *mcc* para ser compilado por um compilador C++ comercial. Este recurso permite tornar o programa criado no ambiente original Matlab, em um arquivo executável que executa mais rápido do que o código original interpretado. Um aplicativo com possibilidade de processar imagens digitalizadas em formato bitmap, desenvolvido na linguagem de programação Java, é apresentado por CRISTINA [18].

A Figura 5.1 mostra em detalhes a interface gráfica desenvolvida em linguagem de programação Matlab, chamado de Smartway. A interface gráfica é constituída da grade de ocupação, com células de dimensão fixa de 100 x 100 mm, sendo que é possível escolher uma dentre três opções de áreas de mapa de avaliação:

a) 7,68m^2 – 2,4 m x 3,2m - Total de 768 células de 100 x 100 mm.

b) 30,72 m^2 - 4,8m x 6,4m – Total de 3072 células de 100 x 100 mm.

c) 122,88 m^2 - 9,6m x 12,8m - Total de 12288 células de 100 x 100 mm.

A grade de ocupação é uma representação do ambiente (mapa) por decomposição em células. Cada célula representa uma região quadrada do ambiente que, no programa desenvolvido, é manualmente designada como, obstáculo (ou parte de um grande obstáculo) , ou coordenada de origem ou coordenada de destino. As células que não foram designadas com uma das opções anteriores são células livres ou não ocupadas, que poderão ser utilizadas ou não no percurso a ser executado pelo robô.

A interface gráfica também é constituída de botões e caixas de texto para permitir criar ou apagar obstáculos e definir as coordenadas de origem e de destino. Depois de definida a configuração da grade, o algoritmo de busca do melhor caminho é iniciado ao pressionar o botão *Iniciar*.

A Figura 5.1 mostra o resultado do planejamento da trajetória utilizando-se um mapa criado manualmente. As células não ocupadas são as em branco. Os obstáculos são representados por células com quadrado interno. A célula com letra I indica a posição inicial do robô e a célula com letra *F* indica a posição final a ser alcançada. O caminho encontrado pelo algoritmo A Estrela pode ser observado pelas células na cor laranja.

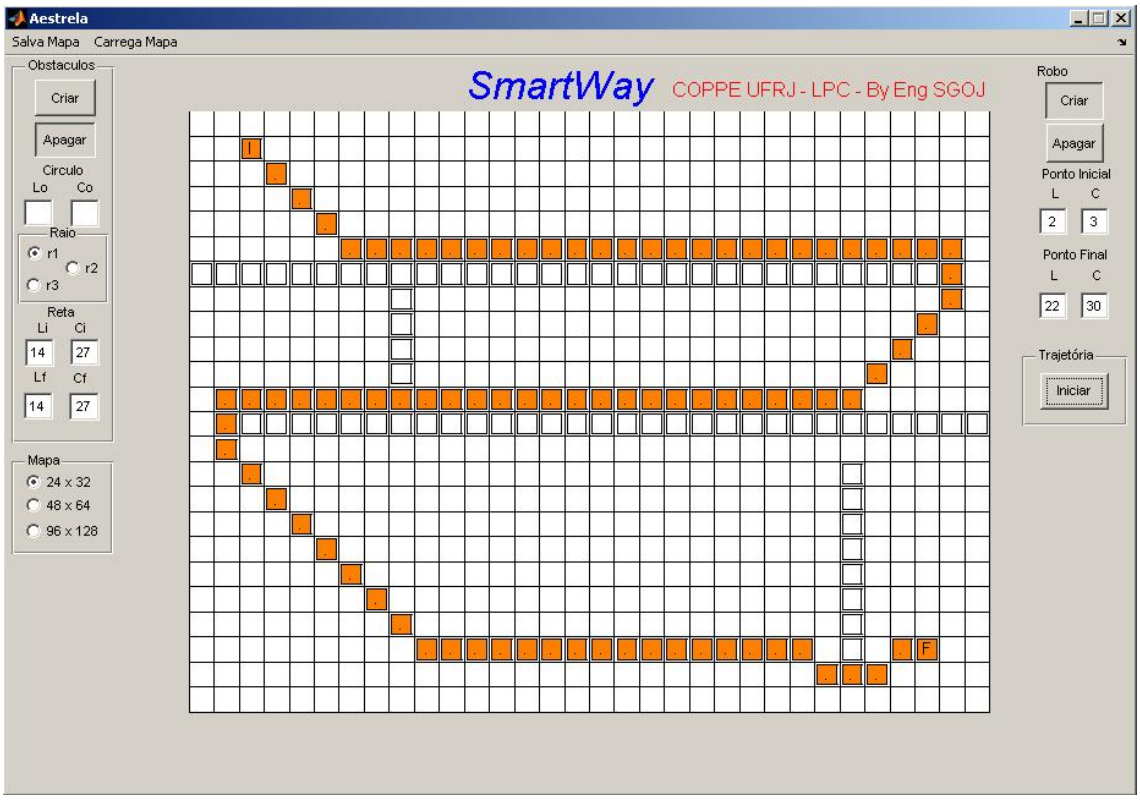

Figura 5.1 Interface gráfica Smartway com aplicação em funcionamento.

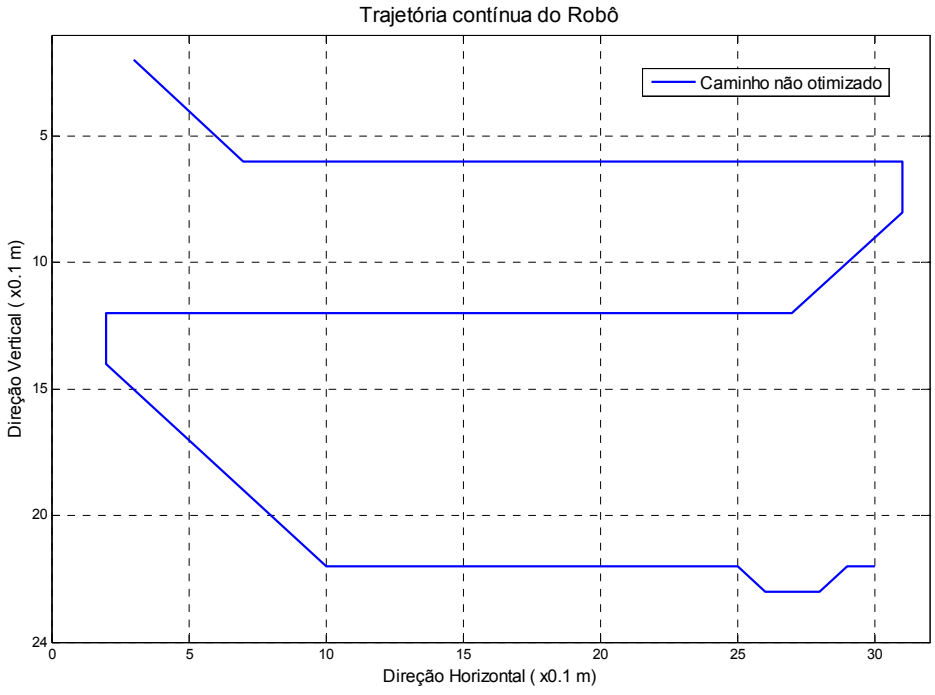

Figura 5.2 Caminho contínuo não otimizado, gerado pelo SmartWay.

A Figura 5.2 mostra o caminho contínuo não otimizado gerado pelo SmartWay. Neste exemplo, o caminho é formado por 11 coordenadas além das coordenadas inicial e final. Na Tabela 5.1 são mostradas as coordenadas geradas pelo SmartWay e na Tabela 5.2 são mostrados as distâncias e os ângulos correspondentes.

Tabela 5.1 Coordenadas geradas pelo SmartWay

Núm Coord	X	Y
1	2	3
2	6	7
3	6	31
4	8	31
5	12	27
6	12	2
7	14	2
8	22	10
9	22	25
10	23	26
11	23	28
12	22	29
13	22	30

Tabela 5.2 Distâncias e ângulos correspondentes às coordenadas.

Trechos	Distância (m)	θ (Graus)
1-2	0,5657	-135
2-3	2,4000	90
3-4	0,2000	135
4-5	0,5657	135
5-6	2,5000	-90
6-7	0,2000	-135
7-8	1,1314	-135
8-9	1,5000	135
9-10	0,1414	-135
10-11	0,2000	-135
11-12	0,1414	135
12-13	0,1000	0

Os ângulos com sinal negativo indicam que o robô deverá girar sobre o seu próprio eixo no sentido anti-horário. Obviamente que esta não é uma trajetória ideal, pois o robô não executará um trajeto suave e contínuo. O robô deverá a partir da coordenada de origem, girar sobre o seu próprio eixo no sentido que o direcione para a próxima coordenada, caso já não esteja direcionado para tal. Depois deverá acelerar até a máxima velocidade e ficar durante um certo tempo, desde que a distância do trecho seja grande o suficiente, e desacelerar até parar na próxima coordenada. A partir daí o processo se repete até a coordenada final.

O planejamento ideal de trajetória deverá incluir ainda **dois algoritmos de otimização** e tomar como base as restrições cinemáticas do robô. Estas restrições são a velocidade mínima e máxima e a aceleração ideal. O **primeiro algoritmo** de otimização deverá transformar as interseções das retas em arco de círculo de raio compatível com a curva e a existência de obstáculo. Para as curvas de raios menores

a velocidade deverá também ser menor e constante para se evitar imprecisões na trajetória. O **segundo algoritmo** deverá transformar os trechos retos em dois ou três trechos dependendo da sua distância. Os trechos menores incluirão um trecho de aceleração constante (mudança de SP de velocidade em pequenos degraus de tempo constante) a partir do SP de velocidade de saída do trecho anterior e um trecho de desaceleração constante até atingir o SP de velocidade previsto para o próximo trecho. Os trechos maiores incluirão entre os dois trechos de aceleração e desaceleração anteriores o SP de velocidade máxima e o tempo de permanência nesta.

5.1. ALGORITMO A ESTRELA

Existem várias técnicas de inteligência artificial (IA) disponíveis para selecionar o melhor caminho em um grafo de vários caminhos possíveis. Um exemplo é o algoritmo de busca branch and bound (Winston, 1984).

O algoritmo de busca A Estrela, é um refinamento do algoritmo branch and bound. Nesse algoritmo, o espaço de busca é reduzido excluindo-se múltiplos caminhos para uma determinada célula e deixando somente o caminho de menor custo. A escolha do caminho de menor custo para expansão em cada estágio é melhorada adicionando-se ao custo atual um custo estimado para os caminhos restantes. Se o custo estimado for sempre menor que o custo atual, a busca A Estrela produzirá uma solução ótima. Uma função típica utilizada na estimativa desse custo no planejamento de trajetória é a distância de uma linha reta.

A Figura 5.1.1 mostra o fluxograma do algoritmo A Estrela para determinação da trajetória do robô, em área com obstáculos.

Descrição do Algoritmo:

1) Adicionar QI à lista fechada.

2) Fazer continuamente:

a. Procurar o quadrado que tenha o menor custo de F na lista aberta, com referência ao quadrado corrente.

b. Movê-lo para a lista fechada.

c. Para cada um dos 8 quadrados adjacentes a este quadrado corrente, verificar:

Se não é passável ou se estiver na lista fechada, ignorar. Caso contrário:

- Se não estiver na lista aberta, acrescente-o à lista aberta. Fazer o quadrado atual o pai deste quadrado. Gravar os custos F, G, e H do quadrado.

- Se já estiver na lista aberta, conferir para ver se este caminho para aquele quadrado é melhor, usando o custo G como medida. Um valor G mais baixo mostra que este é um caminho melhor. Nesse caso, mudar o pai do quadrado para o quadrado atual, e recalcular os valores de G e F do quadrado. Reordenar a lista por F para corresponder à mudança.

d. Parar quando:

- For acrescentado o quadrado alvo à lista fechada, o que determina que o caminho foi achado, ou
- Não encontrar o quadrado alvo e a lista aberta está vazia. Neste caso, não há nenhum caminho.

3) Salvar o caminho. Caminhando para trás do quadrado alvo, ir de cada quadrado a seu quadrado pai até alcançar o quadrado inicial. Este é o caminho.

Definições:

TM – Tamanho do Mapa (Número de Células);
QI – Quadrado inicial;
QF – Quadrado final;
LF – Lista fechada;
LA – Lista aberta;
QA's – Quadrados adjacentes (total de oito) ao quadrado pai;
F = G + H – Equação que define o custo do quadrado em avaliação;
G - Custo para se mover do QI até o quadrado em avaliação;
H - Custo para se mover do quadrado em avaliação até o QF.

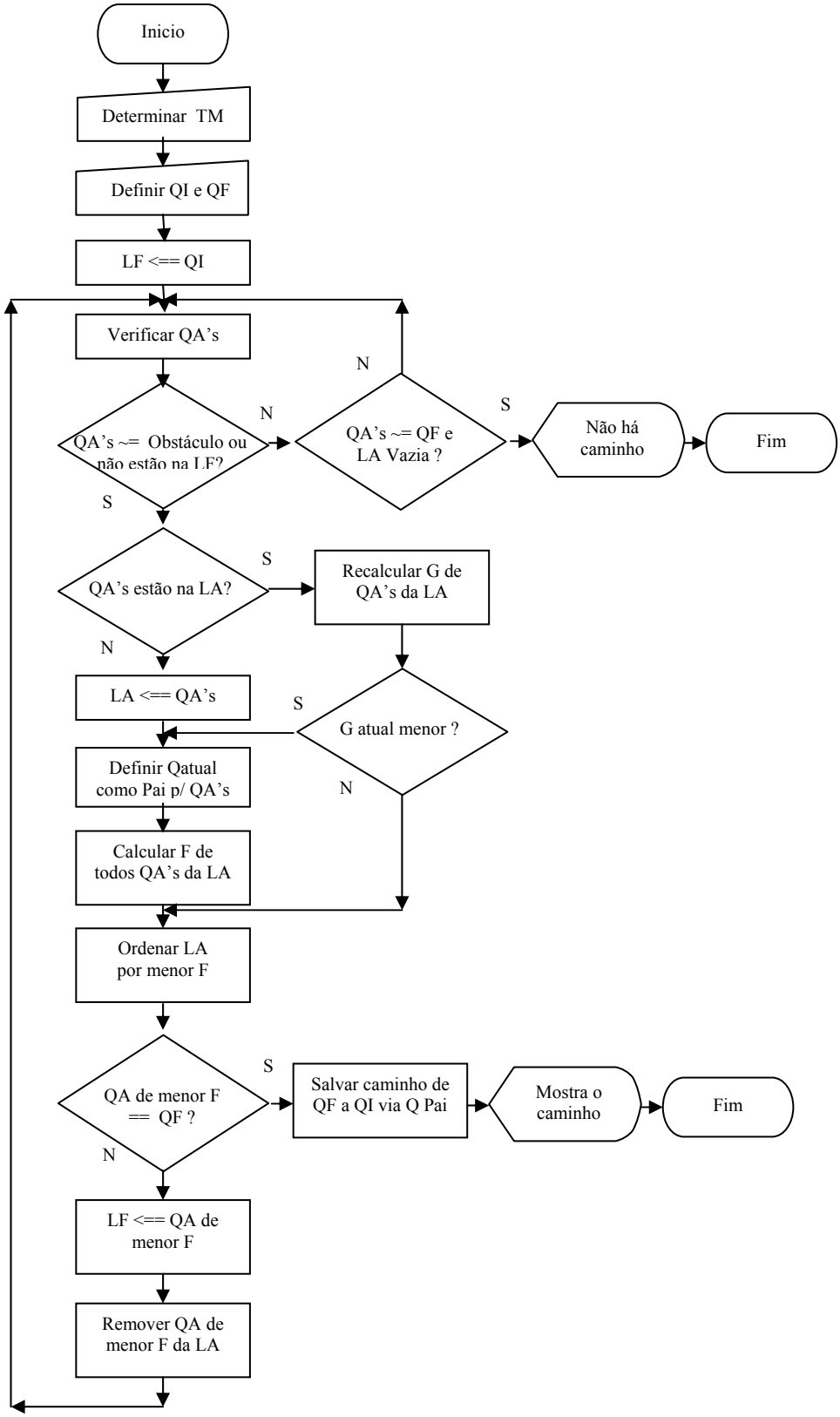

Figura 5.1.1 Fluxograma do Algoritmo A Estrela

6. RESULTADOS

6.1. SINTONIA DOS CONTROLADORES DA RODA DIREITA E RODA ESQUERDA

Com o objetivo de verificar e testar a sintonia final dos controladores da roda direita e esquerda, construiu-se um ambiente de testes no laboratório de Laboratório de Projetos de Circuitos da COPPE-UFRJ (LPC), Figura 6.1.1.

Figura 6.1.1 Ambiente de testes para verificação da sintonia do robô.

Foi implementada uma rotina em linguagem C no microcontrolador para:

- Gerar um degrau de velocidade a partir do valor mínimo, SP_{min} (10% do valor máximo de velocidade, 300 mm/s), até o valor máximo de velocidade, $SP_{máx}$ e após tempo de estabilização, retornar para o valor mínimo de velocidade;
- Gerar uma rampa de subida de velocidade a partir de 0 mm/s, SP_o, até o valor máximo, $SP_{máx}$ e após tempo de estabilização, retornar em rampa de descida para o valor 0 mm/s;
- Medir a velocidade da roda esquerda e direita do robô, através do encoder, para os dois casos anteriores;
- Medir o comportamento da variável manipulada, variação do duty cycle do PWM das rodas direita e esquerda, na entrada do módulo driver MD22, somente para o primeiro caso anterior. O módulo driver irá produzir uma tensão contínua

correspondente para o enrolamento do rotor do motor da roda esquerda e direita do robô.

Para o registro dos resultados obtidos no tempo, foi utilizado o gráfico disponível pela ferramenta de visualização (Visualization Tool) do Simulador com Tempo Exato e Depurador em Tempo Real, do ambiente de programação CodeWarrior (Freescale).

A Figura 6.1.2 mostra o gráfico com os resultados obtidos para o controlador PI da roda direita. A curva em vermelho mostra o setpoint de velocidade desejada, SPRD(t). As curvas de cor verde e azul mostram o comportamento da velocidade no tempo, VRD(t) e a saída do controlador, PWMRD(t), sinal do registrador do PWM (variável manipulada), respectivamente.

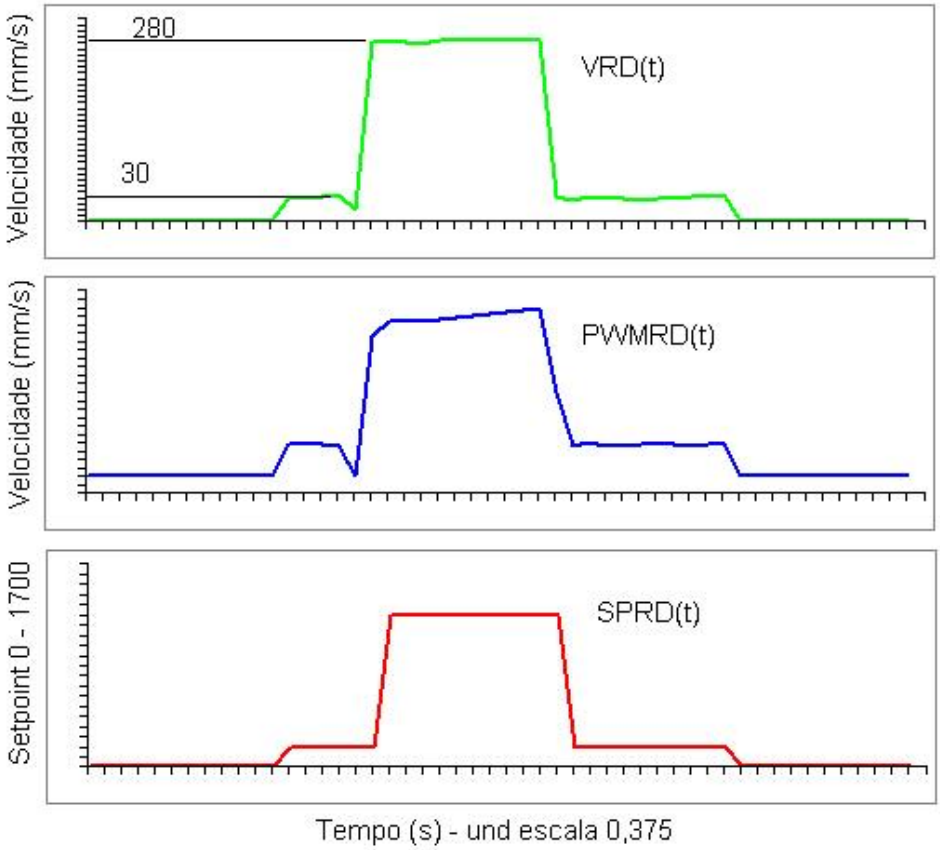

Figura 6.1.2 Desempenho do controlador PI da roda direita, entrada em degrau.

A Figura 6.1.3 mostra o gráfico com os resultados obtidos para o controlador PI da roda esquerda.

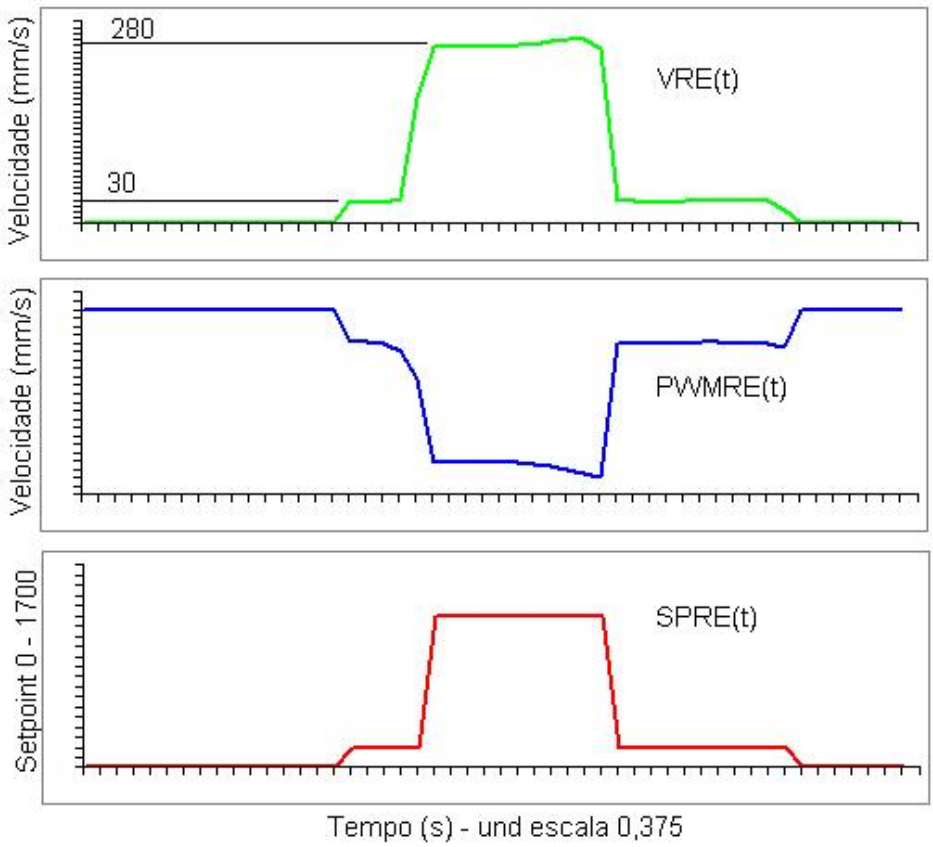

Figura 6.1.3 Desempenho do controlador PI da roda esquerda, entrada em degrau.

A Figura 6.1.4 mostra o gráfico com os resultados obtidos para o controlador PI da roda direita e esquerda, com entrada em rampa de subida. Esta rampa inicia com velocidade nula e é incrementada em passos de 16,1 mm/s durante 0.75 s, até a velocidade de 271 mm/s. A aceleração é constante e igual a 21,46 mm/s^2 .

A curva em vermelho mostra o setpoint de velocidade desejada, SPRD(t) e SPRE(t). As curvas de cor verde e azul mostram o comportamento da velocidade no tempo, para a roda direita, VRD(t) e esquerda, VRE(t), respectivamente.

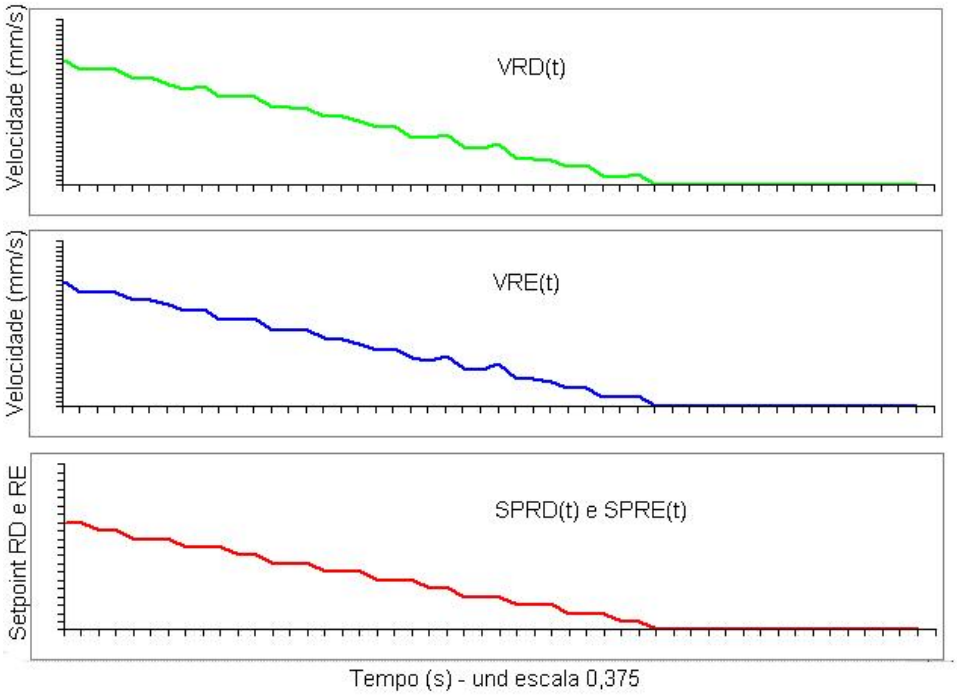

Figura 6.1.4 Desempenho do controlador PI da roda direita e esquerda, entrada em rampa de subida.

A Figura 6.1.5 mostra o gráfico com os resultados obtidos para o controlador PI da roda esquerda e direita, com entrada em rampa de descida.

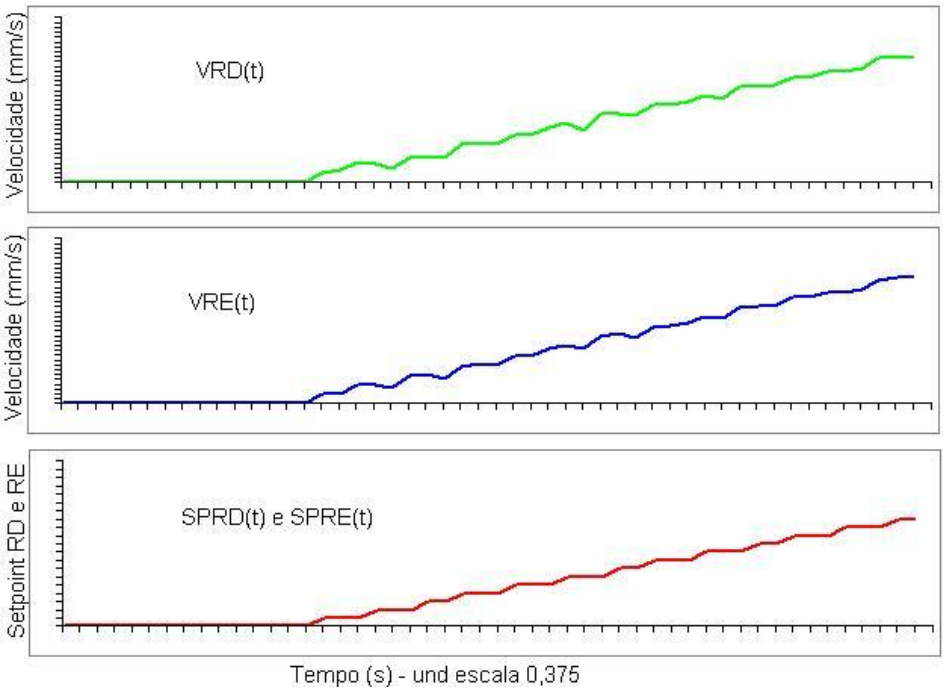

Figura 6.1.5 Desempenho do controlador PI da roda esquerda e esquerda, entrada em rampa de descida.

A menor unidade de marcação da escala de tempo é de 0,375 segundos. Primeiramente foram utilizados os valores de Kp e Ki obtidos pela simulação, conforme descrito em 4.3.2.4. Estes valores provocaram erros de máximo sobresinal acentuado, por isto foram alterados para Kp = 0.01 e Ki = 0.1.

A Figura 6.1.6 mostram os sinais através de osciloscópio, obtidos pelo encoder da roda esquerda e da roda direita.

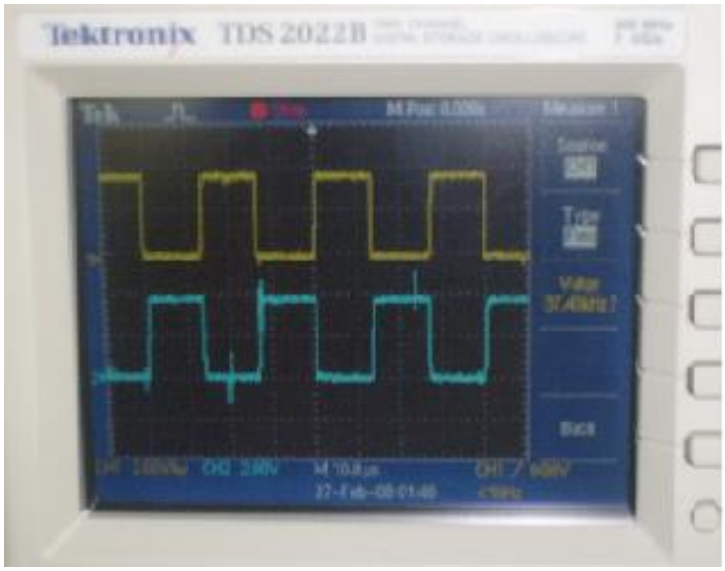

Figura 6.1.6 Sinais do encoder da roda esquerda e direita.

Para determinação da equação de normalização que relaciona os valores a serem carregados no registrador do PWM, conforme definido em 4.2.2, e a velocidade em mm/s, foram feitas medidas com o osciloscópio da freqüência do encoder correspondente a alguns destes valores definidos entre 0 e 1700. A equação 6.1.1 foi utilizada para determinação da velocidade linear.

$$\text{Velocidade} = \left(\frac{\text{Freq Encoder}}{\text{PPR Encoder}}\right)\left(\frac{\text{2pi Raio}}{\text{Razão Redução}}\right) \qquad (6.1.1)$$

Onde:

PPREncoder – Resolução do Encoder 500 PPR;

Freq Encoder – Freqüência do encoder monitorada pelo osciloscópio;

Razão Redução – 127.78;

2piRaio – Distância percorrida para um giro (471,24 mm).

A Tabela 6.1.1 mostra os valores obtidos para determinação da equação de conversão da velocidade linear do robô para o valor a ser carregado no registrador TPM2CXV, responsável por limitar o temo em nível high do sinal PWM. A Figura 6.1.7 mostra o gráfico gerado a partir destes valores.

Tabela 6.1.1 Valores aplicados ao PWM para determinação da equação de conversão da velocidade linear do robô.

Eq Norm2(mm/s)	0	33,81	70,79	126,26	218,71	237,20	274,18	283,42	311,16
Eq Norm1(mm/s)	0	34,35	70,75	125,35	216,35	234,55	270,95	280,05	307,35
Vel Encod (mm/s)	0	34,67	70,81	125,39	213,90	231,60	267,37	280,28	311,26
FreqEncOsc(Hz)	0	4700	9600	17000	29000	31400	36250	38000	42200
Valor	0	200	400	700	1200	1300	1500	1550	1700

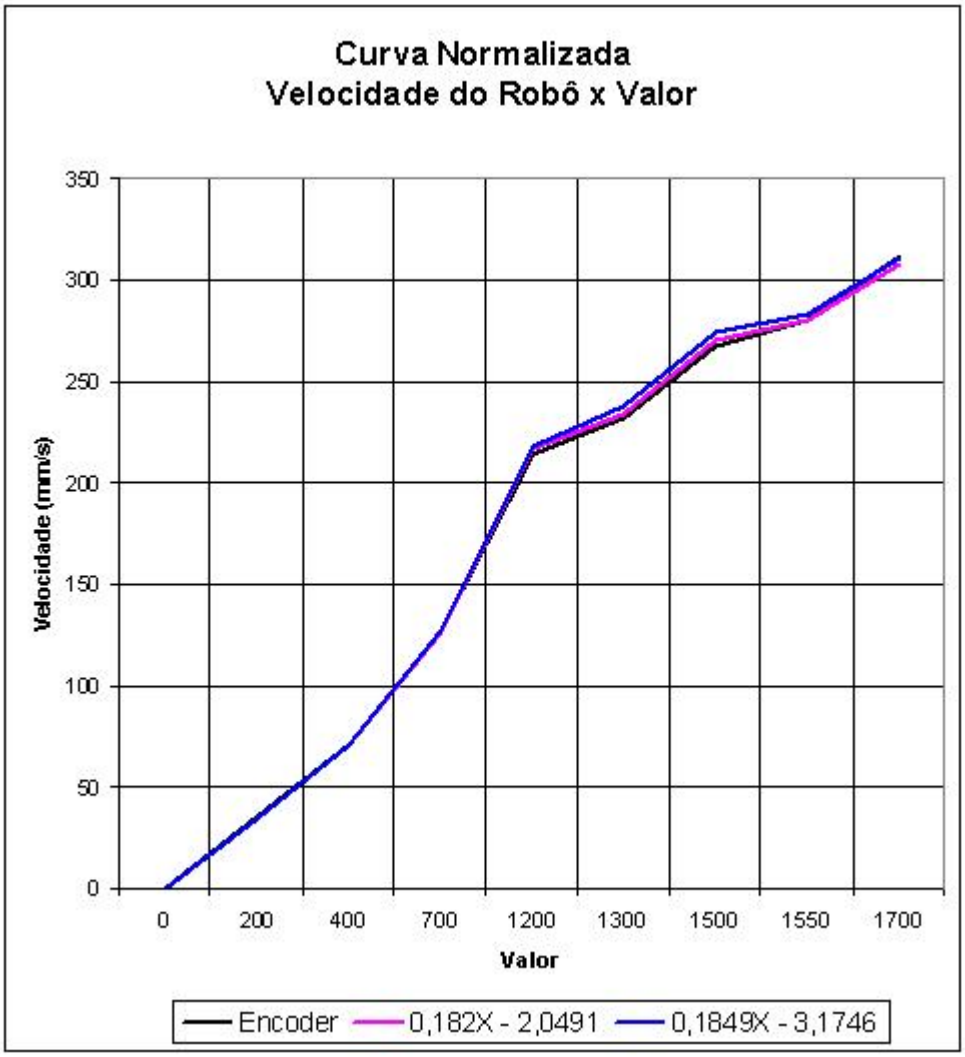

Figura 6.1.7 Gráfico com os resultados da equação de conversão.

A equação 6.1.2 é utilizada para a conversão da velocidade linear do robô em mm/s para o valor a ser carregado no registrador do PWM.

$$VRD = \frac{Vrd(mm/s) + 2{,}049}{0{,}182} \qquad (6.1.2)$$

6.2. VERIFICAÇÃO E ANÁLISE DO CAMINHO PERCORRIDO NO AMBIENTE PELO ROBÔ

Com o objetivo de se verificar a precisão e repetibilidade dos acionamentos do robô e conseqüentemente do caminho estabelecido, foram feitos testes em pequenos trechos de retas e de curvas. Através da interface gráfica da aplicação de planejamento da trajetória, foram inseridos as coordenadas de origem e destino, e então após processamento do algoritmo A estrela, foi obtida a seqüência de trincas numéricas. O robô foi posicionado nas coordenadas iniciais fornecidas pela aplicação e foram enviadas, via sinal de rádio freqüência, a seqüência de trincas numéricas. Os resultados obtidos para trechos retos com velocidade constante (aceleração nula), trecho reto com subtrecho de aceleração constante, velocidade constante e desaceleração constante e trecho com curvas, estão mostrados nas Tabela 6.2.1, 6.2.2 e 6.2.3.

Tabela 6.2.1 Trechos retos com velocidade constante (aceleração nula).

Vrd (mm/s)	Vre (mm/s)	Trinca numérica			Dist. Est.(mm)	Dist. Med(mm)	θ(rd)
		Tempo(x21ms)	VRD	VRE			
34	34	1386	200	-200	1000	1000	0
34	34	2772	200	-200	2000	2010	0,05
89	89	536	500	-500	1000	1000	0
89	89	1072	500	-500	2000	1990	-0,05
257	257	185	1426	-1426	1000	1005	0,1
257	257	370	1426	-1426	2000	1990	0,12

Tabela 6.2.2 Trecho reto com A. const., Vel. const. e Desacel. const.

Ard (mm/s^2) Vrd (mm/s)	Are (mm/s^2) Vre (mm/s)	Trinca numérica			Dist. Est.(mm)	Dist. Med(mm)	θ(rd)
		Tempo(x21ms)	VRD	VRE			
15 25	15 25	640	150	-150	1599	1596	0
- 216	- 216	200	1200	-1200	908,7	908	0
-15 216	-15 216	640	1200	-1200	1599	1596	0
		Total			4106,7	4100	0,07

Tabela 6.2.3 Trechos de curva com velocidade constante (aceleração nula).

Raio (mm)	Ângulo (°)	Vrd (mm/s)	Vre (mm/s)	Trinca numérica			ARC RD(mm) / RE(mm) Est.	ARC RD(mm) / RE(mm) Med.
				Tempo(x21ms)	VRD	VRE		
500	-180	100	54	972	561	-307	2042 1100	2041 1100
150	-180	100	0	449	561	0	942 -	942 -
500	180	54	100	972	307	-561	1100 2042	1105 2040
1000	180	74	100	1720	417	-561	2670 3613	2690 3603

Pode-se perceber que nos trechos retos de baixa velocidade e pequena distância percorrida, o erro associado foi inexistente. Para trechos maiores e com velocidade de translação maior o erro tem efeito acumulativo, ficando em aproximadamente 10 mm a cada dois metros, Tabela 6.2.1. O fato de o robô estar em repouso e instantaneamente ser submetido a uma velocidade de translação constante explica talvez os erros de precisão (máximo sobresinal).

No trecho reto constituído de três subtrechos, rampa de aceleração, velocidade constante e rampa de desaceleração, Tabela 6.2.2, a precisão e a repetibilidade mostrou-se satisfatória. Neste trecho de 4,1 metros o erro foi de aproximadamente 7 mm e o desvio foi de 0,07 radianos. Para realização da rampa foi necessário implementar no software do MCU do robô uma rotina de execução de rampa de velocidade. Na aplicação SmartWay foram inseridas as equações da cinemática, para aceleração constante, com o objetivo de automatizar o cálculo das distâncias. É perceptível a necessidade de se criar uma forma de enviar para o robô uma quadra ao invés de trinca de números, sendo que o quarto número deverá ser o valor sinalizado de rampa de aceleração (ou desaceleração). Abaixo é mostrada a possível sintaxe da quadra de números.

Tempo*VRDinicial*VREinicial*Aceleração$

Onde:

Aceleração - Um número inteiro sinalizado;

VRDinicial – Velocidade inicial da Roda Direita;

VREinicial – Velocidade inicial da Roda Esquerda.

Na execução das curvas, Tabela 6.2.3, o aumento da velocidade de translação do robô não provocou efeitos de derrapagem e os erros ficaram próximos de 10 mm para raios de até 1000mm.

Os testes foram realizados em dois tipos de superfícies diferentes, uma com piso de madeira em compensado naval e a outra com piso em concreto liso. Em

ambas o resultado foi semelhante, sendo que no piso de concreto a repetibilidade foi maior.

A utilização de encoders com resolução maior pode vir a melhorar a performance de execução da trajetória do robô. Prover para o robô uma trajetória contínua e suave é de extrema dificuldade, pois sempre existirão degraus de velocidade entre trechos retos e curvas, em pelo menos uma das rodas. Nos trechos entre curvas de sentidos opostos existirão degraus de velocidade nas duas rodas. Um algoritmo de controle robusto ou de inteligência artificial como uma rede neural ou lógica fuzzy, seria uma possível solução para se cancelar o máximo sobresinal em qualquer ponto de operação.

7. CONCLUSÕES

O presente trabalho apresentado conseguiu mostrar em seus resultados a validação dos cálculos teóricos e a identificação das restrições físicas e cinemáticas. Estas restrições são fundamentais para o planejamento da trajetória, uma vez que foi observado, por exemplo, no Set Point de velocidade de translação mínima que os controladores da roda direita e esquerda conseguem manter sem oscilações um deslocamento somente a partir de 34 mm/s. Esta limitação está associada à resolução do encoder utilizado. Outro aspecto apresentado é o Set Point de velocidade de translação máximo, cujo valor observado foi de 280 mm/s, ficando um pouco abaixo do esperado, 310 mm/s. Esta limitação está condicionada ao estado de carga da bateria, do módulo de alimentação principal.

Quanto ao sistema de navegação desenvolvido através de interface gráfica, a utilização de algoritmo “A Estrela” apresentou resultados satisfatórios nas simulações, comprovando a performance do programa quanto ao tempo de execução e exatidão para ambientes considerados de grande dificuldade física. Para um mapa constituído de 96 x 128 células (ou quadrados) e obstáculos na forma de labirinto o tempo de convergência da solução foi inferior a 1 segundo, mostrando ser perfeitamente adequado para esta utilização, uma vez que a aplicação deste robô não requer uma ação imediata.

A comprovação prática de parte das simulações desenvolvidas não está no escopo desse trabalho, porém sua implementação pode fazer parte de trabalhos posteriores, utilizando as orientações fornecidas no item 5.

Concluindo, este trabalho pode ser considerado válido sob o aspecto de desenvolvimento do programa embarcado no microcontrolador do robô e no planejamento de sua trajetória, porém aperfeiçoamentos podem ser realizados em trabalhos posteriores tais como:

- A utilização de sistema operacional em tempo real, que deverá prover maior confiabilidade, compactação de código e precisão nos tempos de execução das tarefas;
- A utilização de controles robustos nos acionamentos para aplicações que requeiram velocidades de deslocamento superiores a 1m/s;
- A reavaliação da trajetória a cada 10 metros devido ao erro acumulativo, através da utilização de dispositivos externos, como câmeras instaladas estrategicamente no ambiente, ou dispositivos internos como sensores de proximidade.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALBAGUL, A., *Dynamic Modelling and Control of a Wheeled Mobile Robot*. PhD thesis, Newcastle University, Newcastle upon Tyne, United Kingdom, 2000.

[2] BORENSTEIN, J., EVERETT, H. R., FENG, L., *Where am I? Sensors and Methods for Mobile Motion Positioning,* 1 ed. Michigan, The University of Michigan, 1996.

[3] SCHUMACHER ADRIAN, *Integration of a GPS aided Strapdown Inertial Navigation System for Land Vehicles*, Master of Science thesis, Stockholm, Sweden, 2006.

[4] PIERI, E.R., *Curso de Robótica Móvel*, UFSC, Programa de Pós-Graduação em Engenharia Elétrica, 2002.

[5] BARRIENTOS, A., PEÑIN, L.F., BALAGUER, C., ARACIL, R., *Fundamentos de Robótica*, 2 ed. New York, McGraw-Hill, 1997.

[6] GREGORY, D., MICHEL, J., *Computational Principles of Mobile Robotics*, 2 ed. New York, Cambridge University, 2000.

[7] MESTER, G., "Modeling the Control Strategies of Wheeled Mobile Robots". In: *Proceedings of the Kandó Konferencia*, pp. 1-4, Budapest, Hungary, Aug. 2006;

[8] ROTTAVA, L., *Análise e Programação de Robôs Móveis Autônomos da Plataforma Eyebot*, dissertação de M.Sc., Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil, 2003.

[9] FREESCALE SEMICONDUCTOR: *MC1321x Evaluation Kit (EVK) Reference Manual*, Rev 1.0, 36 pg, 2006.

[10] FREESCALE SEMICONDUCTOR: *ZigBee ™ - Compliant Platform , 2.4 GHz Low Power Transceiver for the IEEE ® 802.15.4 Standard plus Microcontroller Reference Manual*, Rev 1.0, 366 pg , 2006.

[11]ELETRONIC, IF., Disponível em http://www.ifeletronic.kit.net/index.html, acessado em outubro/2007.

[12] FERRATÉ, G., AMAT, J., AYZA, J., BASAÑEZ, L., FERRER, F., HUBER, R., TORRES, C., *Robótica Industrial*, 1 ed. Marcombo, Boixareu Editores., 1986.

[13] BORENSTEIN, J., FENG, L. Correction of systematic odometry errors in mobile robots, 1995.

[14] KLAFTER, R.D., CHMIELEWSKI, T.A., NEGI, M., *Robotic Engineering. An Integrated Approach,* 1 ed. New York, Prentice-Hall International, Inc., 1989.

[15] TANENBAUM, A., *Modern Operating Systems*, 2. ed. New York, Prentice Hall, 2001.

[16] FREESCALE SEMICONDUCTOR: *Simple Media Access Controller (SMAC) User's Guide*, SMACRM, Rev. 1.3, 2005.

[17] GRAY, JAMES, W., *FREESCALE SEMICONDUCTOR: PID Routines for MC68HC11K4 and MC68HC11N4 Microcontrollers*, AN 1215/D, 2004.

[18] CRISTINA, S., *Planejamento de Trajetória de um Robô com Rodas*, dissertação de M.Sc., Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 2008.

Apêndices

Apêndice 1: Listagem do Programa em C

Download na placa SRB 13213 do robô

```
#undef BusMaster
/*****************************************************************************************************/
/*                                  Includes Section                                                 */
/*****************************************************************************************************/
#include <hidef.h> /* for EnableInterrupts macro */
/* Inclusão de Biblioteca padrão C */
#include <stdio.h>
#include <math.h>
#include <float.h>
#include "derivative.h" /* include peripheral declarations */
#include "Transceiver.h"
#include "SPI.h"
#include "MyTypes.h"
/*****************************************************************************************************/
/*                                  Constants Section                                                */
/*****************************************************************************************************/
typedef enum
{
    MainMachineIDLE,
    MainMachineReadFrameReceived,
    MainMachineWaitTransceiverIDLE
}_MainStates;
const u08 baTest[] = {"K"}; // Caracter de retorno
/*****************************************************************************************************/
/*                              Function Prototipes Section                                          */
/*****************************************************************************************************/
void vfnMainMachineDvr(void);
/*****************************************************************************************************/
/*                              Global Variables Section                                             */
/*****************************************************************************************************/
_sMachineState sMainMachine;
u08 lbFlag = 0;
/* Variáveis Globais */
// Variáveis RE
unsigned int AMORE;
// Variáveis RD
unsigned int AMORD;
// Cnstantes PID
unsigned int kd = 0;//.0001;//0
word tempot,tempot2;
unsigned char sinal = 0,sinal2 = 0; // sinal2 período de amostragem
//word motor;

/* RD */
unsigned int  frestasrd; //amostraperrd,amostraprd,periodord,
```

```
volatile unsigned char contadorrd;//,contaperrd;
//static unsigned int anteriorrd;

/* RE */
unsigned int  frestasre;//amostraperre,amostrapre,periodore,
volatile unsigned char contadorre;//,contaperre;
//static unsigned int anteriorre;
word temporx, caracternum;
word contadigitot = 10000,contadigitovrd = 1000, contadigitovre = 1000; // variáveis de ajuste de
// escala de mutliplicação.( variável tempo com 5 dígitos, vrd e vre com 4 dígitos )
unsigned char caracter, contador=0, contatrinca=0;
unsigned char contanum=0;
// Array de sequencia de NO MÁXIMO 10 trincas de números para as variáveis tempo, vrd e vre
// 3x10 = 30
signed int sequencia [30],vrdrx, vrerx;
signed char sinalvrd = 1;
signed char sinalvre = 1;
/*****************************************************************************************/
/*                              Functions Section                                        */
/*****************************************************************************************/
// Função Delay. Possui resolução de 20 ms. Variável tempo com número máximo de 65535
// Tempo máximo = 65535 x 20ms = 1310 segundos  ou 21 minutos
void delay_20ms(word tempo)
 {
   __RESET_WATCHDOG();
   tempot = tempo;
   sinal = 0;
   __RESET_WATCHDOG();
   while(!sinal);
 }

// Função de execução do trecho da trajetória

void trecho( unsigned int tempo, signed int vrd, signed int vre )
 {
   // RD
   volatile static unsigned int dutyrd;
   volatile static signed int errord, erro_antrd;
   volatile static signed long somard, saidard;
   signed char sinalrd;
   // RE

   volatile static unsigned int dutyre;
   volatile static signed int errore, erro_antre;
   volatile static signed long somare, saidare;
   signed char sinalre;

   tempot2 = tempo;
   sinal2 = 0;
   //vrd=vrd*2;
   //vre=vre*2;
```

```
while(!sinal2)
{
      // PID RD
 if (vrd > 0 )
 {
  sinalrd = 1;
  errord = (vrd*sinalrd - AMORD);
 }
 else
 {
  sinalrd = -1;
  errord = (AMORD - vrd*sinalrd);
 }
 somard += errord>>2;
 saidard = somard + errord>>4 + (errord - erro_antrd)*kd ;
 erro_antrd = errord;
  if (sinalrd == 1)
 {
 if (saidard < 0) saidard = 0;
 if (saidard > 1700) saidard = 1700;

 } else
 {
 if (saidard < -1700) saidard = -1700;
 if (saidard > 0) saidard = 0;
 }
 // Ajuste de escala

 dutyrd = 2900 + saidard*10/17;
 if (dutyrd < 1900 ) dutyrd = 1900;
 if (dutyrd > 3900) dutyrd = 3900;
 if (vrd == 0) dutyrd = 2900;

 // PID RE
 if (vre > 0 )
 {
  sinalre = 1;
  errore = (vre*sinalre - AMORE);
 }
 else
 {
  sinalre = -1;
  errore = (AMORE - vre*sinalre);
 }
 somare += errore>>2;
 saidare = somare + errore>>4 + (errore - erro_antre)*kd ;
 erro_antre = errore;

 if (sinalre == 1)
 {
```

```
    if (saidare < 0) saidare = 0;
    if (saidare > 1700) saidare = 1700;

    } else
    {
    if (saidare < -1700) saidare = -1700;
    if (saidare > 0) saidare = 0;

    }
    // Ajuste de escala

    dutyre =2900 + saidare*10/17;
    if (dutyre < 1900 ) dutyre = 1900;
    if (dutyre > 3900) dutyre = 3900;

    if (vre == 0) dutyre = 2900;

    // Carrega PWM RD
    //dutyre = vre;

    TPM2C1SC = 0x28;
    TPM2C1VH = dutyrd/256;
    TPM2C1VL = dutyrd - TPM2C1VH*256;


    // Carrega PWM RE
    //dutyre = vre;

    TPM2C2SC = 0x28;
    TPM2C2VH = dutyre/256;
    TPM2C2VL = dutyre - TPM2C2VH*256;
   }
  }

/* ISR TPM2OF */

    interrupt 14 void isrtpm2of(void)
    {
    contadorrd++;
    contadorre++;
     if(!--tempot2) sinal2 = 1;
    if(!--tempot) sinal = 1;

    /* Amostra da velocidade RPM da RD*/
    if (contadorrd>1) // 16 x 1,25ms = 20ms
    {
    AMORD = frestasrd; //6* + (1/10);
    frestasrd = 0;
    contadorrd = 0;
    }
```

```
    /* Amostra da velocidade RPM da RE*/

    if (contadorre>1)
    {
    AMORE = frestasre; //6 + (1/10);
    frestasre = 0;
    contadorre = 0;
    }
    TPM2SC;//The TPM2SC register should be read and
    TPM2SC_TOF = 0;//TOF bit must be set to 0 to clear the interrupt request.
    }

//ISR VRD /
    interrupt 12 void isrvrd(void)
    {
    //static unsigned int atualrd;

    /*
    // Carrega valor da contagem no registrador
    atualrd = TPM2C3VH*256;
    atualrd += TPM2C3VL;

    // Calcula o período pela diferença de contagem
    if(atualrd > anteriorrd)
     {
     periodord = atualrd - anteriorrd;
     if (periodord > 10000) periodord = 10000;
     // Soma os valores do periodo obtido
     amostraprd += periodord;
     contaperrd++;
     anteriorrd = atualrd;
     }else anteriorrd = 0;

    // Conta as frestas */
    frestasrd++;
     TPM2C3SC;// The TPM2C3SC register should be read and
    TPM2C3SC_CH3F = 0;// CH3F bit must be set to 0 to clear the interrupt request.
     }

// ISR VRE
    interrupt 13 void isrvre(void)
    {
    /*static unsigned int atualre;
    // Carrega valor da contagem no registrador
    atualre = TPM2C4VH*256;
    atualre += TPM2C4VL;
    // Calcula o período pela diferença de contagem
    if(atualre > anteriorre)
     {
     periodore= atualre - anteriorre;
```

```
      if (periodore > 10000) periodore = 10000;
      // Soma os valores do periodo obtido
      amostrapre += periodore;
      contaperre++;
      anteriorre = atualre;
      }else anteriorre = 0;

     // Conta as frestas  */
     frestasre++;
      TPM2C4SC;// The TPM2C4SC register should be read and
     TPM2C4SC_CH4F = 0;// CH4F bit must be set to 0 to clear the interrupt request.static word atualre;
     }
void main(void)
{
    u16 lwPowerDelay = 50000;
     TransceiverModemRst(0);
    TransceiverModemRstIO(1);            //MODEM RESET
      while(--lwPowerDelay)
    {
        __RESET_WATCHDOG();
    }

    vfnSPI1Init();
    vfnTransceiverInit();// deu craca no reset do modem!
    transceiverinit(); // deu craca na rotina do modem!
    Timerinit();

    EnableInterrupts;

    #ifdef BusMaster
        sMainMachine.bActualState = MainMachineReadFrameReceived;
    #else
        vfnTransceiverSetRxMode();
        sMainMachine.bActualState = MainMachineWaitTransceiverIDLE;
        sMainMachine.bNextState = MainMachineIDLE;
    #endif
     for(;;)//
    {
        __RESET_WATCHDOG();          /* feeds the dog */

        vfnMainMachineDvr();
        vfnTransceiverDvr();
        caracter = baSPI1RxBuffer[5]; // Recebe o caracter através da quinta posição do array
        // baSPI1RxBuffer, proveniente do registrador SPI.

        /*** Protocolo de comunicação ***/
        //
        // Este protocolo faz com que o robô execute um trecho de cada vêz. Os números são
        // passados através do hyperterminal através de arquivo .txt com 10 linhas,
        // seguindo a regra abaixo. A cada caracter enviado pelo arquivo texto, o MCU envia o
        // caracter "K" de volta para o PC,com o hyperterminal, indicando que o caracter foi
```

```
// recebido.
//
// O hyperterminal deverá ser configurado da seguinte forma:
//
// Bps: 38400 , Data bits: 8,  Parity: none, Stop bits: 1,  Flow Control: none,  Character delay: 80ms,  Line dealy:
     //12000 ms
// A trinca deverá ser passada da seguinte forma:
//
// #tempo*vrd*vre$
//
// Onde:
// # caracter que indica o início da trinca
// * caracter que indica separação entre números
// $ caracter que faz o robô executar o trecho
// tempo  Tempo de execucao do trecho. Variavel inteiro sem sinal com ate 5 digitos,
// minimo valor de 1 ( 1 x 20ms = 20 ms) e maximo valor de 65535(65535 x 20ms = 1310s).
//
// vrd  Velocidade da roda direita. Variavel inteiro sem sinal com ate 4 digitos,
// 2100(SAH) < vrd(Parado = aprox. 3100) < 4100 (SH)
//
// vre  Velocidade da roda esquerda. Variavel inteiro sem sinal com ate 4 digitos,
// 2100(SAH) < vre(Parado = aprox. 3100) < 4100 (SH)
if (caracter == '#')
{
 contanum = 0;
}
if (caracter =='*') // Se caracter = "*" incrementa o contador de número
{
 contanum++;
 baSPI1RxBuffer[5]=0; // zera a posição da memória
}
if ( caracter == '-' && contanum == 1 )
{
  sinalvrd = -1;
 baSPI1RxBuffer[5]=0; // zera a posição da memória
}
if ( caracter == '-' && contanum == 2 )
{
  sinalvre = -1;
  baSPI1RxBuffer[5]=0; // zera a posição da memória
}
caracternum = caracter - 48;// conversão de ASCII para número

if (caracternum >=0 && caracternum <= 9 ) // Se caracter for número, monta o número

{
  switch(contanum)
  {
   case 0:// Se contanum = 0 significa que os números peretencem a variável
  //tempo
   caracternum *= contadigitot;// caracternum = caracternum*contadigitot
```

```
   temporx += caracternum;// monta o número de 5 dígiots da variável tempo
   baSPI1RxBuffer[5]=0; // zera a posição da memória
   contadigitot/=10;  // decrementa contador de dígitos
   break;
   case 1: // Se contanum = 1 significa que os números peretencem a variável
  //vrd

   caracternum *= contadigitovrd;// caracternum = caracternum*contadigito
   vrdrx += caracternum;// monta o número de 4 dígiots da variável vrd
   baSPI1RxBuffer[5]=0; // zera a posição da memória
   contadigitovrd/=10;  // decrementa contador de dígitos
   break;
   case 2:  // Se contanum = 2 significa que os números peretencem a variável
  //vre
   caracternum *= contadigitovre;// caracternum = caracternum*contadigito
   vrerx += caracternum; // monta o número de 4 dígiots da variável vrd
   baSPI1RxBuffer[5]=0; // zera a posição da memória
   contadigitovre/=10;  // decrementa contador de dígitos
   break;

   default :
   contanum=0;// Se contanum diferente de 0,1,2 houve erro!!

  }
}

//if (contanum > 2)

if (caracter =='$') // Se caracter = "$" armazena no array sequencia
{

 sequencia[contatrinca]= temporx;
 sequencia[contatrinca+1]= vrdrx*sinalvrd;
 sequencia[contatrinca+2]= vrerx*sinalvre;
 contatrinca += 3;

 baSPI1RxBuffer[5]=0; // zera a posição da memória
 //trecho(10,0,0); // para o robô
 // Reinicializando as variáveis
 contadigitot  = 10000;
 contadigitovrd = 1000;
 contadigitovre = 1000;
 contanum = 0;
 temporx = 0;
 vrerx = 0;
 vrdrx = 0;
 sinalvrd = 1;
 sinalvre = 1;
}
if (caracter =='@') // Se caracter = "@" executa todos os trechos da sequência
{
```

```
          baSPI1RxBuffer[5]=0; // zera a posição da memória
            // Executa todos os trechos
            for (contador=0;contador <= contatrinca-1; contador=contador+3)
            {

              trecho( sequencia[contador], sequencia[contador+1], sequencia[contador+2] );

            }
           trecho(10,0,0); // para o robô
           //baSPI1RxBuffer[5]=0; // zera a posição da memória
           // Reinicia as variáveis
           contador=0;
           contatrinca=0;
           sequencia[30]=0;
          //trecho(temporx,vrdrx,vrerx);
        }

    }// fim do loop for

} // FIM DE MAIN

void vfnMainMachineIDLE(void)
{
    static u16 lwTxTimeout = 1000;

    //Check IRQ Pending
    if(bfnTransceiverCheckIRQPending())
    {
        sMainMachine.bNextState = sMainMachine.bActualState;
        sMainMachine.bActualState = MainMachineWaitTransceiverIDLE;
        return;
    }

    //Packet Received as Valid
    if(bTransceiverStatus &(1<<TransceiverPckReceivedOk))
    {
        bTransceiverStatus &= ~(1<<TransceiverPckReceivedOk);

        sMainMachine.bPrevState = sMainMachine.bActualState;
        sMainMachine.bActualState = MainMachineReadFrameReceived;

        #ifdef BusMaster
            lwTxTimeout = 1000;
        #endif

        return;
    }

    #ifdef BusMaster
        if(!(lwTxTimeout--))
        {
```

```
            lwTxTimeout = 1000;
            sMainMachine.bPrevState = sMainMachine.bActualState;
            sMainMachine.bActualState = MainMachineReadFrameReceived;
            return;
        }
    #endif
}

void vfnMainMachineReadFrameReceived(void)
{
    //Send Test Message

    vfnTransceiverTxMsg(&baTest[0],sizeof(baTest)-1);
    sMainMachine.bActualState = MainMachineWaitTransceiverIDLE;
    sMainMachine.bNextState = sMainMachine.bPrevState;

}

void vfnMainMachineWaitTransceiverIDLE(void)
{
    if(!bfnTransceiverBusy())
    {
        sMainMachine.bActualState = sMainMachine.bNextState;
    }
}

const void (* const paMainFunctions[])(void) =
{
    vfnMainMachineIDLE,
    vfnMainMachineReadFrameReceived,
    vfnMainMachineWaitTransceiverIDLE
};
void vfnMainMachineDvr(void)
{
                paMainFunctions[sMainMachine.bActualState]();
}
```

Apêndice 2: Listagem do Programa SmartWay em Matlab

Listagem do Programa SmartWay

```
function varargout = Aestrela(varargin)
% AESTRELA M-file for Aestrela.fig
%      AESTRELA, by itself, creates a new AESTRELA or raises the existing
%      singleton*.
%
%      H = AESTRELA returns the handle to a new AESTRELA or the handle to
%      the existing singleton*.
%
%      AESTRELA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in AESTRELA.M with the given input arguments.
%
%      AESTRELA('Property','Value',...) creates a new AESTRELA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Aestrela_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Aestrela_OpeningFcn via varargin.
%
```

```
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Aestrela

% Last Modified by GUIDE v2.5 05-May-2007 18:36:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Aestrela_OpeningFcn, ...
                   'gui_OutputFcn',  @Aestrela_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Aestrela is made visible.
function Aestrela_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Aestrela (see VARARGIN)
% Gerando a estrutura de chamadas
handles=guihandles(hObject);
% Choose default command line output for Aestrela
handles.output = hObject;
handles.robs=2; % Raio do obstáculo
handles.mapa=cell(1,1); % matriz celular mapa
handles.mapaobs=cell(1,1); % matriz celular mapa2 de obstáculo

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Aestrela wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Aestrela_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in inicio.
function inicio_Callback(hObject, eventdata, handles)
% hObject    handle to inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

clc
data = guidata(gcbo); % need handles, may need error info
% Cria duas matrizes celulares vazias
mapa=handles.mapa;
mapaobs=handles.mapaobs;
cll=0;
tamanho2=0;
tamanho3=0;
cl=1;
cc=1;
```

```
% Preenche a matriz celular com o número de posição de cada quadrado
mapa=mapaobs;
tamanho=size(mapa);
contador=100;
contador_obstaculo=[];
cl=1; % conta linha
cc=1; % conta coluna
ident=0; % identificador
for cl=1 : tamanho(1,1)
    for cc=1: tamanho(1,2)
        if mapa{cl,cc}==' '
        else
        mapa{cl,cc}=contador;
        end
        contador=contador+1;
    end
    cc=1;
end


% Define o quadrado inicial Qi
xi = str2num(get(data.xi,'String'));
yi = str2num(get(data.yi,'String'));
mapa{xi,yi}=0;

% Define o quadrado final Qf
xf = str2num(get(data.xf,'String'));
yf = str2num(get(data.yf,'String'));
mapa{xf,yf}=5000;

% Inicializa matriz celular lista_aberta
lista_aberta=cell(1,1);
coordenadas=[];
coordenadasfinal=[];coordenadasfinal1=[];coordenadasfinal2=[];coordenadasnormal=[];coord
enadasfinal3=[];
% Inicializa matriz lista_aberta e matriz caminho
lista_aberta_mat=[];
caminho=[];

% Inicializa matriz celular lista_fechada
lista_fechada=cell(1,1);
cf=1;
% Adicionando a lista aberta o qi e quadrados adjacentes desde que passável
xp=xi;
yp=yi;
ganterior=0;
while  mapa{xp,yp}~=5000

    % Chamada da função calculo de f. Graduação do caminho F=G+H

[lista_aberta,lista_fechada]=calculof(cf,xi,yi,xp,yp,xf,yf,lista_aberta,ganterior,lista_
fechada,mapa,tamanho);
    lista_aberta;
    % Se lista aberta está vazia, não existe um caminho livre. A instrução
    % break desvia para fora do loop while
    if isempty(lista_aberta)==0
        % Conversão de célula para matriz
        lista_aberta_mat=cell2mat(lista_aberta);
        % Ordenação por F
        lista_aberta_mat=sortrows(lista_aberta_mat,5);
        % Salvando o caminho, ou seja , o melhor H da lista aberta, errado!!!.
        cc=1;
        for cc=1 :5
            caminho(cf,cc)=lista_aberta_mat(1,cc);
        end

        ganterior=lista_aberta_mat(1,3);
        % Atualizando xp e yp
        cl=1; % conta linha
        cc=1; % conta coluna
        % Identificando o quadrado pai, xp e yp, do mapa a partir do primeiro da matriz
        % lista aberta
        for cl=1 : tamanho(1,1)
            for cc=1: tamanho(1,2)
                if mapa{cl,cc}==lista_aberta_mat(1,1)
                xp=cl;
```

```
                yp=cc;
                end
            end
            cc=1;
        end
        xp;
        yp;
        cf=cf+1;
        else
            break
    end

end
lista_aberta;
lista_aberta_mat;
lista_fechada;
caminho;
% Se lista aberta está vazia, não existe um caminho livre. Todos os
% quadrados do caminho são mostrados. (Instruções após o else)
if isempty(lista_aberta)==0
    % Caminhando do quadrado alvo de pai para filho até o quadrado origem
    cc=1;
    contador=1;
    tamanho2=size(caminho);
    cl=tamanho2(1,1);
    % Se o caminho for direto, plota o mapa diretamente
    if cl==1
        caminho2=caminho;
        else
        cc=cl-1;
        while cl > 0
            if caminho(cl,2)== caminho(cc,1)
                caminho2(contador,:)=caminho(cl,:);
                contador=contador+1;
                caminho2(contador,:)=caminho(cc,:);
                if abs(cc-cl)>1
                    cl=cc;
                    cc=cl-1;
                else
                    cl=cl-1;
                    cc=cl-1;
                end
                if caminho(cl,2)==0
                    %caminho2(contador+1,:)=caminho(cl,:);
                    cl=0;
                end
            else
                cc=cc-1;
            end

        end
        % Substituindo o conteúdo dos quadrados da trajetória por '.' e criando
        % a matriz de coordenadas.
        caminho2
        tamanho2=size(caminho2);
        cl=1;
        contacoord=1;
        for cll=1: tamanho2(1,1)
            for cl=1 : tamanho(1,1)
                for cc=1: tamanho(1,2)
                    if mapa{cl,cc}==caminho2(cll,1)
                       mapa{cl,cc}='.';
                       coordenadas(contacoord,:)=[cl,cc];
                       contacoord=contacoord+1;
                    end
                end
             cc=1;
            end
           cl=1;
        end
        % Acrescentando a matriz de coordenadas o ponto inicial
        coordenadas(tamanho2(1,1)+1,:)=[xi,yi];
    end
        % Eliminando as coordenadas com x igual ou y igual
    coordenadas
    tamanho3=size(coordenadas);
    cl=1;
```

```
    contacoord=1;
    % Eliminando as coordenadas com x igual
    contanum = 1;
    for cl=1 : (tamanho3(1,1)-1)

        if coordenadas(cl,1) ~= coordenadas(cl+1,1)
            coordenadasfinal(contacoord,:)=coordenadas(cl,:);
            contacoord=contacoord+1;
            contanum = 1;
        elseif contanum < 2
            coordenadasfinal(contacoord,:)=coordenadas(cl,:);
            contacoord=contacoord+1;
            contanum = contanum + 1;
        end
    end

    % Acrescentando a matriz de coordenadasfinal o ponto final
    tamanho4=size(coordenadasfinal);
    coordenadasfinal(tamanho4(1,1)+1,:)=coordenadas(tamanho3(1,1),:);
    tamanho4=size(coordenadasfinal);
    contacoord=1;
    contanum = 1;
    % Eliminando as coordenadas com y igual
    for cl=1 : (tamanho4(1,1)-1)
        if coordenadasfinal(cl,2) ~= coordenadasfinal(cl+1,2)
            coordenadasfinal1(contacoord,:)=coordenadasfinal(cl,:);
            contacoord=contacoord+1;
            contanum = 1;
        elseif contanum < 2
            coordenadasfinal1(contacoord,:)=coordenadasfinal(cl,:);
            contacoord=contacoord+1;
            contanum = contanum + 1;
        end
    end
    % Acrescentando a matriz de coordenadasfinal1 o ponto final
    tamanho5=size(coordenadasfinal1);
    coordenadasfinal1(tamanho5(1,1)+1,:)=coordenadasfinal(tamanho4(1,1),:);
    tamanho5=size(coordenadasfinal1);
    coordenadasfinal1;
    contacoord=1;
    contanum = 1;
    % Eliminando os pontos no meio da diagonal
       for cl=1 : (tamanho5(1,1)-1)
        if ( (abs(coordenadasfinal1(cl,1) - coordenadasfinal1(cl+1,1))== 1) & (
abs(coordenadasfinal1(cl,2) - coordenadasfinal1(cl+1,2)) == 1))
            if contanum < 2
            coordenadasfinal2(contacoord,:)=coordenadasfinal1(cl,:);
            contacoord=contacoord+1;
            contanum = contanum + 1;
            end
        else
            coordenadasfinal2(contacoord,:)=coordenadasfinal1(cl,:);
            contacoord=contacoord+1;
            contanum = 1;
        end
    end
    % Acrescentando a matriz de coordenadasfinal2 o ponto final
    tamanho6=size(coordenadasfinal2);
    coordenadasfinal2(tamanho6(1,1)+1,:)=coordenadasfinal1(tamanho5(1,1),:);
    coordenadasnormal=coordenadasfinal2;
    tamanho6=size(coordenadasfinal2);
    % Invertendo a ordem da matriz, ou seja, o último é o primeiro e
    % vice-versa
    contacoord=tamanho6(1,1);
    for cl=1: tamanho6(1,1)
        coordenadasfinal2(contacoord,1)=((tamanho(1,1)+1)-
coordenadasfinal2(contacoord,1));
        coordenadasfinal3(cl,:)=coordenadasfinal2(contacoord,:);
        contacoord=contacoord-1;
    end
    coordenadasfinal1=coordenadasfinal3;
else
    % Todos os quadrados do caminho são mostrados, caso a lista aberta
    % esteja vazia
    tamanho2=size(caminho);
    for cl=1 : tamanho(1,1)
                for cc=1: tamanho(1,2)
```

```
                    for cll=1: tamanho2(1,1)
                        if mapa{cl,cc}==caminho(cll,1)
                        mapa{cl,cc}='.';
                        end
                    end
                    cll=1;
                end
          cc=1;
    end
end

tamanho=size(mapa);
cl=1; % conta linha
cc=1; % conta coluna
% Carregando no mapa os obstáculos novamente
for cl=1 : tamanho(1,1)
        for cc=1: tamanho(1,2)
            if mapa{cl,cc}=='.' | mapa{cl,cc}==' ' | mapa{cl,cc}==[0] |
mapa{cl,cc}==[5000]
            else
               mapa{cl,cc}='';
            end
        end
    cc=1;
end
% Carregando no mapa o símbolo do ponto inicial e final
mapa{xi,yi}='I';
mapa{xf,yf}='F';
% Plotando o mapa com todos os símbolos
cellplot(mapa)

% Plotando o gráfico equivalente do modo contínuo
figure(1)
%% Create axes
axes1 = axes('YTickLabel',{'24','20','15','10','5'});
axis(axes1,[0 tamanho(1,2) 0 tamanho(1,1)]);
title(axes1,'Trajetória contínua do Robô');
xlabel(axes1,'Distância Horizontal (x0.1 m)');
ylabel(axes1,'Distância Verticall (x0.1 m)');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'all');

%% Create plot
plot1 = plot(...
  coordenadasfinal1(:,2),coordenadasfinal1(:,1),...
  'LineWidth',2,...
  'Parent',axes1);
hold on;
%% Create legend
legend1 = legend(...
  axes1,{'Caminho não otimizado'},...
  'FontName','Arial',...
  'FontSize',12,...
  'Position',[0.6517 0.8432 0.2178 0.03764]);

% Gerando a trajetória com as distâncias e ângulos
coordenadasnormal;
cll = length(coordenadasnormal);
clm=cll;
for cl=1:cll
    coordenadasnormalinv(cl,:)= coordenadasnormal(clm,:);
    clm=clm-1;
end
coordenadasnormalinv
tamanho6=size(coordenadasnormalinv);
%contacoord=tamanho6(1,1);
%cl=1;xo=0;yo=0;contareta=0;
%if tamanho6(1,1) > 2
    %coordenadasotima(1,:)=coordenadasnormal(1,:);
   % for cl=2 : contacoord-1

        %trajetoria(cl,1)=dist2(xp,yp,xs,ys)*100;%considerando que quadrado=100mm
       % contareta = contareta+1;

           %if coordenadasfinal1(cl,1)
         %  xo=coordenadasfinal1(cl,1)+1;
```

```
          %  yo=coordenadasfinal1(cl,2)+1;
          %  plot_arc(xo,yo,1);
          %  contareta=0;

   % end
%end
%trajetoria
% Gerando a trajetória com as distâncias e ângulos
contacoord=tamanho6(1,1);
constante=tamanho6(1,1);
cl=1;cl2=1;xp=0;yp=0;xs=0;ys=0;contareta=0;distrers=0;teta=0;
for contareta=2 : contacoord-1
    %contareta=contareta+1;
    %contacoord=contacoord-1;
    xp=coordenadasnormalinv(contareta-1,1);
    yp=coordenadasnormalinv(contareta-1,2);
    xs=coordenadasnormalinv(contareta,1);
    ys=coordenadasnormalinv(contareta,2);
    trajetoria(cl,1)=dist2(xp,yp,xs,ys)*100;%considerando que quadrado=100mm
    trajetoriafinal(cl2,1)=trajetoria(cl,1);
    cl=cl+1;
    cl2=cl2+1;
    xp=coordenadasnormalinv(contareta,1);
    yp=coordenadasnormalinv(contareta,2);
    xs=coordenadasnormalinv(contareta+1,1);
    ys=coordenadasnormalinv(contareta+1,2);
    trajetoria(cl,1)=dist2(xp,yp,xs,ys)*100;%considerando que quadrado=100mm
    % A cada duas retas, calcula-se a distância entre o ponto de origem da
    % reta de entrada e o ponto destino da reta de saída. Após é calculado
    % o ângulo entre as duas retas e depois o comprimento do arco de círculo,
    % sabendo que a distância entre rodas é de 170mm.
    xp=coordenadasnormalinv(contareta-1,1);
    yp=coordenadasnormalinv(contareta-1,2);
    xs=coordenadasnormalinv(contareta+1,1);
    ys=coordenadasnormalinv(contareta+1,2);
    distrers=dist2(xp,yp,xs,ys)*100;
    teta=angulo3(trajetoria(cl-1,1),trajetoria(cl,1),distrers);
    cl=cl+1;
    trajetoria(cl,1)=(180/pi)*teta;
    trajetoriafinal(cl2,1)=trajetoria(cl,1);
    %teta=(180/pi)*teta
    cl=cl+1;
    cl2=cl2+1;
    if contareta == contacoord-1
        xp=coordenadasnormalinv(contareta,1);
        yp=coordenadasnormalinv(contareta,2);
        xs=coordenadasnormalinv(contareta+1,1);
        ys=coordenadasnormalinv(contareta+1,2);
        trajetoriafinal(cl2,1)=dist2(xp,yp,xs,ys)*100;
    end

end
trajetoriafinal

function xi_Callback(hObject, eventdata, handles)
% hObject    handle to xi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
xi=str2num(get(hObject,'String'));
% Hints: get(hObject,'String') returns contents of xi as text
%        str2double(get(hObject,'String')) returns contents of xi as a double


% --- Executes during object creation, after setting all properties.
function xi_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function yi_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to yi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
yi=str2num(get(hObject,'String'));
% Hints: get(hObject,'String') returns contents of yi as text
%        str2double(get(hObject,'String')) returns contents of yi as a double


% --- Executes during object creation, after setting all properties.
function yi_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function xf_Callback(hObject, eventdata, handles)
% hObject    handle to xf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xf as text
%        str2double(get(hObject,'String')) returns contents of xf as a double
xf=str2num(get(hObject,'String'));

% --- Executes during object creation, after setting all properties.
function xf_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function yf_Callback(hObject, eventdata, handles)
% hObject    handle to yf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of yf as text
%        str2double(get(hObject,'String')) returns contents of yf as a double
yf=str2num(get(hObject,'String'));

% --- Executes during object creation, after setting all properties.
function yf_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function xobs_Callback(hObject, eventdata, handles)
% hObject    handle to xobs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xobs as text
%        str2double(get(hObject,'String')) returns contents of xobs as a double


% --- Executes during object creation, after setting all properties.
function xobs_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to xobs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function yobs_Callback(hObject, eventdata, handles)
% hObject    handle to yobs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of yobs as text
%        str2double(get(hObject,'String')) returns contents of yobs as a double


% --- Executes during object creation, after setting all properties.
function yobs_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yobs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in robs1.
function robs1_Callback(hObject, eventdata, handles)
% hObject    handle to robs1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of robs1
set (handles.robs1,'Value',1);
set (handles.robs2,'Value',0);
set (handles.robs3,'Value',0);
% Atualiza robs
handles.robs=2;
% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in robs2.
function robs2_Callback(hObject, eventdata, handles)
% hObject    handle to robs2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of robs2
set (handles.robs1,'Value',0);
set (handles.robs2,'Value',1);
set (handles.robs3,'Value',0);
% Atualiza robs
handles.robs=3;
% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in robs3.
function robs3_Callback(hObject, eventdata, handles)
% hObject    handle to robs3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of robs3
set (handles.robs1,'Value',0);
set (handles.robs2,'Value',0);
set (handles.robs3,'Value',1);
% Atualiza robs
handles.robs=4;
% Gravando a estrutura
guidata(hObject, handles);

function xio_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to xio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xio as text
%        str2double(get(hObject,'String')) returns contents of xio as a double

% --- Executes during object creation, after setting all properties.
function xio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function yio_Callback(hObject, eventdata, handles)
% hObject    handle to yio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of yio as text
%        str2double(get(hObject,'String')) returns contents of yio as a double


% --- Executes during object creation, after setting all properties.
function yio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function xfo_Callback(hObject, eventdata, handles)
% hObject    handle to xfo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xfo as text
%        str2double(get(hObject,'String')) returns contents of xfo as a double


% --- Executes during object creation, after setting all properties.
function xfo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xfo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function yfo_Callback(hObject, eventdata, handles)
% hObject    handle to yfo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of yfo as text
%        str2double(get(hObject,'String')) returns contents of yfo as a double

% --- Executes during object creation, after setting all properties.
function yfo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yfo (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in criar.
function criar_Callback(hObject, eventdata, handles)
% hObject    handle to criar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of criar
data = guidata(gcbo); % need handles, may need error info
xobs= str2num(get(data.xobs,'String'));
yobs= str2num(get(data.yobs,'String'));
xio= str2num(get(data.xio,'String'));
yio= str2num(get(data.yio,'String'));
xfo= str2num(get(data.xfo,'String'));
yfo= str2num(get(data.yfo,'String'));
mapaobs=handles.mapaobs;
tamanho=size(mapaobs);
apaga=0;
if isempty(xobs) | isempty(yobs) | xobs < 1 | xobs > tamanho(1,1) | yobs < 1 | yobs >
tamanho(1,2)
else
robs= handles.robs;
% Define os quadrados obstáculos na forma de círculos
[mapaobs]=plot_obstaculo(mapaobs,xobs,yobs,robs,apaga);
handles.mapaobs=mapaobs;
cellplot(mapaobs)
end
if isempty(xio) | isempty(yio) | isempty(xfo) | isempty(yfo) | xio < 1 | xio >
tamanho(1,1) | yio < 1 | yio > tamanho(1,2) | xfo < 1 | xfo > tamanho(1,1) | yfo < 1 |
yfo > tamanho(1,2) | xio > xfo | yio > yfo
else
% Define os quadrados obstáculos na forma de retas
[mapaobs]=plot_obstaculo_reta(mapaobs,xio,yio,xfo,yfo,apaga);
handles.mapaobs=mapaobs;
cellplot(mapaobs)
end
% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in mapa1.
function mapa1_Callback(hObject, eventdata, handles)
% hObject    handle to mapa1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of mapa1
set (handles.mapa1,'Value',1);
set (handles.mapa2,'Value',0);
set (handles.mapa3,'Value',0);
% Atualiza o mapa
mapa=cell(24,32);
mapaobs=cell(24,32);
handles.mapa=mapa;
handles.mapaobs=mapaobs;
cellplot(mapaobs)

% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in mapa2.
function mapa2_Callback(hObject, eventdata, handles)
% hObject    handle to mapa2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of mapa2
set (handles.mapa1,'Value',0);
set (handles.mapa2,'Value',1);
set (handles.mapa3,'Value',0);
% Atualiza o mapa
```

```
mapa=cell(48,64);
mapaobs=cell(48,64);
handles.mapa=mapa;
handles.mapaobs=mapaobs;
cellplot(mapaobs)
% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in mapa3.
function mapa3_Callback(hObject, eventdata, handles)
% hObject    handle to mapa3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of mapa3
set (handles.mapa1,'Value',0);
set (handles.mapa2,'Value',0);
set (handles.mapa3,'Value',1);
% Atualiza o mapa
mapa=cell(96,128);
mapaobs=cell(96,128);
handles.mapa=mapa;
handles.mapaobs=mapaobs;
cellplot(mapaobs)
% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in Apagar.
function Apagar_Callback(hObject, eventdata, handles)
% hObject    handle to Apagar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of Apagar
data = guidata(gcbo); % need handles, may need error info
xobs= str2num(get(data.xobs,'String'));
yobs= str2num(get(data.yobs,'String'));
xio= str2num(get(data.xio,'String'));
yio= str2num(get(data.yio,'String'));
xfo= str2num(get(data.xfo,'String'));
yfo= str2num(get(data.yfo,'String'));
apaga=1;
mapaobs=handles.mapaobs;
tamanho=size(mapaobs);
if isempty(xobs) | isempty(yobs) | xobs < 1 | xobs > tamanho(1,1) | yobs < 1 | yobs >
tamanho(1,2)
else
robs= handles.robs;
% Define os quadrados obstáculos na forma de círculos
[mapaobs]=plot_obstaculo(mapaobs,xobs,yobs,robs,apaga);
handles.mapaobs=mapaobs;
cellplot(mapaobs)
end
if isempty(xio) | isempty(yio) | isempty(xfo) | isempty(yfo) | xio < 1 | xio >
tamanho(1,1) | yio < 1 | yio > tamanho(1,2) | xfo < 1 | xfo > tamanho(1,1) | yfo < 1 |
yfo > tamanho(1,2) | xio > xfo | yio > yfo
else
% Define os quadrados obstáculos na forma de retas
[mapaobs]=plot_obstaculo_reta(mapaobs,xio,yio,xfo,yfo,apaga);
handles.mapaobs=mapaobs;
cellplot(mapaobs)
end
% Gravando a estrutura
guidata(hObject, handles);


% --- Executes on button press in Criarpipf.
function Criarpipf_Callback(hObject, eventdata, handles)
% hObject    handle to Criarpipf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of Criarpipf
data = guidata(gcbo); % need handles, may need error info
% Define o quadrado inicial Qi
xi = str2num(get(data.xi,'String'));
yi = str2num(get(data.yi,'String'));
% Define o quadrado final Qf
xf = str2num(get(data.xf,'String'));
yf = str2num(get(data.yf,'String'));
```

```
mapaobs=handles.mapaobs;
tamanho=size(mapaobs);
if isempty(xi) | isempty(yi) | isempty(xf) | isempty(yf) | xi < 1 | xi > tamanho(1,1) |
yi < 1 | yi > tamanho(1,2) | xf < 1 | xf > tamanho(1,1) | yf < 1 | yf > tamanho(1,2) |
mapaobs{xi,yi}==' ' | mapaobs{xf,yf}==' '
    set(data.xi,'String','');
    set(data.yi,'String','');
    set(data.xf,'String','');
    set(data.yf,'String','');
else
    mapaobs{xi,yi}='i';
    mapaobs{xf,yf}='f';
end
handles.mapaobs=mapaobs;
cellplot(mapaobs)
% Gravando a estrutura
guidata(hObject, handles);

% --- Executes on button press in Apagarpipf.
function Apagarpipf_Callback(hObject, eventdata, handles)
% hObject    handle to Apagarpipf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of Apagarpipf
data = guidata(gcbo); % need handles, may need error info
% Define o quadrado inicial Qi
xi = str2num(get(data.xi,'String'));
yi = str2num(get(data.yi,'String'));
% Define o quadrado final Qf
xf = str2num(get(data.xf,'String'));
yf = str2num(get(data.yf,'String'));
mapaobs=handles.mapaobs;
tamanho=size(mapaobs);
if isempty(xi) | isempty(yi) | isempty(xf) | isempty(yf) | xi < 1 | xi > tamanho(1,1) |
yi < 1 | yi > tamanho(1,2) | xf < 1 | xf > tamanho(1,1) | yf < 1 | yf > tamanho(1,2) |
mapaobs{xi,yi}==' ' | mapaobs{xf,yf}==' '
    set(data.xi,'String','');
    set(data.yi,'String','');
    set(data.xf,'String','');
    set(data.yf,'String','');
else
    mapaobs{xi,yi}='';
    mapaobs{xf,yf}='';
end
handles.mapaobs=mapaobs;
cellplot(mapaobs)
% Gravando a estrutura
guidata(hObject, handles);

% --------------------------------------------------------------------
function salva_Callback(hObject, eventdata, handles)
% hObject    handle to salva (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Salvando em arquivo a matriz celular de obstáculos
mapaobs=handles.mapaobs;
[arquivo, caminho]=uiputfile('*.mat','Salva o mapa de obstáculos');
save(arquivo, 'mapaobs')

% --------------------------------------------------------------------
function carrega_Callback(hObject, eventdata, handles)
% hObject    handle to carrega (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Carregando o arquivo com a matriz celular de obstáculos
data = guidata(gcbo); % need handles, may need error info
[arquivo]=uigetfile('*.mat','Carrega o mapa de obstáculos');
load (arquivo)
if arquivo ~= 0
    handles.mapaobs=mapaobs;
    cellplot(mapaobs)
    guidata(hObject,handles)
end
```

Apêndice 3: Diagrama de Interligação dos Módulos