CENTRO UNIVERSITARIO DA FEI

PROJETO DE INICIAÇÃO CIENTÍFICA

CONTROLE EMBARCADO DE VELOCIDADE PARA UM ROBO ONMINIDIRECIONAL – ROBOFEI

RELATÓRIO FINAL

Aluno: André de Oliveira Santos
Números FEI: 11.106.283-2
12.109.144-1
Orientador: Flavio Tonidandel
Departamento de Ciência da Computação

Inicio: Outubro de 2007
Conclusão: Julho de 2009

RESUMO

Este projeto visou desenvolver o controle de velocidades embarcado em um robô onminidirecional capaz de se mover em qualquer direção, pertencente à categoria *Small Size* de futebol de robôs da equipe RoboFEI. O robô utilizado é o novo protótipo da equipe, que foi terminado recentemente e exige a pesquisa de novos conhecimentos e tecnologias para que um time funcional possa ser formado. Serão demonstradas as características e a história do futebol de robôs, bem como do protótipo desenvolvido, além dos resultados, e as tarefas futuras.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O projeto de Competições Robóticas, conhecido atualmente como RoboFEI [1], teve inicio em 2003 com o intuito de promover pesquisa e desenvolvimento tecnológico com os alunos de graduação e pós-graduação. Desde seu inicio, projetos em futebol de robôs vêm sendo desenvolvidos, seja no âmbito de hardware ou software. Até hoje, foram concluídos, em pouco mais de cinco anos, diversas iniciações científicas, trabalhos de conclusão de curso e uma dissertação de mestrado. As iniciações científicas já concluídas possibilitaram ao projeto ter uma equipe completa e estável pertencente à categoria *Very Small* [2] com um novo robô, sistema de visão computacional, sistema de controle e estratégia de alto nível, além de ter dado inicio uma equipe da categoria *Small Size* [3], com a finalização recente de um protótipo eletrônico e mecânico.

Assim, é necessário que novos trabalhos de iniciação científica sejam realizados, para que todas as partes que constituem uma equipe possam agora ser desenvolvidas para a nova categoria, aproveitando todo o conhecimento acumulado anteriormente.

1.1 Objetivo

O objetivo dessa Iniciação Científica é desenvolver o sistema de controle de velocidades embarcado em um robô onminidirecional, com a análise de seu modelo cinemático, além do estudo da necessidade e possibilidade da instalação de novos componentes para auxiliar nesta tarefa.

1.2 Justificativa

A equipe de Futebol de Robôs da categoria *Very Small* [2] da FEI tem conseguido ótimos resultados desde que foi criada em 2003, obtendo reconhecimento nacional, e se tornando referência. Devido a isso, surge agora o interesse em explorar novas categorias do

Futebol de Robôs, que se mostrou um amplo campo de pesquisa, com inúmeras possibilidades. A escolha foi pela categoria *Small-Size* [3] que é regulamentada e organizada pela *Robocup* [4], visando sua participação em competições de âmbito nacional e internacional, além de se prestar como plataforma de desenvolvimento e pesquisa.

Recentemente esta equipe foi iniciada, com o desenvolvimento da eletrônica embarcada e dispositivos mecânicos de um protótipo, que agora demanda pesquisa e desenvolvimento das outras partes que constituem a equipe, como o controle que é proposto aqui.

Com o conhecimento adquirido até este momento na categoria Very Small [2] ficou claro que apenas um controle realizado pelo computador remoto, realimentado pelo sistema de visão não é suficiente, devido a sua latência excessivamente grande. Com a implementação de um controle embarcado de velocidades, o robô é capaz de corrigir a si próprio durante este período de latência, e resultados muito melhores podem ser conseguidos.

1.3 Metodologia

Com o conhecimento aprofundado do hardware e software desenvolvido, foi possível, durante a execução do projeto, modificar e melhorar o que foi desenvolvido no robô até o momento.

O trabalho foi iniciado com uma extensa revisão bibliográfica, a fim de permitir avaliar os problemas e soluções existentes para este tipo de controle, pesquisando o modelo cinemático deste robô, e as soluções aplicadas pelas equipes que têm mais experiência.

Alguns programas foram desenvolvidos externamente ao robô, como para calcular as velocidades ideais do robô, realizar a comunicação, e avaliar os dados reais retornados.

A maior parte do trabalho foi realizada em nível de software, já que o hardware foi concluído recentemente. Todos os softwares necessários são gratuitos, ou tem versão licenciada para a instituição.

Também foi verificada a possibilidade de instalação de componentes que pudessem ajudar neste controle como, por exemplo, acelerômetros e giroscópios, que não geraram

grandes alterações no hardware, já que a sua utilização foi prevista e já existem as conexões necessárias.

2. REVISÃO BIBLIOGRÁFICA

2.1 O Futebol de Robôs

O Futebol de Robôs é um campo de pesquisa que tem como foco o desenvolvimento de robôs móveis autônomos, ou seja, desenvolvidos para atuar sem supervisão de um operador humano. Campo de atuação cientifica e técnica que vem crescendo e ganhando destaque mundial, como por exemplo, as sondas de exploração espacial.

Em 1993 foi criada a *Robocup* [4], um projeto mundial para o desenvolvimento da Inteligência Artificial, robótica e assuntos relacionados. Seu objetivo é o desenvolvimento e organização de diversas categorias de Futebol de Robôs, que é uma proposta padrão com um grande universo de possibilidades e tecnologias que podem ser utilizadas, onde todas as pesquisas devem ser publicadas em congressos que acontecem paralelamente às competições, para que seus resultados possam ser conhecidos e aproveitados, e talvez até aplicados à indústria, ou à vida cotidiana.

2.2 O Futebol de Robôs na FEI

O projeto de Futebol de Robôs na FEI começou no ano de 2003, como um grupo de pesquisas dedicado somente à simulação, sem robôs físicos. Esse projeto foi iniciado pelo Profº. Reinaldo Bianchi, que participou do desenvolvimento dos times FUTEPOLI [5] e Guaraná [6], times de Futebol de Robôs de expressão nacional e internacional. Em setembro do mesmo ano aconteceria o 6º SBAI (Simpósio Brasileiro de Automação Inteligente), em conjunto com a *Second IEEE Student Robotics Competition*, onde haveria uma competição da categoria de Futebol de Robôs *Mirosot*, ou *Very-Small* [7]. Surgiu então o interesse em se

construir uma equipe de robôs adequada a esta competição, onde nasceu o projeto RoboFEI [1], coordenado inicialmente pelos professores Reinaldo Bianchi e Flavio Tonidandel.

Desde então ótimos resultados foram alcançados por esta equipe, nas cinco competições nacionais que aconteceram até o momento, a equipe RoboFEI foi primeira colocada em três oportunidades, segunda colocada em duas e terceira colocada na outra.

2.3 Visão Geral

O desenvolvimento de uma equipe de futebol de robôs envolve esforços em frentes diferentes:

a) Visão computacional, que capta a imagem de todo o campo de jogo e possibilita a localização e identificação dos jogadores e de cada elemento do jogo;

b) Estratégia, que com as informações da visão, localiza e identifica cada robô do time, do time adversário e a bola, e baseado nessas posições, avalia as possibilidades e toma uma determinada decisão de defesa ou ataque;

c) Sistema de trajetória, que com as informações da visão e estratégia, define a trajetória que cada jogador deve ter em determinada situação, de modo a desviar de obstáculos, interceptar a bola, entre outros;

d) Sistema de controle, que com as informações da trajetória, determina como cada robô deve ser acionado, controlando sua velocidade linear e angular de modo que este siga a trajetória previamente definida;

e) Projeto mecânico, que projeta e constrói a estrutura física de cada robô, de acordo com as características das categorias (tamanho, tipo de movimentação, etc.);

f) Projeto eletrônico, que projeta e constrói a interface eletrônica de cada robô, para que este receba as informações do controle, interprete e possa executá-las da maneira mais exata possível.

Este projeto teve como objetivo o desenvolvimento de uma das partes que constitui o sistema de controle, tão fundamental quanto qualquer outra parte de uma equipe bem sucedida.

2.4 A plataforma de trabalho

O controle de velocidade embarcado foi desenvolvido para o novo protótipo da categoria Small Size [3], um robô onminidirecional, capaz de mover-se em qualquer direção sem a necessidade de girar em seu próprio eixo, ou realizar em conjunto movimentos de translação e rotação. Ele pode ser visto na Figura 1. Para alcançar um sistema de controle eficiente, o principal é conhecer o sistema que se pretende controlar. Para isso é necessário uma caracterização tanto mecânica quanto elétrica do robô. Este protótipo foi construído como resultado de duas Iniciações Científicas [8] e [9], de onde foram extraídas estas informações.

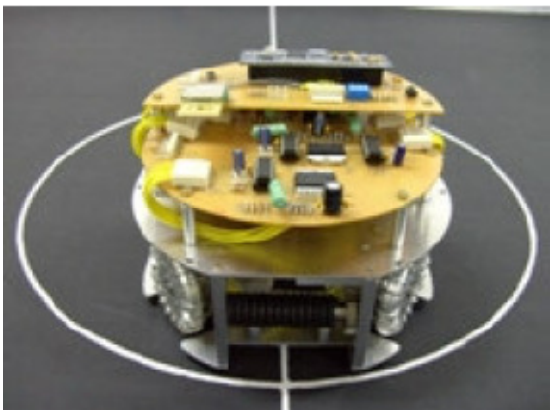


Figura 1 – Protótipo desenvolvido

2.4.1 Caracterização mecânica

O desenvolvimento da mecânica envolve tanto a parte estrutural como de dispositivos do robô, ou seja, se relaciona tanto com o formato, tamanho e peso do robô, como com os dispositivos que têm funções específicas, como o dispositivo de chute ou dispositivo de drible.

O desenvolvimento da estrutura mecânica do robô é a parte do projeto que é mais influenciada pelas regras da competição, já que cada categoria é caracterizada pelo tamanho de seu robô. No caso da categoria *Small Size* [3], o robô deve estar limitado a um cilindro de 180mm de diâmetro e 150mm de altura [10]. Nesta etapa também é definido o tipo de movimentação do robô, que pode ser onminidirecional ou diferencial, e se escolhida o onminidirecional, devem ser definido o número de rodas, que pode ser maior ou igual a três.

O sistema de movimentação escolhido foi o sistema onminidirecional, por ser o que apresenta o maior competitividade e vantagens. A opção foi por utilizar um sistema com quatro rodas onminidirecionais, devido a apresentar uma aceleração mais distribuída em todos os ângulos, e uma maior estabilidade. Um exemplo de roda onminidirecional pode ser visto na Figura 2.

Figura 2 - Roda onminidirecional

Esta roda é composta por uma roda central, que oferece tração na direção normal ao eixo do motor, e por rodas menores montadas em sua periferia que permitem que a roda seja arrastada com menor atrito na direção axial ao eixo do motor. Desde o princípio do projeto descrito em [8] e [9], houve a preocupação em construir um robô que pudesse competir no nível das equipes que participam de campeonatos mundiais, então foram escolhidos motores Faulhaber 2232U006SR [11], que tem um tensão nominal de 6V, potência de 11W, torque nominal de 10mNm, encoder de 512 pulsos por revolução integrado, muito semelhante ao utilizado por equipes de ponta no mundo.

A estrutura do robô foi construída em alumínio, a fim de reduzir o peso, mantendo a resistência mecânica. Basicamente ela é constituída de duas chapas presas por quatro colunas, onde também são fixados os motores [8]. Toda esta estrutura abriga os componentes e dispositivos que compõem o robô, além de oferecer suporte para a fixação da placas de eletrônica.

O protótipo completamente montado apresenta a medidas de 180mm de diâmetro, 130mm de altura, 1,4Kg de massa, e quatro rodas de 55mm de diâmetro [8]. Um diagrama simplificado do robô pode ser visto na Figura 3.

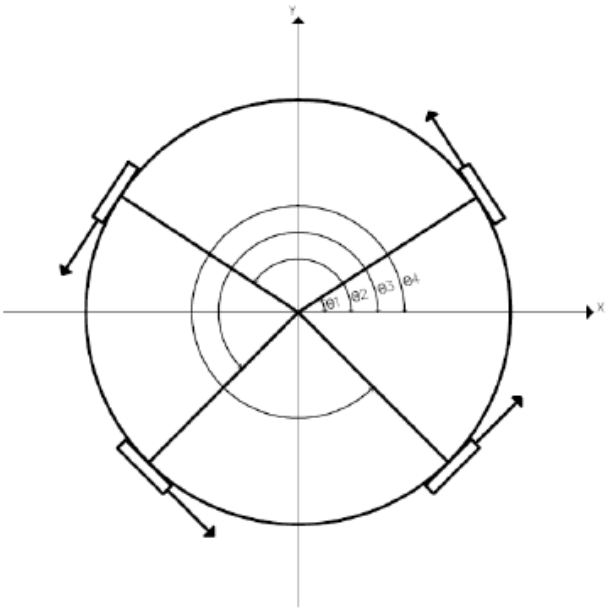


Figura 3 – Modelo Simplificado do robô

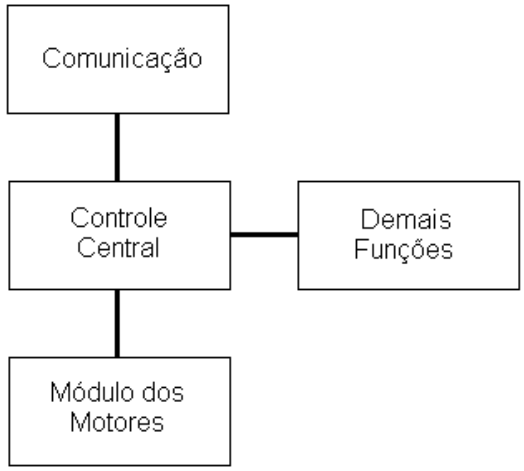
As setas nas roda indicam o sentido positivo de rotação dos motores. Os ângulos Θ1, Θ2, Θ3 e Θ4 valem respectivamente 33°, 147°, 225° e 315°. A disposição das rodas influencia no controle do robô, sendo parâmetros importantes em sua modelagem cinemática, como será visto mais adiante. Os cálculos seriam simplificados se as rodas fossem dispostas de maneira ortogonal, porém é necessário espaço livre à frente do robô, para que possam ser instalados os dispositivos de drible e chute, além disso, com essa configuração o robô seria extremamente instável.

Além da estrutura, também existem os dispositivos de chute, composto basicamente por um solenóide, e o dispositivo de drible, composto por um motor acoplado a um eixo emborrachado, porém esses itens não apresentam relevância ao projeto aqui desenvolvido, sendo citados apenas a título de conhecimento.

2.4.2 Caracterização elétrica

A eletrônica embarcada no robô, descrita detalhadamente em [9], pode ser dividida em duas partes, a eletrônica de controle e a eletrônica de potência. A primeira é responsável por receber as informações vindas do computador remoto, tratá-las e gerar as saídas necessárias de modo a cumprir o que foi determinado. A segunda faz a interface entre os sinais de controle, que tem pouca potência, e o acionamento das cargas, como os motores e o dispositivo de chute, além de gerar os diferentes níveis de tensão que alimentam os circuitos a partir das baterias.

A opção durante o projeto foi por uma eletrônica de controle descentralizada. A Figura [4] mostra um diagrama simplificado desta arquitetura. Um microcontrolador LPC2148 [12] gerencia o robô como um todo, sendo então considerado um controle central. Este microcontrolador apresenta ótimas características, como um núcleo ARM7 [13] de 32-bits, arquitetura utilizada a muito tempo em sistemas embarcados e que se torna mais popular a cada dia, capaz de operar a 60MHz, contando com 32KB de RAM e 512KB de Flash internos, diversos periféricos integrados, tudo isso aliado a um baixo consumo de energia. Ele é responsável por gerenciar o módulo de comunicação, que é recebe e envia informações para o computador remoto, por transmitir as informações referentes à velocidade para o módulo de controle dos motores, além de outras funções como acionamento do dispositivo de chute, sinalização através de *leds*, e demais funções que não são relevantes para este trabalho. Operando desta maneira, este microcontrolador se encontra subaproveitado, porém isso foi intencional em seu projeto, reservando sua capacidade exatamente para a implementação de ações mais complexas [9], como o controle embarcado de velocidade que está sendo proposto aqui.



Figura[4] – Diagrama simplificado da eletrônica de controle

O módulo de controle dos motores aciona tanto os motores de deslocamento como o de retenção da bola, totalizando cinco motores. A escolha para tanto, em [9], foi por utilizar um microcontrolador MC9S08QG8 [14] para o controle individual de cada motor, este microcontrolador tem um núcleo de 8-bits, que opera a até 8MHz e conta com 512Bytes de RAM e 8KB de Flash internos, e também com diversos periféricos integrados. Cada microcontrolador é responsável por monitorar o sinal vindo do *encoder* de seu respectivo motor, analisar este sinal, e calcular a saída necessária para que a velocidade desejada seja alcançada. Além disso, ele é capaz de monitorar a corrente que percorre cada motor, podendo evitar assim a sua queima por sobrecarga ou travamento. Cada MC9S08QG8 [14] se comunica com o LPC2148 [12] através de uma interface serial de alta velocidade, capaz de operar a até 4MHz [9].

Como dito anteriormente, a eletrônica de potência somente faz a interface entre os sinais de pequena intensidade vindos dos microcontroladores, e os dispositivos que devem ser adicionados. O componente de maior relevância aqui é a Ponte-H L298N [15], que faz o acionamento dos motores, permitindo o controle da velocidade do motor, bem como a seleção do sentido de rotação e o monitoramento da corrente consumida por cada motor [9].

3 O PROJETO

3.1 Modelo Cinemático

A principal característica de um robô omnidirecional é que ele é capaz de se movimentar linearmente sobre uma superfície em qualquer direção, sem ter necessariamente que girar antes, além de poder combinar este movimento linear com um movimento de rotação, fazendo com sua movimentação seja muito mais ágil. Essa característica acontece devido a configuração de suas rodas [16], para um entendimento inicial será utilizada a Figura 5.

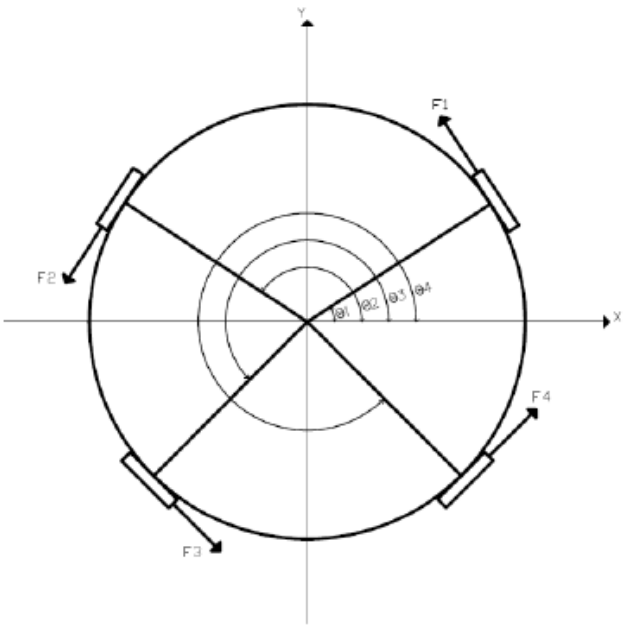


Figura 5 – Modelo onminidirecional baseado nas forças

A primeira informação importante a ser notada é que nesse tipo de movimentação, a referência está sempre fixa ao robô, nunca o ambiente onde ele se encontra, então, a orientação +y refere-se à frente do robô, e +x à sua lateral direita. A partir disso, são definidos

os ângulos Θ_n, que representam a posição de cada motor em relação ao eixo x, bem como a orientação positiva das forças F_n, que representam a força que cada motor aplica ao chão.

Para o inicio das análises, parte-se da segunda lei de Newton para movimentos lineares:

$$F = m \times a \Rightarrow a = \frac{F}{m} \qquad \text{EQ. (1)}$$

Onde:

F: Força [N];

m: Massa [Kg];

a: Aceleração linear [m/s2].

E de maneira semelhante para movimentos rotacionais:

$$\tau = \alpha \times I = F \times R \Rightarrow \alpha = \frac{R}{I} \qquad \text{EQ. (2)}$$

Onde:

τ: Torque [N.m];

α: Aceleração angular [rad/s2]

I: Momento de inércia [Kg.m2]

F: Força

R: Raio

Adaptando as Equações (1,2) para o robô [16], estas equações resultam em:

$$a = \frac{1}{m} \times (F_1 + F_2 + F_3 + F_4) \qquad \text{EQ. (3)}$$

$$\alpha = \frac{R}{I} \times (F_1 + F_2 + F_3 + F_4)$$

EQ. (4)

As forças F1, F2, F3 e F4 representam o vetor gerado por cada motor, para decompô-lo nos eixos x e y é necessário notar que elas são aplicadas com um ângulo _n + 90°, portanto valem as identidades trigonométricas:

$$\cos(x+90) = -sen(x)$$
$$sen(x+90) = \cos(x)$$

EQ. (5,6)

Para a análise da aceleração angular, é preciso lembrar que o momento de inércia para um cilindro com toda a sua massa concentrada em seu centro vale:

$$I = \frac{1}{2} mR^2$$

EQ. (7)

E para um cilindro com toda a sua massa concentrada em sua periferia, o momento de inércia vale:

$$I = mR^2$$

EQ. (8)

Por isso o momento de inércia pode ser escrito como:

$$I = \beta \times m \times R^2$$

EQ. (9)

Utilizamos 0,5 > _ > 1, pois é muito difícil determinar com precisão o centro de massa do robô como um todo, mas é possível afirmar que ele encontra-se entre o seu centro e a sua periferia.

Aplicando as Equações (5,6) na Equação (3), e a Equação (9) na Equação (4), será obtido um sistema de três equações com três incógnitas:

$$a_x = -F_1 \times sen(\Theta_1) - F_2 \times sen(\Theta_2) - F_3 \times sen(\Theta_3) - F_4 \times sen(\Theta_4)$$
$$a_y = -F_1 \times \cos(\Theta_1) + F_2 \times \cos(\Theta_2) + F_3 \times \cos(\Theta_3) + F_4 \times \cos(\Theta_4)$$
$$\alpha = -\frac{1}{\beta \times m \times R} \times (F_1 + F_2 + F_3 + F_4)$$

EQ. (10, 11, 12)

Como este sistema de equações será solvido por métodos computacionais, ele será escrito na forma matricial, o que facilita a sua resolução [16]:

$$\begin{bmatrix} a_x \\ a_y \\ R\alpha \end{bmatrix} = \frac{1}{m} \begin{bmatrix} -sen(\Theta_1) & -sen(\Theta_2) & -sen(\Theta_3) & -sen(\Theta_4) \\ \cos(\Theta_1) & \cos(\Theta_2) & \cos(\Theta_3) & \cos(\Theta_4) \\ \frac{1}{\beta} & \frac{1}{\beta} & \frac{1}{\beta} & \frac{1}{\beta} \end{bmatrix} \times \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

EQ. (13)

Com esta equação matricial é possível obter a aceleração linear e angular que atua sobre o robô, partir da força que cada roda exerce sobre o chão. Nesta equação todos os resultados serão obtidos em m/s^2.

Mas também é possível obter os valores de força de cada motor, a partir das acelerações resultantes, bastando para isso inverter a matriz 3x4, e multiplicar seu resultado pela matriz de acelerações resultantes. A princípio apenas matrizes quadradas podem ser invertidas, mas é possível obter a matriz pseudo-inversa, utilizando método de Moore-Penros visto em [17] e [18]:

$$M^+ = (M^T \times M)^{-1} \times M^T$$

EQ. (14)

A partir dessa definição é possível encontrar a equação que fornece a força necessária em cada motor para gerar as acelerações resultantes necessárias.

Mas o mais importante de toda essa relação é que a partir da derivada da inversa da Equação (13), chegamos a uma relação entre a velocidade resultante do robô e as velocidades de cada roda [16]:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} -sen(\Theta_1) & \cos(\Theta_1) & 1 \\ -sen(\Theta_2) & \cos(\Theta_1) & 1 \\ -sen(\Theta_3) & \cos(\Theta_1) & 1 \\ -sen(\Theta_4) & \cos(\Theta_1) & 1 \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ R\omega \end{bmatrix} \qquad \text{EQ. (15)}$$

As variáveis desta equação são mais bem compreendidas com a ajuda da Figura 6.

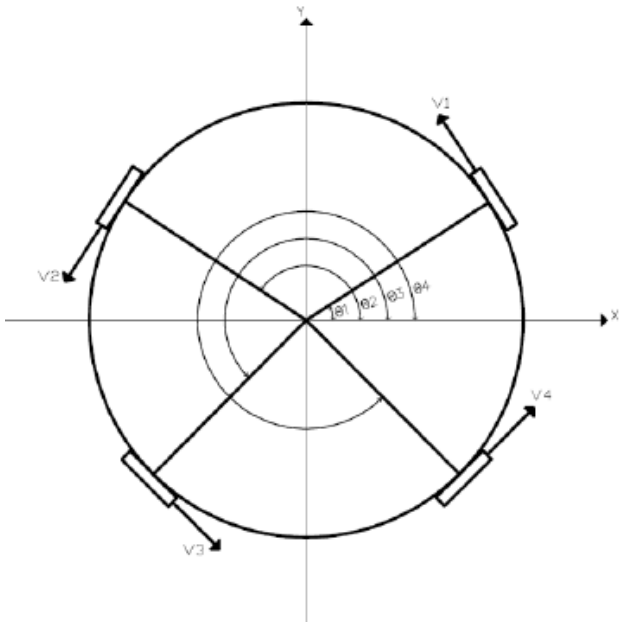


Figura 6 – Modelo onminidirecional baseado nas velocidades

A Equação (15) é sem duvida a mais importante para o controle do robô, pois a partir do vetor de velocidades resultantes sobre o robô, é possível definir a velocidade que cada roda deve ter para que a esta seja obtida. Ela é importante também porque a grandeza que pode ser facilmente medida nas rodas é exatamente a velocidade, através do encoders acoplados aos motores.

Da mesma maneira que anteriormente, é possível aplicar a Equação (14) a esta, para obter a partir da velocidade de cada roda a resultante teórica. Isso pode ser importante para fazer uma verificação entre um modelo ideal e um modelo real do robô permitindo, por exemplo, identificar rodas que estão derrapando [16].

A Equação (13) e a sua inversa podem ser utilizadas com adição de um acelerômetro ao robô, que irá permitir medir a aceleração resultante sobre o robô, e trabalhar com essa grandeza, a fim de obter mais um método para a correção da velocidade.

3.2 Softwares desenvolvidos

Uma das partes mais importantes em um sistema de controle é a aquisição de dados, sem isso não é possível conhecer verdadeiramente o sistema que se pretende controlar. Pensando nisso, a primeira parte desse projeto foi focada no desenvolvimento de softwares que permitissem a captura e o processamento de dados vindos do robô, seja por meio de rádio frequência, ou através do sistema de visão. Outros softwares também foram desenvolvidos com a função de movimentar o robô, já que ainda não existia nenhuma implementação, mesmo que simples, que permitisse controlar o robô como um todo, ou os motores independentemente devido a recente conclusão do protótipo.

3.2.1 Software para a Modelagem Cinemática

Como pode ser visto na seção 3.1, a modelagem cinemática de um robô onminidirecional envolve muitos cálculos, utilizando funções trigonométricas e operações matriciais, o torna a sua utilização demorada e suscetível a erros. Então o primeiro software a ser desenvolvido foi um Software para a Modelagem Cinemática, que pode ser visto na Figura 7.

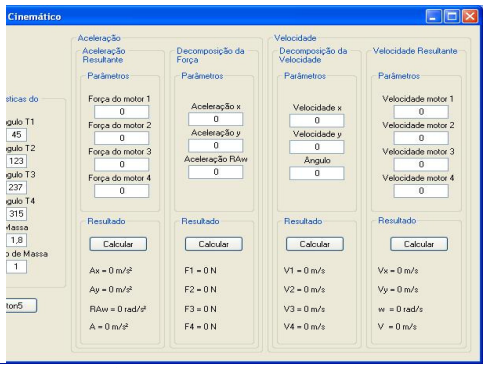


Figura 7 - Software de Modelagem Cinemática

Este software permite a entrada de parâmetros relativos às características do robô, como o ângulo das rodas, massa e posição do centro de massa. Ele então se divide em dois tipos de cálculos, relacionados à aceleração/força ou a velocidade, permitindo a entrada de dados de cada roda de maneira individual, e calculando a resultante sobre o robô, ou partindo da resultante e obtendo a contribuição de cada roda em ambos os casos.

Vale salientar aqui que a modelagem considera condições ideais de funcionamento, como a inexistência de escorregamento da rodas. Porém, para cálculos iniciais, o programa se mostrou de grande valia, diminuindo o tempo de execução dos cálculos e eliminando a possibilidade de erro.

3.2.2 Software de Controle Manual

O segundo software a ser desenvolvido foi o Software de Controle Manual, que pode ser visto na Figura 8.

Com esse software é possível controlar a velocidade de cada motor de deslocamento ou de drible de maneira individual ou conjunta, bem como seu sentido de rotação. A escala de velocidade varia de 100% a -100%, sendo os valores positivos para o que o motor gire no sentido horário.

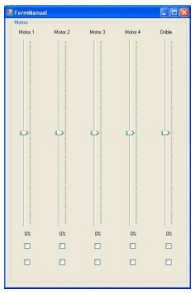


Figura 8 - Software de Controle Manual

Esse software foi utilizado nos testes iniciais do robô, para verificar o alinhamento das rodas, e para observar a resposta dinâmica dos motores, quando submetidos a variação de velocidades. Mais tarde esse software também foi utilizado para verificar a relação entre as velocidades enviadas para o robô e que o mesmo retorna, com um programa que será visto mais a frente.

3.2.3 Software de Joystick

Outro software desenvolvido foi o Software de Joystick, com ele é possível controlar o robô como um todo, fazendo com que ele execute tanto movimentos de translação quanto de rotação. A janela do software pode ser vista na Figura 9.

Figura 9 - Software de Joystick

Com este software é utilizado um joystick USB padrão, que conta com duas alavancas direcionais analógicas, que permitem variar a velocidade resultante sobre o robô. A primeira alavanca permite definir um vetor com módulo, sentido e direção, que é decomposto e aplicado no modelo cinemático do robô gerando o movimento de translação, e a segunda alavanca gera um módulo que representa o movimento de rotação para o robô, estes dois

movimentos são combinados e enviados para o robô. Na tela é possível ver o valor medido para cada eixo do joystick, bem como a velocidade que é calculada em tempo real.

Este software foi útil para verificar em um primeiro momento como o robô se comportaria, em relação a sua aceleração, velocidade final, estabilidade, entre outros, sem um controle de auto nível externo, e mesmo sem um controle embarcado de velocidades, e para posteriormente possibilitar um teste comparativo.

3.2.4 Software de Visão

O Software de Visão utilizado é mesmo que vem sendo utilizado na categoria *Very Small* [2], que pode ser visto em [19]. Ele na verdade foi adaptado para servir ao propósito de captura de dados, a Figura 10 ilustra a tela criada para a utilização do software.

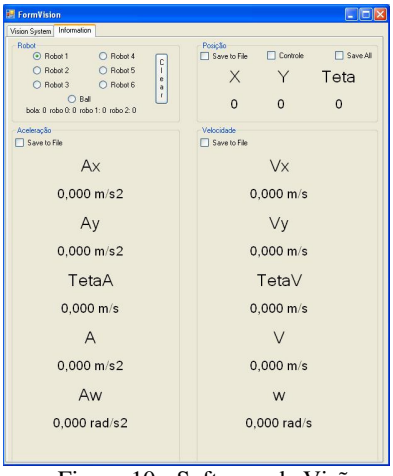


Figura 10 - Software de Visão

Com as adaptações feitas é possível selecionar o robô que será monitorado, ou até mesmo monitorar a bola, e obter em tempo real as medidas de aceleração linear e angular, e as suas decomposições, a velocidade linear e angular, além da posição e orientação do robô dentro do campo de jogo. Também é possível salvar os dados individual ou coletivamente para arquivo de texto, que podem posteriormente serem analisados e utilizados na geração de

gráficos como o da Figura 11, que representa o deslocamento da bola ao longo do eixo x em relação ao tempo.

Figura 11 - Gráfico da posição da bola em função do tempo

3.2.5 Software de Retorno de Informações

Outra etapa fundamental para a aquisição de dados é referente a obter os dados internos do robô, para isso foi criado o Software de Retorno de Informações, que pode ser visto na Figura 12.

Figura 12 - Software de Retorno de Informações

Como é possível ver, nele está integrada a configuração da porta serial, que é a fonte dos dados recebidos e que são enviados pelo transmissor do robô. A informação recebida é referente a velocidade de cada motor obtida pelo microcontrolador e o encoder, dada em porcentagem, que representa a primeira coluna de números. Com base nisso outras

informações são calculadas, como a velocidade em RPM (rotações por minuto) ou em m/s de cada motor, informações estas que servem como um *debug* em tempo real, onde é possível verificar se as velocidades enviadas pelo Software de Controle Manual ou pelo Software de Joystick estão sendo cumpridas. Também, é possível adaptar o software de maneira simples para que outra informações possam ser recebidas, como a corrente instantânea de cada robô, ou a carga das baterias. Este programa será útil por exemplo na determinação da velocidade máxima real de cada motor, e consecutivamente do robô, pois devido as limitações físicas e mecânicas não seria possível estimar esse valores com precisão.

3.3 Firmware desenvolvido

O firmware, ou software embarcado, desenvolvido, tem como objetivo, receber as velocidade calculadas pelo computador remoto, e garantir que cada motor do robô tenha a velocidade correta.

A informação chega até o robô pelo módulo de comunicação, então é imterpretada pelo microcontrolador LPC2148 [12], que envia a informação referente a cada motor para seu respectivo microcontrolador MC9S08QG8 [14]. O firmware deste microcontrolador foi baseado em interrupções, a fim de manter o processamento das informações o mais rápido possível. Ao receber a informação de velocidade, o microcrontrolador verifica se a informação é valida, além de verificar se a velocidade recebida é positiva ou negativa.

Para a medição da velocidade, é necessária uma base de tempo, que é gerada também através de uma interrupção, que acontece em intervalos constantes de tempo. Com a contagem dos pulsos do motor, e esta base de tempo é possível determinar a velocidade da roda naquele momento. Outra interrupção é responsável por verificar a direção em que o motor está rodando, utilizando os dois canais do encoder.

O controle implementado foi do tipo proporcional [20], com uma saída do tipo:

$$saida = Kp \times erro$$

O erro é dado pela diferença entre a velocidade enviada pelo computador remoto, e a velocidade medida pelo robô, a constante de proporcionalidade *Kp* foi determinada experimentalmente. A saída era então normalizada, e transformada em uma porcentagem para o módulo de PWM do microcontrolador.

A velocidade medida pode ser envidada para o computador remoto, e ser mostrada através do software descrito na seção 3.2.5. O firmware desenvolvido pode ser encontrado no Apêndice A.

3.4 Testes realizados

Após o desenvolvimento destas etapas, foram desenvolvidos diversos testes, a fim de comprovar o melhor desempenho do robô com as implementações feitas.

3.4.1 Teste apenas do motor

O primeiro teste teve como objetivo observar a linearidade da velocidade do motor, a relação entre a velocidade desejada e a velocidade real e entre a tensão aplicada e a velocidade de saída. Para o sistema em malha aberta, foram coletados os dados de velocidade em função do ciclo ativo do PWM de acionamento do motor. Já para o sistema em malha fechada, foi coletado o valor da velocidade em função do setpoint enviado para o microcontrolador.

3.4.1.1 Motor sem carga

Este teste foi realizado com o eixo do moto livre, sem carga, para observar a linearidade do motor. De acordo com o modelo teórico do motor de corrente contínua, a velocidade tem uma relação perfeitamente linear com a tensão aplicada a ele, mas devido a imperfeições, perdas por atrito ou aquecimento, isso não é observado.

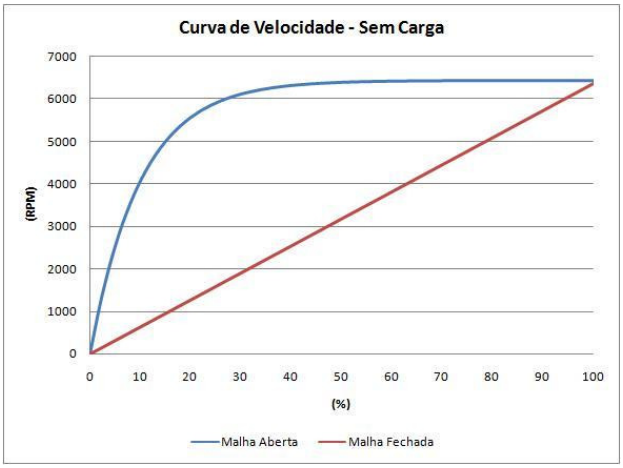


Figura 13 – Curva de velocidade – Sem carga

Como pode ser observado na Figura 13, o motor em malha aberta não apresenta uma relação linear perfeita, ele tende a acelerar muito rapidamente, atingindo a velocidade máxima com uma tensão de aproximadamente 40% da tensão nominal. Enquanto isso, com o controle em malha fechada, é possível observar um comportamento que se aproxima muito do ideal, permitindo determinar facilmente a velocidade em cada ponto de funcionamento, tornando muito mais previsível o funcionamento do motor.

3.4.1.2 Motor com carga

Neste teste, o motor foi testado acoplado à roda do robô, e com esse se movimentando, para observar com seu comportamento com carga. Novamente o esperado é que ele se comportasse de maneira linear ao longo de toda faixa de tensões de alimentação, mas novamente será observado que isso não acontece.

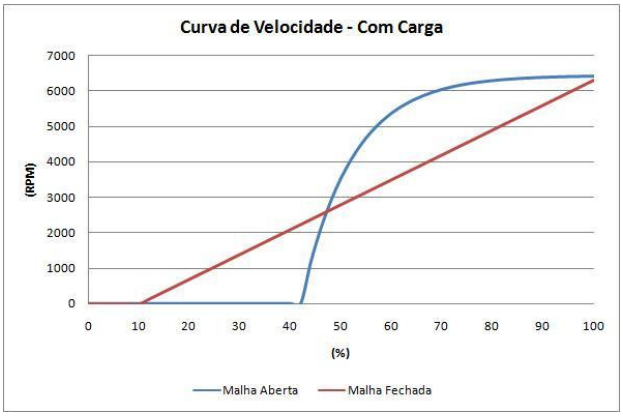


Figura 14 – Curva de Velocidade – Com carga

Na Figura 14, é possivel observar uma outra característica de motores de corrente contínua reais, a chamada zona morta. A zona morta representa uma faixa de velocidades onde o motor não tem torque suficiente para girar com carga. Para o sistema em malha aberta, a zona morta ocorre de 0% a 40%, e após isso ele tende a disparar novamente, apresentando pouca ou nenhuma linearidade. Já com a malha fechada, a zona morta surge apenas até os 10%, isso quer dizer que o motor pode funcionar a velocidades baixas de maneira constante e controlada, isso permite, por exemplo, que o robô se movimente, em baixas velocidades e com controlabilidade. E, além disso, o restante da faixa de operação se aproxima muito do ideal.

3.4.2 Testes de trajetórias com o robô

Os próximos testes tiveram o objetivo de comparar trajetórias feitas pelo robô, comparando o seu comportamento com o controle interno de velocidades, e sem ele, além de diferentes tipos de deslocamentos, onde o arranjo dos motores tem influência.

3.4.2.1 Robô andando para frente

Primeiramente, o robô foi colocado para andar para frente, este é o ângulo de funcionamento onde a composição de forças e velocidades é maior para o robô, permitindo assim, que nessa direção ele apresente o melhor desempenho.

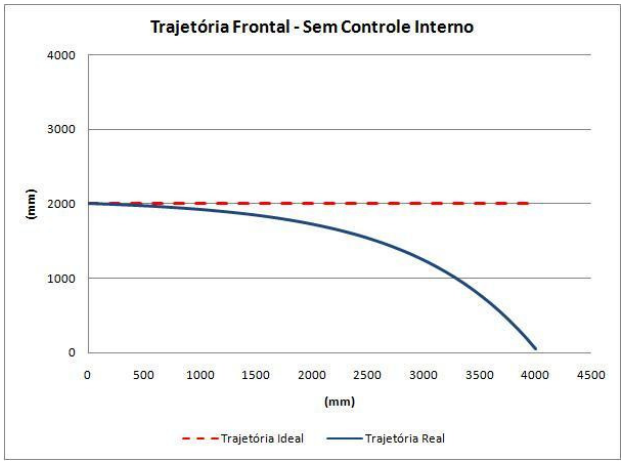


Figura 15 - Trajetória Frontal – Sem Controle Interno

Como pode ser visto na Figura 15, o robô sem o controle interno tende a perder muito a trajetória, mesmo onde a composição de velocidades favorece a ele. Com o controle interno, Figura 16, o desempenho é muito superior, claro que também apresenta erro, mas o acumulado em quatro metros pode ser considerado insignificante se considerarmos que o sistema, como um todo, ainda utiliza outra malha fechada, mais externa, com visão computacional que atenuaria tal erro ainda mais.

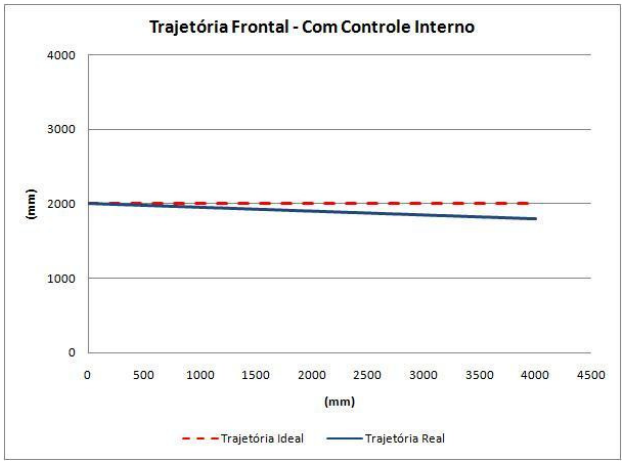


Figura 16 - Trajetória Frontal – Com Controle Interno

3.4.2.2 Robô andando de lado

Neste teste, o robô deveria se deslocar lateralmente. Nesta configuração o arranjo dos motores resulta numa menor força e velocidade, por isso o esperado é que o desempenho do robô seja inferior ao do teste anterior.

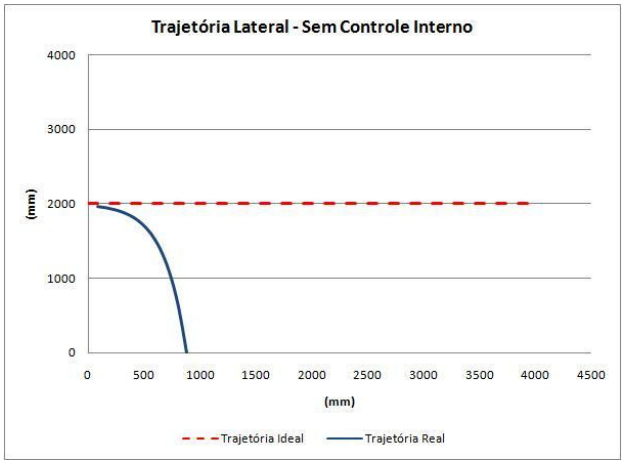


Figura 17 - Trajetória Lateral - Sem Controle Interno

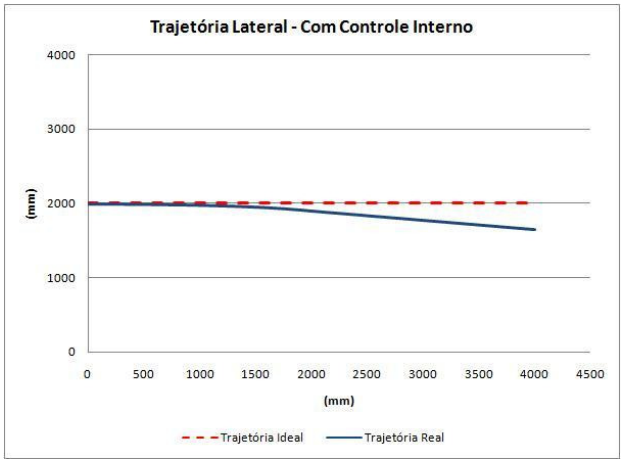


Figura 18 – Trajetória Lateral – Com Controle Interno

Como pode ser visto na Figura 17, sem o sistema de controle, o robô perde rapidamente a trajetória, isso acontece devido ao fato do baixo torque na baixas velocidades,

observado na Figura 14. Isso fica mais evidente neste teste devido ao fato da menor força e velocidade resultantes já comentados antes. Enquanto isso, na Figura18 é possível observar que o comportamento do robô com controle interno novamente é muito superior, desta vez acumulando um erro maior, mas ainda assim satisfatório.

3.4.3 Realimentação pela visão

Os testes realizados até agora já demonstraram a superioridade do sistema com controle interno de velocidades, seja para o motor individualmente, seja para o robô como um todo, porém mais um teste se mostra necessário. O robô em funcionamento durante uma partida é comandado por um computador remoto, que define trajetórias e velocidade baseado em uma realimentação feita através de um sistema de visão. Com isso, a trajetória e velocidades podem mudar com grande velocidade, e era necessário testar como o robô se comportaria nessas circunstâncias.

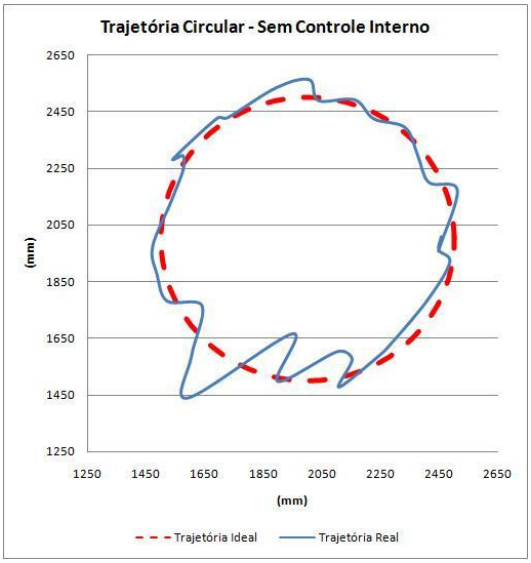


Figura 19 – Trajetória Circular – Sem Controle Interno

Para isso, foi criada no computador uma trajetória circular, e esta enviada para o robô ponto a ponto, a medida que ele avançava. Na Figura 19 fica evidente como o sistema sem

controle interno é instável, entrando em oscilação em diversos pontos da trajetória. Já na Figura 20 o sistema com controle interno mostrou um desempenho novamente melhor. Outra informação que não aparece nos gráficos é referente à velocidade média do robô, devido a zona morta, vista do no item 3.4.1.2, o robô sem controle interno necessariamente precisa andar a uma velocidade mínima maior que a do robô com controle, o que faz com que ele perca a trajetória mais facilmente, além de limitar sua faixa de funcionamento.

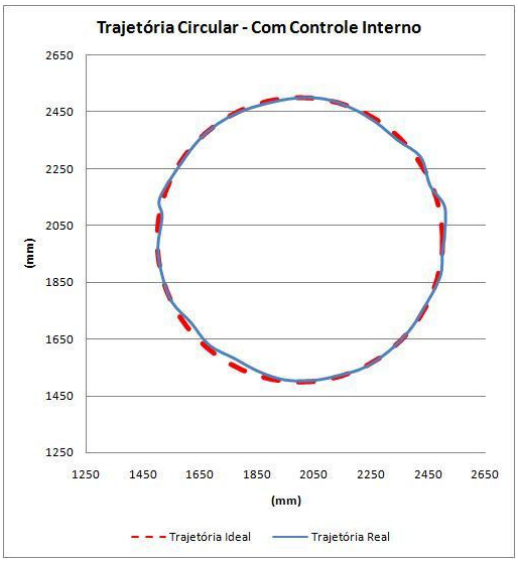


Figura 20 – Trajetória Circular – Com Controle Interno

4 CONCLUSÃO E TRABALHOS FUTUROS

Com a pesquisa realizada ao longo dessa Iniciação Científica, foi possível complementar os trabalhos anteriores, melhorando o que já foi desenvolvido, e abrindo campo para novos trabalhos, além de agregar conhecimento ao projeto de futebol de robôs da FEI. Os testes da seção 3.4 demonstram o bom resultado que foi alcançado, sendo um ótimo ponto de partida pra pesquisas futuras.

Outros pontos podem ser abordados em trabalhos futuros, como um controle de posição embarcado, realimentação através de outros sensores, como acelerômetro ou giroscópio, o que pode trazer um resultado ainda melhor.

5 BIBLIOGRAFIA

[1] - “RoboFEI” - Disponível em: <http://www.fei.edu.br/robo>. Acesso em: 10 Mar. 2008.

[2] - “Robo Soccer Mirosot”. Disponivel em: <http://www.fira.net/soccer/mirosot/overview.html>. Acesso em: 10 Mar. 2008.

[3] - “Small Size Robot League”. Disponível em: <http://small-size.informatik.unibremen.de/>. Acesso em: 10 Mar. 2008.

[4] - “RoboCup Official Site” . Disponível em: <http://www.robocup.org>. Acesso em: 10 Mar. 2008.

[5] - “The Futepoli Team Homepage”. Disponível em: <http://www.lti.pcs.usp.br/robotics/futepoli>. Acesso em: 10 Mar. 2008.

[6] - Guaraná COSTA, A. H.R.; PEGORARO, R. **Construindo Robôs Autônomos para Partidas de Futebol: O time Guaraná.** Controle e Automação SBA, v.11,n.2, p.141-149., 2000.

[7] - “Federation of International Robosoccer Association”. Disponível em: <http://www.fira.net>. Acesso em: 10 Mar. 2008.

[8] - Tavares, Fernando Perez. **Desenvolvimento da Estrutura e Dispositivos Mecânicos do Time de Futebol de Robôs da Categoria Small Size da FEI - ROBOFEI.** Relatório de Iniciação Científica, orientador: Prof. Dr. Flavio Tonidandel. Novembro, 2007. Disponível em: < http://www.fei.edu.br/robo/arquivos/ProjetoIC_Fernando.pdf>. Acesso em: 19 Mar. 2008.

[9] - Santos, André de Oliveira. **Desenvolvimento do Time de Futebol de Robôs da Categoria Small Size da FEI - ROBOFEI.** Relatório de Iniciação Científica, orientador: Prof. Dr. Flavio Tonidandel. Setembro, 2007. Disponível em: < http://www.fei.edu.br/robo/arquivos/ProjetoIC_André.pdf>. Acesso em: 19 Mar. 2008.

[10] - "Laws of the F180 League 2007". Disponível em: < http://smallsize.informatik.unibremen.de/_media/rules:f180rules2007.pdf?id=rules%3Amai &cache=cache>. Acesso em: 10 Mar. 2008.

[11] - "DC-Micromotors 10mNm". Disponível em: <http://www.faulhabergroup. com/uploadpk/e_2232SR_DFF.pdf>. Acesso em: 11 Mar. 2008.

[12] - "User Manual LPC214X". Disponível em <http://www.standardics.nxp.com/support/documents/microcontrollers/pdf/user.manual.lpc21 41.lpc2142.lpc2144.lpc2146.lpc2148.pdf>. Acesso em 10 Mar. 2008.

[13] - "ARM7". Disponível em <http://www.arm.com/products/CPUs/families/ARM7Family.html>. Acesso em 11 Mar. 2008.

[14] - "MC9S08QG8 MC9S08QG4 Data Sheet". Disponível em: <http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S08QG8.pdf>. Acesso em: 12 Mar. 2008.

[15] - "DUAL FULL-BRIDGE DRIVER". Disponível em: <http://www.ortodoxism.ro/datasheets2/2/052daje928cw7pc0uqs1ipyryppy.pdf>. Acesso em: 10 Mar. 2008.

[16] – GLOYE, Alexander; ROJAS, Raúl. **Holonomic Control of a Robot with an Omnidirectional Drive**. Künstliche Intelligenz, 2006.

[17] - MOORE, E. H. **On the reciprocal of the general algebraic matrix.** Bulletin of the American Mathematical Society 26. p.394-395., 1920 .

[18] - PENROSE, Roger. **A generalized inverse for matrices**. Proceedings of the Cambridge Philosophical Society 51. p.406-413. 1955.

[19] – TONIDANDEL, Flavio; MARTINS, Murilo Fernandes, BIANCHI, Reinaldo Augusto C. **Reconhecimento de Objetos em Tempo Real para Futebol de Robôs.** II Jornada de Robótica Inteligente, 2006, Campo Grande-MS. Anais do XXVI Congresso da Sociedade Brasileira de Computação. 2006.

[20] - Ogata, Katsuhiko. **Engenharia de controle moderno.** LTC, Editora, 3º Edição, 2000.

APÊNDICE A – Firmware Desenvolvido

```
/*
** ###################################################################
**     This code is generated by the Device Initialization Tool.
**     It is overwritten during code generation.
**         USER MODIFICATION ARE PRESERVED ONLY INSIDE INTERRUPT SERVICE
ROUTINES
**
**     Project   : Motor_control
**     Processor : MC9S08QG8CPB
**     Version   : Bean 01.241, Driver 01.00, CPU db: 2.87.090
**     Datasheet : MC9S08QG8 Rev. 1.01 10/2005
**     Date/Time : 8/27/2008, 11:25 AM
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     (c) Copyright UNIS, spol. s r.o. 1997-2006
**     UNIS, spol s r.o.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
** ###################################################################
*/

#include <MC9S08QG8.h>              /* I/O map for MC9S08QG8CPB */

#include "my_defines.h"
int ERRO = 0, SETPOINT = 0, CONTAGEM = 0,CICLO = 0, CICLO1 = 0;
char KP = 2, KI = 0, KD = 0, Flag_MTIM = 0;
signed char CLKW = 1;
/*
** ===================================================================
**     Method      :  MCU_init (bean MC9S08QG8_16)
**
**     Description :
**         Device initialization code for selected peripherals.
** ===================================================================
*/
void MCU_init(void)
{
  /* ### MC9S08QG8_16 "Cpu" init code ... */
  /*  PE initialization code after reset */
  /* Common initialization of the write once registers */
  /* SOPT1: COPE=1,COPT=1,STOPE=0,BKGDPE=1,RSTPE=0 */
  SOPT1 = 0xD2;
```

```
  /* SPMSC1: LVDF=0,LVDACK=0,LVDIE=0,LVDRE=1,LVDSE=1,LVDE=1,BGBE=1 */
  SPMSC1 = 0x1D;
  /* SPMSC2: PDF=0,PPDF=0,PPDACK=0,PDC=0,PPDC=0 */
  SPMSC2 = 0x00;
  /* SPMSC3: LVDV=0,LVWV=0 */
  SPMSC3 &= (unsigned char)~0x30;
  /*  System clock initialization */
  ICSTRM = *(unsigned char*far)0xFFAF; /* Initialize ICSTRM register from a non volatile memory
*/
  ICSSC = *(unsigned char*far)0xFFAE;  /* Initialize ICSSC register from a non volatile memory */
  /* ICSC1: CLKS=0,RDIV=0,IREFS=1,IRCLKEN=0,IREFSTEN=0 */
  ICSC1 = 0x04;                        /* Initialization of the ICS control register 1 */
  /* ICSC2: BDIV=0,RANGE=0,HGO=0,LP=0,EREFS=0,ERCLKEN=0,EREFSTEN=0 */
  ICSC2 = 0x00;                        /* Initialization of the ICS control register 2 */
  /* SOPT2: COPCLKS=0,IICPS=0,ACIC=0 */
  SOPT2 = 0x00;
  /* Common initialization of the CPU registers */
  /* PTAPE: PTAPE0=0 */
  PTAPE &= (unsigned char)~0x01;
  /* PTASE: PTASE5=1,PTASE4=1,PTASE3=1,PTASE2=1,PTASE1=1,PTASE0=1 */
  PTASE |= (unsigned char)0x3F;
  /*                                                                                    PTBSE:
PTBSE7=1,PTBSE6=1,PTBSE5=1,PTBSE4=1,PTBSE3=1,PTBSE2=1,PTBSE1=1,PTBSE0=1 */
  PTBSE = 0xFF;
  /* PTADS: PTADS5=0,PTADS4=0,PTADS3=0,PTADS2=0,PTADS1=0,PTADS0=0 */
  PTADS = 0x00;
  /*                                                                                    PTBDS:
PTBDS7=0,PTBDS6=0,PTBDS5=0,PTBDS4=0,PTBDS3=0,PTBDS2=0,PTBDS1=0,PTBDS0=0 */
  PTBDS = 0x00;
  /* ### Init_TPM init code */
  /* TPMSC: TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=0,PS2=0,PS1=0,PS0=0 */
  TPMSC = 0x00;                    /* Stop and reset counter */
  TPMMOD = 0x01A2;                  /* Period value setting */
  TPMC0V = 0x00;                   /* Compare 0 value setting */
  (void)(TPMC0SC == 0);             /* Channel 0 int. flag clearing (first part) */
  /* TPMC0SC: CH0F=0,CH0IE=0,MS0B=1,MS0A=0,ELS0B=1,ELS0A=0 */
  TPMC0SC = 0x28;                   /* Int. flag clearing (2nd part) and channel 0 contr. register setting */
  (void)(TPMSC == 0);               /* Overflow int. flag clearing (first part) */
  /* TPMSC: TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=1,PS2=0,PS1=0,PS0=0 */
  TPMSC = 0x08;                    /* Int. flag clearing (2nd part) and timer control register setting */
  /* ### Init_COP init code */
  SRS = 0xFF;                      /* Clear WatchDog counter */
  /* ### Init_MTIM init code */
  /* MTIMMOD: MOD=0 */
  MTIMMOD = 0x00;
  /* MTIMCLK: CLKS=3,PS=0 */
  MTIMCLK = 0x30;
  /* MTIMSC: TOF=0,TOIE=1,TRST=0,TSTP=0 */
  MTIMSC = 0x40;
```

```
  /* ### Init_RTI init code */
  /* SRTISC: RTIF=0,RTIACK=1,RTICLKS=0,RTIE=1,RTIS2=0,RTIS1=0,RTIS0=1 */
  SRTISC = 0x51;
  /* ### Init_GPIO init code */
  /* PTAD: PTAD2=0,PTAD1=0 */
  PTAD &= (unsigned char)~0x06;
  /* PTADD: PTADD2=1,PTADD1=1 */
  PTADD |= (unsigned char)0x06;
  /* ### Init_GPIO init code */
  /* PTBD: PTBD7=1,PTBD6=1 */
  PTBD |= (unsigned char)0xC0;
  /* PTBDD: PTBDD7=1,PTBDD6=1,PTBDD1=0 */
  PTBDD = (PTBDD & (unsigned char)~0x02) | (unsigned char)0xC0;
  /* ### Init_ADC init code */
  /*                                                                                                  APCTL1:
ADPC7=0,ADPC6=0,ADPC5=0,ADPC4=0,ADPC3=1,ADPC2=0,ADPC1=0,ADPC0=0 */
  APCTL1 = 0x08;
  /* ADCSC2: ADACT=0,ADTRG=0,ACFE=0,ACFGT=0 */
  ADCSC2 = 0x00;
  /*                                                                                                  ADCCV:
ADCV9=0,ADCV8=0,ADCV7=0,ADCV6=0,ADCV5=0,ADCV4=0,ADCV3=0,ADCV2=0,ADCV1=
0,ADCV0=0 */
  ADCCV = 0x00;
  /*                                                                                                  ADCCFG:
ADLPC=0,ADIV1=0,ADIV0=0,ADLSMP=0,MODE1=0,MODE0=0,ADICLK1=0,ADICLK0=0 */
  ADCCFG = 0x00;
  /*  ADCSC1:  COCO=0,AIEN=0,ADCO=0,ADCH4=0,ADCH3=0,ADCH2=0,ADCH1=1,ADCH0=1
*/
  ADCSC1 = 0x03;
  /* ### Init_KBI init code */
  /* KBISC: KBIE=0 */
  KBISC &= (unsigned char)~0x02;
  /*                                                                                                  KBIES:
KBEDG7=0,KBEDG6=0,KBEDG5=0,KBEDG4=1,KBEDG3=0,KBEDG2=0,KBEDG1=0,KBEDG0
=0 */
  KBIES = 0x10;
  /* KBISC: KBIMOD=0 */
  KBISC &= (unsigned char)~0x01;
  /*                                                                                                  KBIPE:
KBIPE7=0,KBIPE6=0,KBIPE5=0,KBIPE4=1,KBIPE3=0,KBIPE2=0,KBIPE1=0,KBIPE0=0 */
  KBIPE = 0x10;
  /* KBISC: KBACK=1 */
  KBISC |= (unsigned char)0x04;
  /* KBISC: KBIE=1 */
  KBISC |= (unsigned char)0x02;
  /* ### Init_SPI init code */
  /* SPIC1: SPIE=0,SPE=0,SPTIE=0,MSTR=0,CPOL=0,CPHA=0,SSOE=0,LSBFE=0 */
  SPIC1 = 0x00;                  /* Disable the SPI module clearing the SPRF flag */
  /* SPIC2: MODFEN=1,BIDIROE=0,SPISWAI=0,SPC0=0 */
```

```
  SPIC2 = 0x10;
  /* SPIBR: SPPR2=0,SPPR1=0,SPPR0=1,SPR2=0,SPR1=0,SPR0=0 */
  SPIBR = 0x10;
  (void)(SPIS == 0);                /* Dummy read of the SPIS registr to clear the MODF flag */
  /* SPIC1: SPIE=1,SPE=1,SPTIE=0,MSTR=0,CPOL=0,CPHA=0,SSOE=0,LSBFE=0 */
  SPIC1 = 0xC0;
  /* ### */
  asm CLI;                          /* Enable interrupts */
} /*MCU_init*/


/*
** ===================================================================
**     Interrupt handler : RTI_Interrupt
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__interrupt void RTI_Interrupt(void)
{

    if(CLKW > 0)                              // sentido de rotação positivo
        CONTAGEM = MTIMCNT + (Flag_MTIM * 256);      // número de pulsos durante os 8ms

    else                                      // sentido de rotação negativo
        CONTAGEM = -(MTIMCNT + (Flag_MTIM * 256));   // número de pulsos durante os 8ms


    ERRO  = SETPOINT - CONTAGEM;                  // erro equivalente

    CICLO =   KP * ERRO;                          // cálculo da saída

    if(CICLO > 0)                             // se a saída é positiva
    {
        DIR1 = 0;                             // gira o motor no sentido positivo
        DIR2 = 1;                             //1
    }

    else if(CICLO < 0)                        // se a saída é negativa
    {
        DIR1  = 1;                            // gira o motor no sentido negativo
        DIR2  = 0;                            //  0
    }

    if (CICLO > 418)                          // testa se a saída é muito grande
        TPMC0V = 418;                         // adequa para o maximo
```

```
    else if ((CICLO < 0)&&(CICLO > -418))            // testa se a saída é negativa e dento dos valores possíveis
        TPMC0V = - CICLO;                    // torna positivo

    else if (CICLO < - 418)                  // testa se a saída é muito "pequena"
        TPMC0V = 418;                        // adequa para o máximo

    else
        TPMC0V = CICLO;                      // se não for muito grande ou muio pequeno, não presica ser adequado

    CICLO1      = TPMC0V;                    // atualiza o valor do CICLO1



    if(CONTAGEM<0)                           // verifica se é negativo
        SPID = (char) - CONTAGEM / 3;            // corrige e envia a velocidade em porcentagem para a SPI
    else                                     // se for positivo
        SPID = (char) CONTAGEM / 3;              // envia a velocidade em porcentagem para a SPI


    MTIMSC_TRST = 1;                         // zera a contagem de pulsos
    Flag_MTIM   = 0;                         // limpa o fator multiplicativo do MTIM
    SRTISC_RTIACK = 1;                       // apaga o flag da interrupção

}
/* end of RTI_Interrupt */


/*
** ===================================================================
**     Interrupt handler : isrVadc
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__interrupt void isrVadc(void)
{
  /* Write your interrupt code here ... */

}
/* end of isrVadc */
```

```
/*
** ===================================================================
**     Interrupt handler : KBI_Interrupt
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__interrupt void KBI_Interrupt(void)
{
    if(!CHANEL_B)           // verifica o canal B em relação ao A
    {

        CLKW = -1;          // sentido negativo
        LED1 = 0;
    }

    else
    {

        CLKW = 1;           // sentido positivo
        LED1 = 1;
    }

    KBISC_KBACK = 1;        // apaga o flag da interrupção
}
/* end of KBI_Interrupt */


/*
** ===================================================================
**     Interrupt handler : SPI_INTERRUPT
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__interrupt void SPI_INTERRUPT(void)
{
    char VEL;                       // variável que garda o dado recebido
    static char c = 0;
    LED2 = ~LED2;                       // pisca o led para sinalização


    if(c==0)
```

```
    {
      VEL = SPIS;                    // lê o STATUS da SPI, parte do processo para apagar o flag
      VEL = SPID;                    // lê o dado recebido

      /*if (VEL != 0)
      {

       TPMC0V = 418;
       DIR1 = 0;                     // gira o motor no sentido positivo
       DIR2 = 1;                     // para o drible
      }
      else
       TPMC0V = 0;*/


      if (VEL < 127)                 // se a velocidade for positiva
         SETPOINT = VEL * 4;             // adequa para o PID

      else                           // se for negativo
         SETPOINT = -((256 - VEL) * 4);     // adequa para o PID
      c = 1;
    }

    else if(c==1)
    {
      KP = SPIS;
      KP = SPID;

      c = 0;
    }

    __RESET_WATCHDOG()               // reseta o Wathdog, anti-travamento
}
/* end of SPI_INTERRUPT */


/*
** ===================================================================
**     Interrupt handler : MTIM_Overflow
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__interrupt void MTIM_Overflow(void)
{
    MTIMSC_TRST = 1;
```

```
    Flag_MTIM++;
}
/* end of MTIM_Overflow */


/*
** ===================================================================
**     Interrupt handler : isrVtpmovf
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
__interrupt void isrVtpmovf(void)
{
  /* Write your interrupt code here ... */

}
/* end of isrVtpmovf */



/* Initialization of the CPU registers in FLASH */

/* NVPROT: FPS=0x7F,FPDIS=1 */
const unsigned char NVPROT_INIT @0x0000FFBD = 0xFF;

/* NVOPT: KEYEN=0,FNORED=1,SEC01=1,SEC00=0 */
const unsigned char NVOPT_INIT @0x0000FFBF = 0x7E;



extern void _Startup(void);

/* Interrupt vector table */
#define UNASSIGNED_ISR 0xFFFF        /* unassigned interrupt service routine */

void (* const _vect[])() @0xFFD0 = {   /* Interrupt vector table */
        RTI_Interrupt,              /* Int.no. 23 Vrti (at FFD0)              Used */
        UNASSIGNED_ISR,             /* Int.no. 22 Reserved2 (at FFD2)         Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 21 Reserved3 (at FFD4)         Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 20 Vacmp (at FFD6)             Unassigned */
        isrVadc,                    /* Int.no. 19 Vadc (at FFD8)              Used */
        KBI_Interrupt,              /* Int.no. 18 Vkeyboard (at FFDA)         Used */
        UNASSIGNED_ISR,             /* Int.no. 17 Viic (at FFDC)              Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 16 Vscitx (at FFDE)            Unassigned */
        UNASSIGNED_ISR,             /* Int.no. 15 Vscirx (at FFE0)            Unassigned */
```

```
    UNASSIGNED_ISR,            /* Int.no. 14 Vscierr (at FFE2)          Unassigned */
    SPI_INTERRUPT,             /* Int.no. 13 Vspi (at FFE4)             Used */
    MTIM_Overflow,             /* Int.no. 12 Vmtim (at FFE6)            Used */
    UNASSIGNED_ISR,            /* Int.no. 11 Reserved13 (at FFE8)       Unassigned */
    UNASSIGNED_ISR,            /* Int.no. 10 Reserved14 (at FFEA)       Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  9 Reserved15 (at FFEC)       Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  8 Reserved16 (at FFEE)       Unassigned */
    isrVtpmovf,                /* Int.no.  7 Vtpmovf (at FFF0)          Used */
    UNASSIGNED_ISR,            /* Int.no.  6 Vtpmch1 (at FFF2)          Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  5 Vtpmch0 (at FFF4)          Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  4 Reserved20 (at FFF6)       Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  3 Vlvd (at FFF8)             Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  2 Virq (at FFFA)             Unassigned */
    UNASSIGNED_ISR,            /* Int.no.  1 Vswi (at FFFC)             Unassigned */
    _Startup                   /* Int.no.  0 Vreset (at FFFE)           Reset vector */
};


/* END */

/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 2.98 [03.80]
**     for the Freescale HCS08 series of microcontrollers.
**
** ###################################################################
*/
```