ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
DEPARTAMENTO DE ENGENHARIA DE SISTEMAS ELETRÔNICOS

GUILHERME BAPTISTA DE SIQUEIRA
RODRIGO DAL PAI FABBRI

VEÍCULO DE TRANSPORTE INDIVIDUAL

"PAPALÉGUAS"

PAPA
LÉGUAS

São Paulo
2010

GUILHERME BAPTISTA DE SIQUEIRA
RODRIGO DAL PAI FABBRI

VEÍCULO DE TRANSPORTE INDIVIDUAL

"PAPALÉGUAS"

Monografia apresentada junto ao curso de
Engenharia Elétrica com ênfase em Sistemas Eletrônicos
para a disciplina PSI2594
Projeto de Formatura I.

Área de Concentração:

Engenharia Elétrica e Eletrônica

Orientador: Prof. Dr. Felipe Pait
Co-orientador: Antonio Spadim

São Paulo
2010

FICHA CATALOGRÁFICA

M2010P

Siqueira, Guilherme Baptista
Papa-léguas: veículo de transporte individual / G.B. Siqueira; R.D.P. Fabbri. -- São Paulo, 2010.
86 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Transporte individual I. Fabbri, Rodrigo Dal Pai II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

T81

2182016

RESUMO

A presente monografia tem como objetivo apresentar detalhadamente as características do Veículo de Transporte individual "Papaléguas", produto do projeto de conclusão da Graduação, englobando temas que vão desde a sua modelagem teórica , passando por seu projeto e implementação e por fim a realização dos testes de validação do protótipo. O funcionamento do veículo é baseado no problema clássico de controle de um pêndulo invertido. Através da utilização de sensores em estado sólido e de um microcontrolador, foi possível conceber um sistema de controle em malha-fechada do tipo PID, que atua em dois motores elétricos, tornando possível o equilíbrio do veículo. O desenvolvimento deste projeto satisfaz as exigências das disciplinas PSI2591- Projeto de Formatura I e PSI2594 – Projeto de Formatura II e foi realizado ao longo do ano de 2010.

Palavras-chave : Pêndulo Invertido , veículo de balanço, Segway, controlador PID.

ABSTRACT

This monograph will present a detailed view of the Individual transport vehicle "Papaléguas", which has been designed for the graduation project in Electrical Engineering. We will discuss topics that go from its mathematic modeling, passes through its project and implementation process and finally terminate with the validation tests. The vehicle functioning is based on the classic Inverted Pendulum Control Problem. By using solid state sensors and a microcontroller, it was possible to create a PID closed-loop control system, that actuates on two DC motors, making possible the equilibration of the vehicle.The development of this project complies with the requirements of PSI2591 – Projeto de Formatura I and PSI2594 – Projeto de Formatura II courses and was made along 2010 year.

Keywords: Inverted pendulum , balancing scooter , Segway , PID controller .

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE GRÁFICOS

LISTA DE TABELAS

INTRODUÇÃO

Nos últimos anos, vem crescendo a necessidade do ser humano em se locomover entre grandes e médias distâncias de forma rápida e barata. Soma-se a este fato a preocupação com o meio-ambiente e a busca por fontes de energia renováveis, tanto para meios de transporte quanto para outros usos.

A principal motivação para os desenvolvedores foi a de criar uma forma limpa, barata e eficiente para o transporte de pessoas. Além disso, também pode-se considerar que uma outra motivação veio do fato de tentar criar um veículo que pudesse ajudar deficientes na sua locomoção.

Para atacar o problema da locomoção eficiente, foi decidido que trabalharíamos num meio de transporte individual, que não usasse rodovias e que tivesse a eletricidade como força motriz.

Dadas as necessidades e a problemática, chegamos à conclusão que um veículo semelhante ao já existente SegWay® seria algo próximo do ideal para solucionar e atender as expectativas do usuário, que procura um veículo de dimensões reduzidas e que pode ser utilizado tanto em ambientes internos quanto externos.

O SegWay consiste em um veículo com duas rodas dispostas paralelamente, as quais suportam um chassi sobre o qual um condutor permanece em pé. Dois motores elétricos de corrente contínua o impulsionam e atuam de forma a garantir o equilíbrio. As curvas são controladas por uma pequena haste semelhante a um joystick, encontrada no guidão.

O problema de controle relacionado a este veículo é o do pêndulo invertido. Este problema consiste no equilíbrio de um pêndulo que possui sua fixação sobre uma base e por possuir tal disposição resulta no posicionamento do centro de gravidade do sistema acima da fixação. O equilíbrio é obtido através de ações geradoras de momentos que compensam a tendência do pêndulo em se desequilibrar. Toda a modelagem matemática será realizada tendo este problema em mente.

O elemento central do veículo, portanto, encontra-se no Sistema de Controle, o qual possibilitará o equilíbrio, a estabilidade e correto funcionamento do mesmo.

Este sistema foi implementado utilizando-se sensores em estado sólido que medem a variação e a posição angular da plataforma, resultando assim em sinais que são enviados a um Microcontrolador, responsável pelo processamento que tornará possível a realização do Controle. O Controlador foi concebido e desenvolvido

utilizando os conceitos de malha-fechada. A ação de controle é realizada através de componentes proporcionais, integrativos e derivativos dos sinais dos sensores, resultando assim em um controlador do tipo PID, largamente utilizado nas mais diversas aplicações na Engenharia.

A alimentação do sistema será dada por uma bateria de chumbo-ácido desenvolvida para ser utilizada em motocicletas.

Como resultado, pretendemos obter um meio de transporte individual compacto, de baixo custo e que seja facilmente manipulado por um indivíduo sem que este necessite de maiores treinamentos. Este meio de transporte deve ser capaz de conduzir uma pessoa de até 120 kg e ser capaz de manter o equilíbrio mesmo em condições adversas.

É esperado que as baterias tenham pelo menos meia hora de duração e que a velocidade máxima atingida seja de ao menos 6 km/h. A bateria deve ser recarregável e, se possível, através de uma tomada comum.

A estrutura do veículo não tem a intenção de parecer profissional. Portanto, uma estrutura simples de metal, de elevada resistência, concebida para suportar o peso acima referido, se mostra suficiente.

A curva de aprendizagem do dispositivo deve ser suave o suficiente para que qualquer usuário, não importando a habilidade, consiga pilotá-lo.

Um fator fundamental para o sucesso do projeto é a correta implementação do sistema de controle. Necessitamos de respostas rápidas dos motores sem que ao mesmo tempo tenhamos geração de forças e momentos que resultem em movimentos abruptos, podendo desta forma ocasionar desconforto ou até mesmo a queda do usuário. Tendo isto em vista, trabalharemos com o compromisso entre tempo de resposta e sobressinal nas respostas do sistema. A sintonia perfeita será obtida através da calibração dos parâmetros do controlador PID, a qual será baseada em resultados teóricos, a partir da modelagem inicialmente, e depois aperfeiçoada através dos resultados empíricos.

1 REVISÃO DE LITERATURA

Para a etapa de revisão de literatura, foram levantados dois trabalhos relacionados, com maior destaque na internet. Os principais são o *MIT DIY Segway [1]* e o *TLB Scooter[3]*. Com base no apresentado nesses dois projetos, selecionamos o resto das matérias e biografia necessária para o entendimento e desenvolvimento do sistema.

1.1 O Problema do Pêndulo Invertido

A modelagem a seguir tem caráter meramente exemplificativo, ou seja, apenas descreve o problema. A sua utilização no projeto em si foi bastante reduzida. O papel desempenhado pela modelagem foi o de analisar as variáveis envolvidas e servir como referência para trabalhos futuros.

Um pêndulo invertido típico é um dispositivo físico que consiste de uma barra cilíndrica, usualmente metálica, a qual é livre para movimentar em torno de um ponto fixo. Esse ponto é montado em um carro que por sua vez é livre para mover na direção horizontal. O carroé acionado por um motor que pode exercer uma força variável no deslocamento do mesmo. Ahaste naturalmente tende a cair, pois sua posição vertical é uma condição de equilíbrio instável.Usa-se uma malha de controle com o objetivo de estabilizar a haste do pêndulo na posição vertical. Isso é possível exercendo-se uma força através do movimento do carro que tende a contrabalançar a dinâmica natural do pêndulo. A intensidade da força pode ser controlada a partir da informação da posição angular da haste. A razão pela qual esse sistema é de interesse para estudos do ponto de vista da tecnologia de controle, é que ele ilustra as dificuldades práticas associadas com aplicações de sistemas de controle no mundo real. Por exemplo, o modelo resultante é muito similar aos usados para estabilização de foguetes em vôo, no posicionamento de guindastes especiais, etc.

O pêndulo invertido é um sistema inerentemente instável e bastante complexo para se analisar por meio de seu modelo matemático completo.Para isto é necessário analisar as relações entre as variáveis do sistema e obter um modelo matemático o mais preciso possível. Geralmente os sistemas dinâmicos são de natureza contínua no tempo, e as equações matemáticas que os descrevem são equações diferenciais.

A figura 1 servirá como base para a modelagem do sistema proposto. Por meio de relações trigonométricas e das decomposições das forças no sistema pode se escrever as relações abaixo:

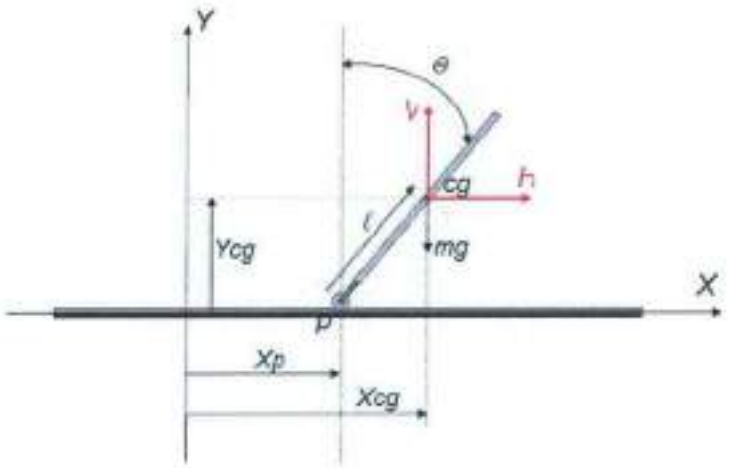


Figura 1 – Representação do problema no eixo cartesiano. [Ricardo Ribeiro, 2007]

Inicialmente serão mostradas as equações que regem o movimento do pêndulo. As derivadas dasequações dos deslocamentos são respectivamente as informações de velocidades e acelerações:

$$X_{cg} = X_p + \ell\,\mathrm{sen}(\theta);$$

$$\dot{X}_{cg} = \dot{X}_p + \ell\cos(\theta)\dot{\theta};$$

$$\ddot{X}_{cg} = \ddot{X}_p + \ell\cos(\theta)\ddot{\theta} - \ell\,\mathrm{sen}(\theta)\dot{\theta}^2;$$

$$Y_{cg} = Y_p + \ell\cos(\theta);$$

$$\dot{Y}_{cg} = \dot{Y}_p - \ell\,\mathrm{sen}(\theta)\dot{\theta}; \quad \dot{Y}_p = 0;$$

$$\ddot{Y}_{cg} = -\ell\,\mathrm{sen}(\theta)\ddot{\theta} - \ell\cos(\theta)\dot{\theta}^2.$$

A somatória das forças na direção *X*, denominada como *H* é dada por:

$$\sum F_x = m\ddot{X}_{cg}$$

$$H = m\left[\ddot{X}_p + \ell\cos(\theta)\ddot{\theta} - \ell\,\mathrm{sen}(\theta)\dot{\theta}^2\right].$$

$$H = m\ddot{X}_p + m\ell\cos(\theta)\ddot{\theta} - m\ell\,\mathrm{sen}(\theta)\dot{\theta}^2.$$

A somatória das forças na direção *Y*, denominada como *V* é expressa por:

$$\sum F_y = m\ddot{Y}_{cg}$$

$$V - mg = m\left[-\ell\,\mathrm{sen}(\theta)\ddot{\theta} - \ell\cos(\theta)\dot{\theta}^2\right],$$

$$V = -m\ell\,\mathrm{sen}(\theta)\ddot{\theta} - m\ell\cos(\theta)\dot{\theta}^2 + mg.$$

A somatória dos momentos de inércia é modelada por:

$$\sum M_{cg} = I\ddot{\theta} + B_r\dot{\theta}$$

$$V\ell\,\mathrm{sen}(\theta) - H\ell\cos(\theta) = I\ddot{\theta} + B_r\dot{\theta}$$

Combinando as equações anteriores vem:

$$I\ddot{\theta} + B_r\dot{\theta} = V\ell\,\mathrm{sen}(\theta) - H\ell\cos(\theta);$$

$$I\ddot{\theta} + B_r\dot{\theta} = \left[-m\ell\,\mathrm{sen}(\theta)\ddot{\theta} - m\ell\cos(\theta)\dot{\theta}^2 + mg\right]\ell\,\mathrm{sen}(\theta) -$$

$$-\left[m\ddot{X}_p + m\ell\cos(\theta)\ddot{\theta} - m\ell\,\mathrm{sen}(\theta)\dot{\theta}^2\right]\ell\cos(\theta);$$

$$I\ddot{\theta} + B_r\dot{\theta} = -m\ell^2\,\mathrm{sen}^2(\theta)\ddot{\theta} - m\ell^2\,\mathrm{sen}(\theta)\cos(\theta)\dot{\theta}^2 -$$

$$+ mg\ell\,\mathrm{sen}(\theta) - m\ell\ddot{X}_p\cos(\theta) - m\ell^2\cos^2(\theta)\ddot{\theta} +$$

$$+ m\ell^2\,\mathrm{sen}(\theta)\cos(\theta)\dot{\theta}^2;$$

$$I\ddot{\theta} + B_r\dot{\theta} = -m\ell^2\ddot{\theta} + mg\ell\,\mathrm{sen}(\theta) - m\ell\ddot{X}_p\cos(\theta);$$

$$(I + m\ell^2)\ddot{\theta} + B_r\dot{\theta} - mg\ell\,\mathrm{sen}(\theta) = -m\ell\ddot{X}_p\cos(\theta).$$

Assumindo que a haste é uniforme, que possui momento de inércia $\frac{m\ell^2}{3}$ e também que Θ é muito pequeno vem:

$$\frac{4}{3}m\ell^2\ddot{\Theta}(t)+B_r\dot{\Theta}(t)-mg\ell\Theta(t)=-m\ell\ddot{X}_p(t);$$

$$\ddot{\Theta}+\frac{3B_r}{4m\ell^2}\dot{\Theta}-\frac{3g}{4\ell}\Theta=-\frac{3}{4\ell}\ddot{X}_p.$$

Visando a obtenção de uma representação padrão, são definidos os parâmetros em (1) obtendo a equação (2), onde aplicando a transformada de Laplace resulta em (3) que é a função de transferência típica de um sistema de pêndulo invertido. [8]

$$2\xi\omega_n=\frac{3Br}{4m\ell^2}, \qquad \omega_n^2=\frac{3g}{4\ell}, \qquad K_p=\frac{3}{4\ell}. \tag{1}$$

$$\ddot{\Theta}(t)+2\xi\omega_n\dot{\Theta}(t)-\omega_n^2\Theta(t)=-K_p\ddot{X}_p(t) \tag{2}$$

$$\boxed{\frac{\Theta(s)}{X(s)}=\frac{-K_p s^2}{s^2+2\xi\omega_n s-\omega_n^2}} \tag{3}$$

1.2 Modelagem de um veículo de balanço

O modelo do pêndulo invertido sobre rodas gerou muito interesse entre pesquisadores recentemente. Esse fato deu-se tanto no nível industrial, quanto em projetos de Hobby e gerou, pelo menos um produto na área comercial: Segway. Esse tipo de robo é caracterizado por duas rodas conectadas à um corpo que, por sua vez, carrega toda eletrônica embarcada (Sensores, microcontrolador, etc...).

Duas rodas independentes no mesmo eixo, um sensor giroscópico – para detectar a velocidade angular – somados à um acelerômetro compõem a eletrônica sensorial do veículo.

Devido a essa configuração de duas rodas cocêntricas cada uma delas é acoplada a um motor DC. O veículo é capaz de fazer curvas estacionárias na forma de U enquanto se mantém equilibrado. Esse tipo de veículo é interessante pois é pequeno e pode fazer curvas facilmente.

O modelo cinemático do problema se provou impossível de controlar, portanto, o equilíbrio do pêndulo só pode ser atingido considerando efeitos dinâmicos.

Tais robôs têm a capacidade de equilibrar-se e ao mesmo tempo fazer uma curva em torno do próprio eixo. Essa manobrabilidade permite a navegação em variados tipos de terrenos. Nota-se que essas capacidades combinadas podem resolver um sem número de desafios das indústrias e sociedades. Por exemplo, uma cadeira de rodas motorizada utilizando esse sistema, ofereceria ao usuário uma capacidade de manobrar muito maior do que a tradicional, fornecendo acesso à um maior número de lugares para o cadeirante. Pequenos veículos que utilizam esse sistema permitem que humanos viajem através de curtas distâncias (Pequenas áreas de fábricas, estacionamentos, etc..) de forma muito mais barata do que se usassem veículos maiores.

Nesse trabalho, um modelo com 3 graus de liberdade de um pêndulo invertido sustentado em duas rodas é derivado e tal modelo será usado para o desenvolvimento de um controlador robusto. A modelagem dinâmica é feita diretamente em termos das variáveis de interesse para o planejamento do controle do sistema: A inclinação e a velocidade. Uma abordagem Newtoniana é utilizada para a geração das equações. A equação de espaço de estados do sistema é da seguinte forma:

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}\mathbf{X}(t) + \mathbf{f}(\mathbf{X}.t)$$

Na qual A e B são matrizes de constante e f (X ,t) é a matriz de incerteza. Essa matriz contém as componentes relativas as perturbações devidas ao chassi e às duas rodas. O modelo não linear foi convertido à um linear usando duas hipóteses colocadas mais adiante.

Modelar é o processo de identificar todos os principais efeitos dinâmicos à serem considerados na análise de um dado sistema. Escrever equações diferenciais e algébricas através das leis da física que se aplicam. E, por fim, reduzindo o sistema de equações à um modelo de equações diferenciais.

São apresentadas as deduções matemáticas e o diagrama de corpo livre *[6]*:

$$(I_p + M_p l^2)\ddot{\theta}_p - \frac{2k_m k_b}{Rr}\dot{x} + \frac{k_m}{R}V_{aL} + \frac{k_m}{R}V_{aR} - M_p g l \sin\theta_p = -M_p l \ddot{x}\cos\theta_p$$

$$\left(2M_r + 2\frac{I_R}{r^2} + M_p\right)\ddot{x} = -\frac{2k_m k_b}{Rr}\dot{x} + \frac{k_m}{Rr}V_{aL} + \frac{k_m}{Rr}V_{aR} - M_p l\dot{\theta}_p^2\cos\theta_p + M_p l\dot{\theta}_p^2\sin\theta_p$$

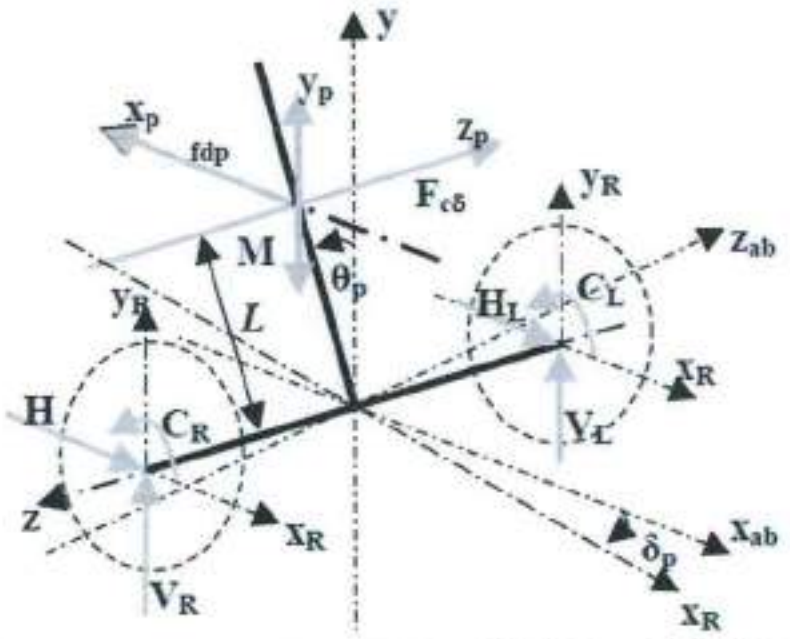


Figura 2 – Diagrama do corpo livre [S.W, Ahmad M.N, Osman J.H.S. 2006]

Para a equação de controle do guidão, temos:

$$-H_R + H_L + \frac{k_m V_{aR}}{Rr} - \frac{k_m V_{aL}}{Rr} = 0$$

$$H_L - H_R = \frac{k_m V_{aL}}{Rr} - \frac{k_m V_{aR}}{Rr}$$

$$\ddot{\delta} = (H_L - H_R)\frac{D}{2I_{pdel}}$$

Substituindo entre as duas últimas equações:

$$\ddot{\delta} = \frac{k_m D V_{aL}}{2RrI_{pdel}} - \frac{k_m D V_{aR}}{2RrI_{pdel}}$$

A forma não linear do espaço de estados torna-se:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_p \\ \ddot{\theta}_p \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{\left(B_4 + \frac{B_5 A_2}{A_1}\right)}{Z_1} & 0 & \frac{B_6}{Z_1} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{\left(A_2 - \frac{A_6 B_5}{B_3}\right)}{Z_2} & 0 & \frac{\left(\frac{A_6 B_6}{B_3}\right)}{Z_2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \\ \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{\left(B_1 + \frac{B_5 A_3}{A_1}\right)}{Z_1} & \frac{\left(B_2 + \frac{B_5 A_4}{A_1}\right)}{Z_1} \\ 0 & 0 \\ \frac{\left(\frac{A_6 B_3}{B_3} - A_3\right)}{Z_2} & \frac{\left(\frac{A_6 B_7}{B_3} - A_4\right)}{Z_2} \\ 0 & 0 \\ -B_7 & B_7 \end{bmatrix} \begin{bmatrix} V_{esq} \\ V_{dir} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{B_5 A_5}{A_1} \\ 0 \\ -A_5 \\ 0 \\ 0 \end{bmatrix}$$

Na qual:

$$\dot{x} = A(x)x + B(x)u + f(x)$$

Aonde:

$$A_1 = I_p + M_p l^2 . A_2 = \frac{2k_m k_e}{Rr}$$

$$A_3 = A_4 = \frac{k_m}{Rr} . A_5 = M_p g l \sin\theta_p$$

$$A_6 = M_p l \cos\theta_p . B_1 = B_2 = \frac{k_m}{Rr}$$

$$B_3 = 2M_w + \frac{2I_w}{r^2} + M_p . B_4 = \frac{2k_m k_e}{Rr^2}$$

$$B_5 = M_p l \cos\theta_p . B_6 = M_p l \dot{\theta}_p \sin\theta_p$$

$$B_7 = \frac{k_m D}{2RrI_{pdal}} . Z_1 = \left(B_5 + \frac{A_6 B_5}{A_1}\right)$$

$$Z_2 = \left(A_1 + \frac{A_6 B_5}{B_3}\right)$$

1.3O controlador PID

O controlador PID é constituído por 3 componentes principais:

Controle proporcional (P): A saída varia baseada em quão longe você está do seu set point. Essa componente é a mais simples das três. A saída varia baseada simplesmente no erro entre a posição atual e a posição desejada. Também há uma constante de ganho, que serve como controle de sensitividade – quanto maior o ganho, mais resposivo o sistema é com relação a erros na saída – do erro do sistema.

A fórmula para o controle proporcional é a seguinte:

$$P = Kp * Kerr$$
$$Kerr = Ponto\ desejado - Ponto\ atual$$

Essa componente sofre do problema do erro do estado estacionário. Esse erro acontece, por exemplo, quando você quer controlar uma posição de um braço mecânico. Quando mais perto o sistema chega do erro nulo, menor a saída do controlador e, eventualmente, o motor pararia de funcionar antes de chegar à um erro nulo. Para eliminar esse erro, geralmente é adicionado o controle integral.

Controle integral (I): A saída varia baseada em quanto tempo leva pra você atingir o seu set point.

O controle integral é similar ao controle proporcional. Porém, ao invés do erro instantâneo, é feita – como o próprio nome sugere – a integral do erro. Ou seja, quanto mais tempo o sistema demora para atingir o ponto desejado, maior a saída do controlador.

Com o controlador I, elimina-se o erro de estado estacionário discutido no controlador P, já que se o motor parar de funcionar antes do set point, a componente I vai aumentando e volta a gerar força para que o sistema atinja o ponto desejado.

A fórmula para o controlador integral é a seguinte:

$$I = Ki * Ierr$$
$$Ierr = Ierr\ anterior + Kerr$$

Controle derivativo(D): A saída varia baseada em variações no erro, evitando variações bruscas na saída.

Essa componente é útil no sentido de manter uma certa posição ou velocidade no sistema em malha fechada.

A fórmula para o controlador derivativo é a seguinte:

$$D = Kd * Derr$$
$$Derr = Kerr - Kerr\ anterior$$

.O controlador PID é simplesmente a combinação dessas três componentes apresentadas. Combinações diferentes entre os termos geram resultados bons para um ou outro problema. Por exemplo, o controlador PI é muito útil para chegar ao ponto desejado rapidamente, porém com o custo de um erro de acurácia. Já o PD pode chegar ao ponto de forma muito rápida e manter-se nele por tempos indeterminados. O PID completo, por sua vez, pode manter a posição com acurávia, mas não será o mais rápido ou suave.

No presente projeto, após discussões teóricas acerca da modelagem e testes empíricos, utilizou-se o controlador PD.

O controlador PD foi utilizado devido ao fato de que tinha-se a derivada do erro posicional, quase sem custo computacional - através do giroscópio - e a parte proporcional através do acelerômetro.

Apesar disso, ainda sim é utilizada a integral. Ver-se-á mais adiante que a saída do giroscópio integrada e o acelerômetro são mais ou menos confiáveis dependendo do tempo total em que foram amostradas. Para tempos menores que um certo T, prefere-se o resultado de um em detrimento de outro.

1.4 Filtro Complementar

Parte fundamental do presente projeto é, através de um acelerômetro e um giroscópio, conseguir uma estimativa razoável para o ângulo de inclinação da plataforma.

O acelerômetro, em teoria, seria suficiente para essa estimativa através de manipulações da saída –que está relacionada diretamente com o ângulo de inclinação, já que mede a força da gravidade aplicada no sensor – que o microcontrolador obtém através de um conversor A/D. Por outro lado, o acelerômetro não fornece boas estimativas em pequenos instantes de tempo. O ruído prejudica a acurácia da medida.

Para resolver esse problema, ficou claro que seria necessária a integração da saída do giroscópio, que apresentou bons resultados em altas freqüências.

Consegue-se, então, obter uma boa estimativa de ângulo através da saída direta do acelerômetro e da integração da saída do giroscópio. A literatura especializada sugere que um filtro do tipo Kalman seria o ideal para esse tipo de estimativa. Após inúmeros estudos, decidiu-se que tal abordagem seria muito complexa e, portanto, inviável de ser implementada em um microcontrolador simples.

Após extensivas pesquisas, foi encontrada uma abordagem eficiente e voltada ao tipo de projeto desenvolvido nesse relatório: O filtro complementar.

Esse filtro consiste em um passa-baixas conectado à saída do acelerômetro e um passa-altas conectado a um integrador que por sua vez está conectado à saída do giroscópio. Isso tudo feito de forma digital, inclusive a integração.

Em pseudo-código, poder-se-ia ter a seguinte equação:

ângulo = (0.98)*(angulo + giroscópio * dt) + (0.02)*(acelerômetro);

O segundo termo faz as vezes de um passa-baixa, o que pode ser entendido da seguinte forma:

ângulo = (0.98)*angulo + (0.02)*acelerômetro;

Fica claro que as variações de alta freqüência serão filtradas, já que o termo relativo à saída do acelerômetro tem o peso de 0.02, enquanto o ângulo da iteração passada tem um peso muito maior.

Por exemplo, se a saída do acelerômetro for de 10 graus, vão ocorrer sucessivas iterações e a variável ângulo vai chegando perto da entrada. Contudo, se há uma mudança brusca pra zero graus numa iteração e depois o retorno pra 10 na seguinte, a variável não vai ter uma diminuição significativa –já que o peso do 0 graus vai ser de apenas 0.02 - e vai continuar a tendência aos 10 graus.

Raciocínio análogo pode ser aplicado à parte de passa-altas relativa ao giroscópio.

A definição das constantes do filtro são feitas com base no período de amostragem do sinal. Ou seja, caso o código de controle principal rode a 100Hz, o período de amostragem é de 0.01 segundos.

A constante de tempo do filtro – o qual define as freqüências para qual atuarão o passa-baixa ou o passa-altas – é calculada através da utilização do tempo de amostragem e das constantes do filtro como se segue.

Seja:

y = (a)*(y) + (1-a)*(x);

A fórmula geral do filtro complementar, a constante de tempo é dada por*[7]*:

$$\tau = \frac{a \cdot dt}{1-a} \leftrightarrow a = \frac{\tau}{\tau + dt}$$

Claro que o caminho é percorrido ao contrário. Primeiro encontra-se o período de amostragem, depois define-se a constante de tempo e, por fim, os coeficientes do filtro são obtidos através da formulação acima.

A constante de tempo é escolhida de forma experimental, foram feitos testes em protoboard e com a ajuda de transferidores para que fosse encontrada a melhor combinação possível entre essas variáveis para a obtenção de uma boa acurácia no ângulo.

Enfim, o presente projeto utilizará o acelerômetro e o giroscópio para que seja obtida uma estimativa de ângulo da base e será aplicado o conceito de controlador PID em malha fechada para que o sistema possa ser devidamente controlado.

2 MATERIAIS E MÉTODOS

2.1 Componentes elétricos

Nesta seção relacionaremos alguns dos componentes, tanto mecânicos quanto elétricos, que pretendemos utilizar na realização do projeto. Tais componentes foram escolhidos mediante pesquisa e consulta a projetos semelhantes já feitos.

Motores Elétricos de Corrente Contínua *"FIRST CIM Motor"*

Em nosso projeto, utilizamos motores elétricos de corrente contínua, alimentados por uma bateria. Para o controle da velocidade deste, utilizamos circuitos eletrônicos de potência denominados Speed Controlers, os quais detalharemos mais abaixo. A exigência de nosso veículo é que tenhamos um grande torque disponível em detrimento da velocidade.Portanto, nossa estratégia é de se aproveitar da grande velocidade de rotação máxima apresentada por estes motores,aplicandoa relação de transformação Torque x Velocidade de rotação através do uso de umacaixa de redução, o que permite a obtenção de um torque elevado.

Para a escolha do motor, levou-se em conta uma carga grande o suficiente para que nela ficassem embutidos os efeitos do momento de inércia da roda.

Supondo uma carga de 120 Kg, pontual e aplicada no meio do guidão (Distante, em altura, 0,5m do motor).O torque necessário,à ser aplicada pelo motor,é de, no máximo (Situação em que o guidão encontra-se a 45 graus do solo, pois o sistema evita inclinações maiores), 420N.m .

Tendo em conta a dificuldade de achar um motor com tamanho torque, utilizou-se uma caixa de redução de 16:1. A partir do valor da redução, calculamos então a potência mínima necessária ao motor. O motor CIM M4-R0062satisfez essa condição, já que:

1.2019 [mN-m/A, à máxima potência] x 40 [A] x 16 [Redução] = 769.216 N.m

Embora o motor nem sempre vá se encontrar na situação de máxima potência, ficou resolvido que seria este o motor na medida em que se procurou um compromisso entre valor e satisfação dos requisitos.

Figura 3 – Motor CIM

Seguem abaixo algumas características do motor:

Performance	
Modelo	M4-R0062-12
Tensão de Operação	: 6v - 12v
Tensão Nominal	: 12v
RPM sem Carga	: 5310
Corrente Sem Carga	: 2.7A
Torque de Stall	: 343.27 oz-in 2424 mN-m
Corrente de Stall	: 133A
Kt	: 2.58 oz-in/A 18.2 mN-m/A
Kv	: 443 rpm/V
Eficiência	: 65%
RPM – Máx. Eficiência	: 4614
Torque - Máx. Eficiência	: 45 oz-in/A 317.8 mN-m/A
Corrente - Máx. Eficiência	: 19.8A

Dimensões		
Peso	:	(1304g)
Comprimento - Motor	:	(109.6mm)
Diâmetro	:	(66mm)
Diâmetro do Eixo	:	(8mm)
Comprimento do Eixo	:	(35.6mm)

Tabela 1 – Características elétricas do motor

TYPICAL PERFORMANCE @ 12Vdc	Torque	Speed	Current	Power	Effcy
	Oz-In	RPM +/- 10%	Amps MAX	Wo	%
Free Load	0	5310	2.7	0	0%
Normal Load	64.0	4320	27	205	63%
@ Max Efficiency	45.0	4614	19.8	154	65%
@ Max Power	171.7	2655	67.9	337	41%
@ Stall	343.4	0	133.0	0	0%
HIPOT: 600 Vac/0.5 mA/1 sec Insulation Resistance 10 MΩ Insulation Class B					

Tabela 2 – Performance do motor em diferentes situações

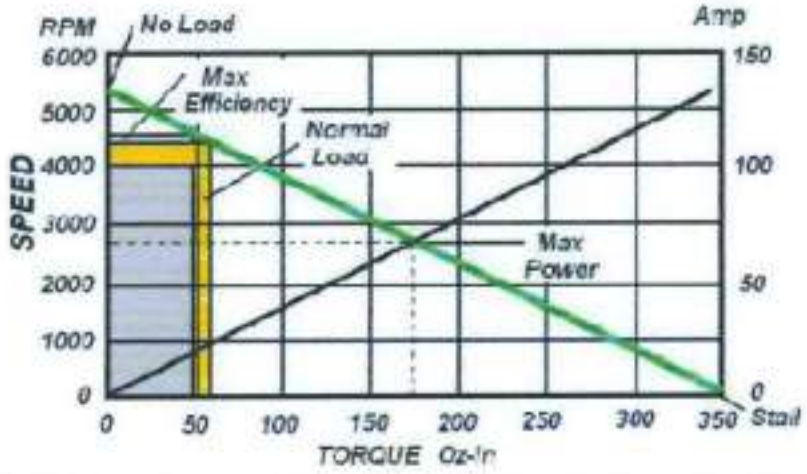


Gráfico 1 – RPM e Corrente vs. Torque (Motor CIM)

Speed Controllers "Victor 884"

Os Speed Controlers Victor 884 são componentes eletrônicos de potência, utilizam transistores de efeito de campo (FET) capazes de suportarem altas correntes, necessidade esta gerada pelos motores de Corrente Contínua que podem drenar corrente tão altas quanto 120 A, como observamos pela tabela de características de

nossos motores. O controle da velocidade é realizado através do recebimento de um sinal tipo PWM (Pulse-Wide Modulation).

Um dispositivo para controle de velocidade de um motor DC pode ou não realmente medir a velocidade com que se encontra o motor em um dado instante. Se o dispositivo realiza a medida, temos um controlador de velocidade em malha fechada (Closed Loop Speed Controller), caso contrário, a denominação que o controlador recebe é o de controlador de velocidade em malha aberta (Open Loop Speed Controller), sendo este último o caso do Victor 884.

Logo, para o controle da velocidade, o Speed Controller de malha aberta se aproveita do fato de que a velocidade de um motor DC é diretamente proporcional à tensão de suprimento, como podemos verificar através da equação fundamental que dita a Velocidade de um Motor CC:

$$N = \frac{V_a - I_a . R_a}{k . \emptyset}$$

Onde:

N – Velocidade de rotação do Motor DC.

V_a – Tensão de armadura (Controlada pelo Speed Controller)

I_a – Corrente de armadura

R_a – Resistência de armadura

k – Constante do motor

Ø – Fluxo do pólo.

Então, o Speed Controller funciona através da variação da tensão média enviada ao motor DC. Como a tensão de suprimento fornecida pela bateria é constante e vale 12 V, a variação é conseguida através do rápido chaveamento dos transistores MOSFET que compõe o circuito do Speed Controller. Se o chaveamento for rápido o suficiente, o motor será suprido com a tensão média no período.

Através do gráfico abaixo, podemos notar o efeito da variação da tensão na velocidade do motor:

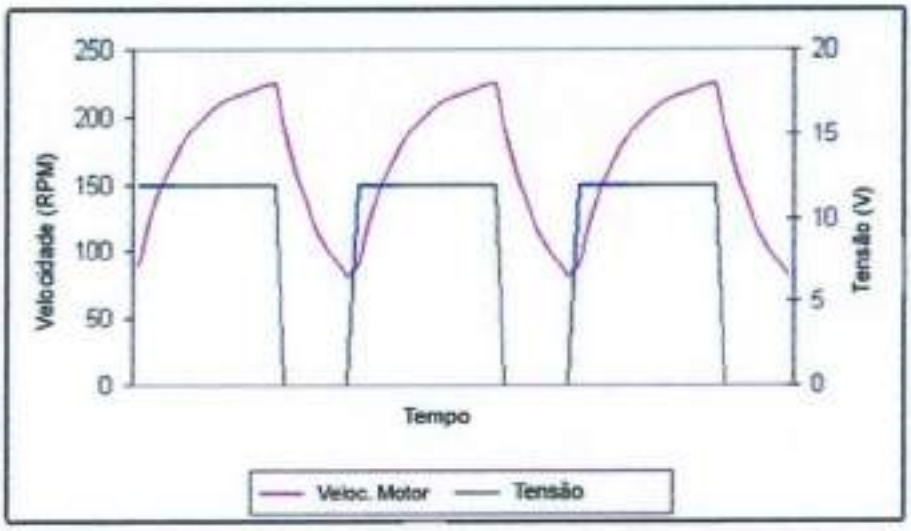


Gráfico 2 – RPM e Corrente vs. Torque (Motor CIM)

Como podemos verificar, a velocidade média é de aproximadamente 150 RPM, no entanto com uma razoável amplitude de variação em torno dela. Porém, se a tensão de suprimento for chaveada de maneira suficientemente rápida, a velocidade apresentará uma mudança mínima e poderemos considerar a velocidade praticamente constante. O sinal que controla o chaveamento é o sinal PWM, que descreveremos mais abaixo. Este é o princípio de funcionamento de um circuito controlador de velocidade.

O sinal PWM

O sinal PWM (Pulse-Width Modulation ou Modulação por Comprimento de Pulsos) recebido pelo Victor 884 deve ter o seu duty-cycle (isto é, o tempo que o sinal permanece em "nível alto" , no caso, 5 V) situado entre 1 ms e 2 ms, isto é, para um duty-cycle de 1 ms o motor estará sob regime de reversão máxima enquanto para que um duty-cycle de 2ms o motor estará com aceleração máxima à frente. Já o período que deve ser utilizado entre dois duty-cycles deve estar situado entre 15 ms e 30 ms.

Abaixo encontram-se as representações do sinal PWM para as situações de reversão máxima, motor em neutro e motor com aceleração máxima à frente:

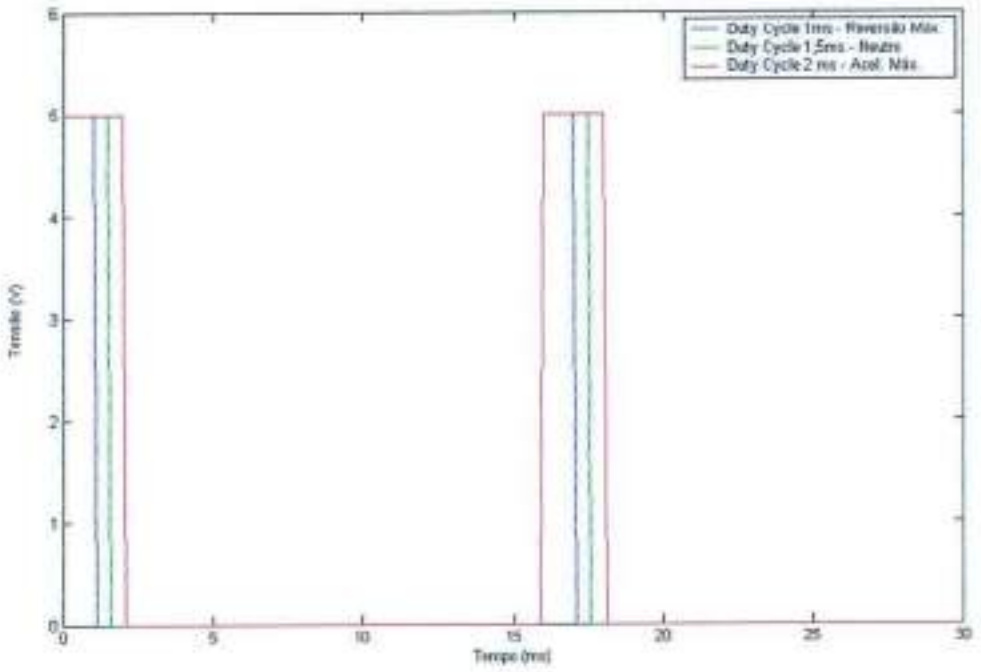


Gráfico 3 – Situações do PWM

Seguem abaixo algumas de suas especificações:

Voltagem Nominal	12V
Voltagem Min/Max	6-15V
Corrente Continua	**40A**
Corrente de Surto (2 sec)	-
Corrente de Surto (1 sec)	-
PWM Output Chop Rate	120 Hz
Potência Mínima	3%

Tabela 3 – Especificações Victor 884

Bateria de Chumbo-ácido

Para a alimentação de nosso circuito, utilizamos uma bateria de 12 V com carga de 18 A.h, modelo este tipicamente utilizado em motocicletas e pequenas embarcações marítimas. Tal carga permite que possamos utilizar o veículo por um tempo de aproximadamente 30 minutos, o que satisfaz as nossas necessidades.

A carga desta bateria está sendo realizada através de um carregador de tomada convencional, que uma corrente máxima de 900mA, o que faz com que o carregamento demore em torno de 20 horas para ser completado.

Giro-Direcional "ADXRS610"

O Giro-Direcional ADXRS610 é um componente eletrônico capaz de medir a taxa da variação angular em um dado sentido. Por estar medindo a componente de variação da posição angular, temos o sinal deste sensor sendo utilizado pela parcela derivativa de nosso controlador PID.

Para a estimação do ângulo através deste sensor, efetuamos a integração do sinal de variação angular obtido e somamos com a parcela do ângulo estimada pelo acelerômetro, tendo em vista a implementação com o filtro complementar acima descrito. A parcela da estimação do ângulo dada por este sensor possui características de alta freqüência, uma vez que é possível de se determinar variações mínimas do ângulo da plataforma e portanto este sensor será filtrado pelo filtro passa-altas implementado pelo filtro complementar.

A saída deste sensor estará ligada a uma das entradas com conversor A/D (Analógico / Digital) do microcontrolador, permitindo assim que seja feita a leitura e posterior processamento dos sinais adquiridos.

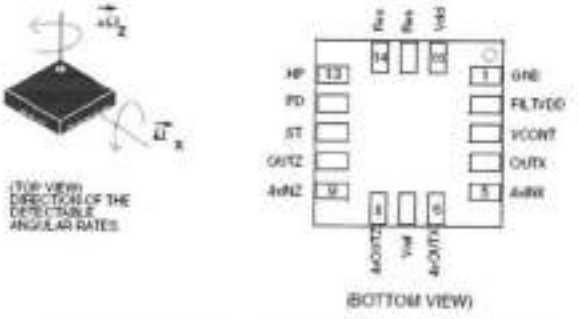


Figura 4 – Diagrama de pinagem(ADXRS610)

Acelerômetro "ADXL206 +/-1g"

Utilizaremos o acelerômetro para medir as acelerações sofridas pelo nosso veículo e a partir de tais sinais conseguiremos determinar o ângulo em que a base se encontra. Para a realização de tal cálculo, aproveitamo-nos do fato que a aceleração da gravidade sempre estará presente incidindo perpendicularmente ao solo. Tendo isto em vista, concluímos que um sensor de aceleração com dois eixos, um paralelo ao solo e o outro perpendicular ao solo devem medir respectivamente o valor de 0 e 1 g (~9,8 m/s^2) quando encontram-se em repouso. O ângulo com que o sensor encontra-se em um dado momento poderá, portanto, ser calculado através da componente da gravidade que incide no eixo paralelo ao solo quando em repouso. Portanto, o problema da determinação do ângulo reduz-se ao problema da decomposição de forças e por seguinte a relação que permite o cálculo do ângulo é a seguinte:

$$\theta = \sin^{-1}\frac{A_x}{1g}$$

Com:

θ = Ângulo de inclinação da base

Ax= Aceleração medida no eixo X (pararelo ao solo)

g = Aceleração da gravidade (~ 9,8 m/s^2)

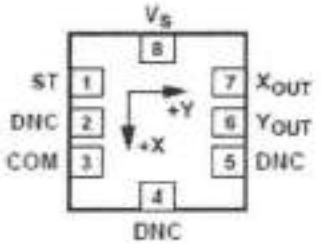


Figura 5 – Diagrama de pinagem(ADXL 203)

Transmissor e receptor ZigBee

Dada a natureza de nosso projeto, um veículo, e pela própria disposição dos componentes no mesmo, a medição de sinais utilizando instrumentos convencionais ,

como um multímetro, se torna inviável. A partir deste pressuposto, utilizaremos a interface Zig-Bee para o recebimentos dos mais diversos parâmetros que possibilitarão a análise do comportamento de nosso projeto.

Este componente tem importância fundamental em nosso projeto, uma vez que com tal módulo é possível a obtenção dos sinais do circuito de controle de maneira remota, implementando assim um sistema de telemetria em nosso veículo. Outra possibilidade advinda da utilização deste componente é a gravação *onboard* (isto é, sem a necessidade de tirar o microcontrolador do circuito) do programa no microcontrolador, através da função Bootloader, o que é bastante conveniente já que utilizando utilizando este métodoevitamos possíveis danos ao circuito integrado (principalmente quanto à quebra de pinos), uma vez que a calibração do sistema de controle é um processo empírico e exige um grande número de regravações do Firmware do Microcontrolador, o que tornaria inviável, ou pelo menos bastante trabalhoso, o processo de gravação.

Características da Comunicação ZigBee

O ZigBee é um padrão de comunicação que utiliza módulos de rádio digital de pequena potência, baseados no padrão IEEE 802.15.

O padrão IEEE 802.15 é um conjunto de especificações que define tanto o protocolo quanto o comportamento da comunicação entre rádios dentro de uma PAN (Personal Area Network). Devido ao fato de trabalhar sobre baixas razões de dados, as redes que empregam este padrão são conhecidas pelo acrônimo LR-WPAN (Low Rate – Wireless Personal Area Network).

O padrão ZigBee utiliza a frequência de operação situada na faixa dos 2,4 GHz, possuindo 16 canais com largura de banda de 5 MHz. A concepção de acesso ao canal é através do modo CSMA/CA (Carrier Sense Multiple Access/ Colision Avoidance). A velocidade de comunicação desses módulos pode chegar até 250kbps nos módulos mais avançados, podendo ser esta taxa ajustada conforme a necessidade do usuário. Em nosso projeto, estamos utilizando a velocidade de 19200 kbps, de forma que casasse com a taxa selecionada no Microcontrolador.

O alcance de um módulo ZigBee pode chegar a até 100 m , distância esta bastante apropriada para os fins de nosso projeto.

Figura 6 – Módulo ZigBee

Especificações

Abaixo, encontram-se algumas das características deste módulo:

Performance

- Rendimento da Potência de saída: 1 mW (0 dBm);
- Alcance em ambientes internos/zonas urbanas: 30m;
- Alcance de RF em linha visível para ambientes externos: 100m;
- Sensibilidade do receptor: -92 dBm;
- Freqüência de operação: ISM 2.4 GHz;
- Taxa de dados de RF: 250.000 bps;
- Taxa de dados da Interface (Data Rate): 115.200 bps;

Alimentação

- Tensão de alimentação: 2.8 à 3.4v;
- Corrente de transmissão (típico): 45 mA @ 3.3 V;

Tabela 4 – Especificações ZigBee

Programação e utilização do Módulo ZigBee

Para o correto funcionamento do módulo ZigBee, faz-se necessária a programação de seu firmware especificando o número do canal, o ID de cada módulo e o tipo de comunicação.

A baud rate escolhida foi de 19200, foram também escolhidos os canais e os IDs sem nenhum critério em especial, apenas seguindo as instruções do manual do aparelho.

O modo de comunicação escolhido foi "AT", que nada mais é que um substituto do fio na comunicação serial. Este modo, que foi escolhido em detrimento ao modo

API, tem como pressuposto o fato de que apenas haverá uma comunicação ponto-a-ponto entre dois Zigbees, como, de fato, é o caso.

Um ponto interessante a se comentar é o fato de que o ZigBee trabalha com valores de tensão de 0 a 3.3V, ao contrário do PIC que é alimentado com 5V. Para contornar este problema utilizamos um resistor na direção PIC, Zigbee. De forma que a corrente fizesse um pouco da tensão cair e então chegasse ao nível lógico correto.

A inserção do Zigbee como substituto do fio não provocou piora no sistema de aquisição e os dados, em sua maioria, chegaram de forma adequada. Não obstante, alguns pacotes não chegam na ordem esperada, o que não comprometeu o resultado final.

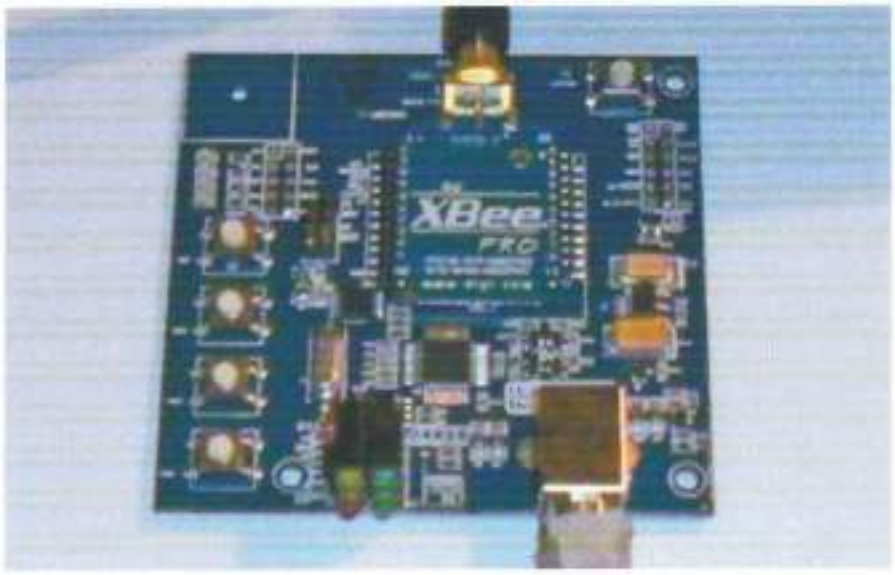


Figura 7 – Módulo Xbee de comunicação USB

Microcontrolador

Através do microcontrolador, efetuaremos o processamento dos sinais providos dos sensores e a partir deles conseguiremos originar o sinal de controle, que deverá ser modulado em PWM para que seja enviado aos Speed Controllers, que por sua vez ditarão o funcionamento dos motores. Utilizamos um microcontrolador PIC devido a nossa maior familiaridade com esta linha. Optamos também por um modelo que possui encapsulamento DIP, uma vez que os modelos com encapsulamento SMD tornariam o trabalho de solda muito difícil, exigindo até mesmo a utilização de técnicas e equipamentos sofisticados.

Para a escolha do modelo do Microcontrolador PIC, as seguintes características deveriam ser satisfeitas:

- Possuir, no mínimo, três canais de entrada com conversor Analógico Digital (A/D), para receber os sinais do Acelerômetro, Giro Direcional e do potenciômetro do Stick para a realização de curvas.
- Ter suporte à interface USART, compatível com o protocolo RS232 para a comunicação serial.
- Possuir ao menos 15 portas I/O,.
- Encapsulamento DIP.

Tendo todas as características acima em mente, e verificando a disponibilidade para a compra do circuito integrado, optamos pelo modelo PIC 18F4331, que se adapta perfeitamente às nossas necessidades.

PIC 18F4331

O microcontrolador PIC 18F4331 utiliza uma arquitetura em 8 bits, possui memória flash com 8 Kbytes destinados à gravação do Firmware e pode operar com clock máximo de 40 MHZ, características essas que superam as exigências de nosso projeto.

O seu encapsulamento é do tipo DIP, possuindo 40 pinos, sendo que 36 destes são de portas I/O.

Tem ainda como características a presença de 9 canais com conversores A/D, um módulo USART compatível com o protocolo RS232, oscilador interno com 8 frequências selecionáveis, além de suportar o uso de osciladores externos,como utilizado em nosso circuito, com um oscilador de cristal de 20 MHZ.

Para a gravação do Firmware de nosso Microcontrolador, utilizamos como hardware a ferramenta de desenvolvimento EasyPIC 3 da Mikroeletronika, plataforma esta que também serviu de base para os nossos testes iniciais, uma vez que possui potenciômetros já acoplados às portas do Microcontrolador, o que nos permitiu simular a saída dos sensores e verificar se as respostas do sistema de controle estavam condizentes.

Figura 8 – Placa de desenvolvimento EasyPIC

Já o Software que utilizamos para a gravação do Firmware é o PICFlash, que utiliza o compilador da MikroC , sendo os dados transmitidos através de uma porta USB ligada à placa de desenvolvimento.

Segue abaixo o diagrama de pinagem do PIC 18F4331:

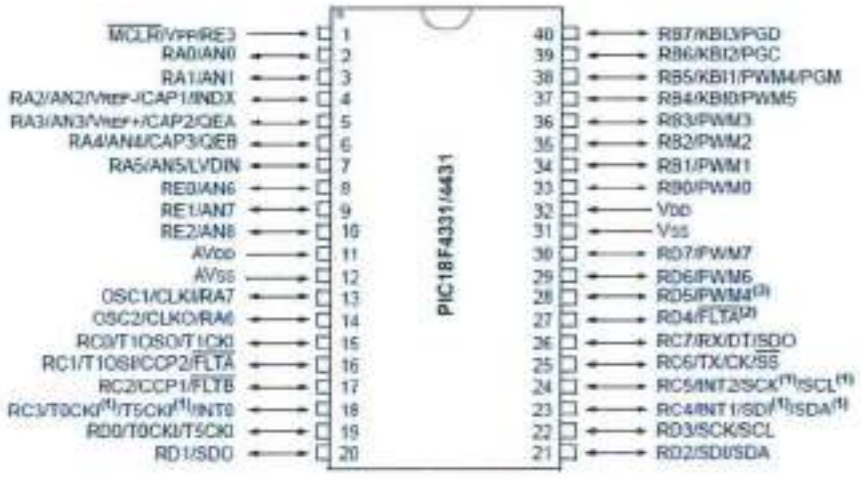


Figura 9 – Diagrama de pinagem(PIC 18F4331)

Circuito Eletrônico

Os testes iniciais de nosso protótipo foram realizados utilizando-se placas protoboard, de modo a facilitar as mudanças, que são freqüentes na etapa de concepção do circuito, até que chegamos no circuito consolidado e assim pudemos prosseguir com a confecção do circuito impresso. Destacamos porém a dificuldade em se testar o veículo utilizando aquela disposição, já que o mesmo sujeita os componentes a uma grande vibração e movimentos bruscos, o que ocasionou muitas vezes o surgimento de mau contatos além da desconexão de fios. Com esta experiência verificamos o quão importante seria a realização de um circuito em uma placa de circuito impresso, tomando uma precaução extra com a fixação dos componentes, levando em conta os motivos acima citados.

Tomamos a decisão de nós mesmos fazermos a placa de circuito impresso, pois assim conseguiríamos uma boa economia e também colocaríamos em prática os conceitos aprendidos em algumas das disciplinas do curso.

Portanto, prosseguimos com a realização do esquemático do circuito, utilizando o software *Eagle*, no qual já tínhamos experiência por termos executado projetos em outras disciplinas. O resultado do esquemático encontra-se na figura abaixo:

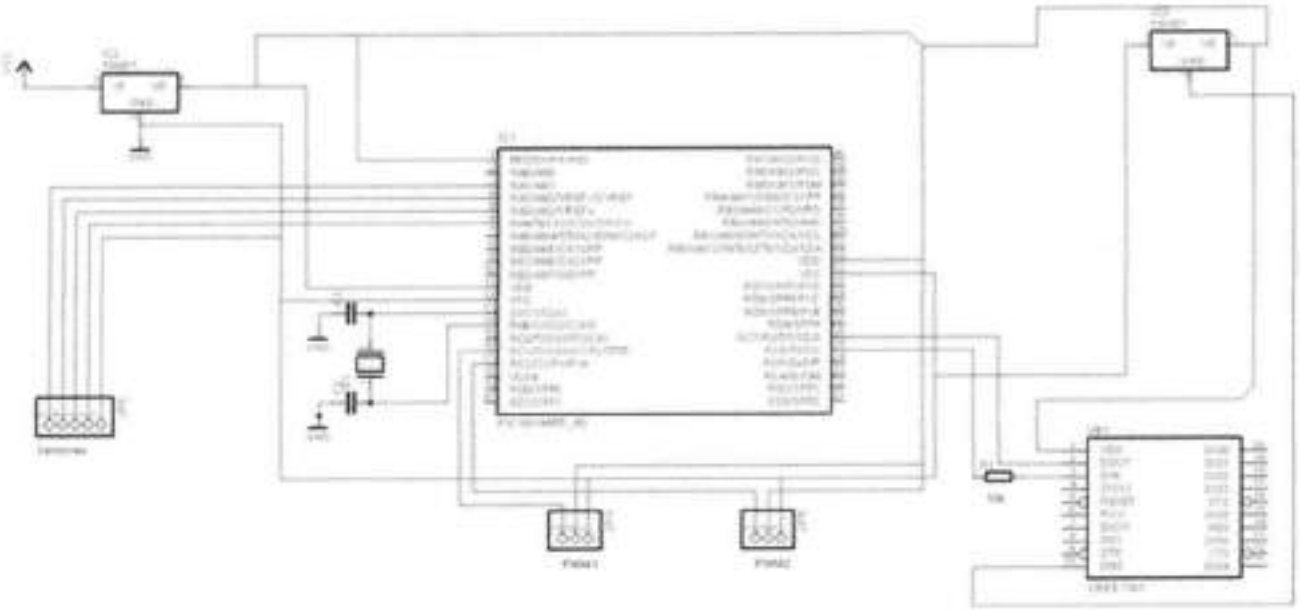


Figura 10 – Circuito eletrônico de controle

Os seguintes componentes formam o circuito eletrônico:

- Capacitor Eletrolítico de 4700 µF: O capacitor foi colocado na entrada da alimentação do regulador de Tensão de 5 V, com o objetivo de filtrar possíveis interferências do circuito elétrico, já que durante o funcionamento do veículo prevemos possíveis variações na tensão dados os diferentes regimes de funcionamento dos motores, que certamente trariam ruídos indesejados ao circuito eletrônico. A escolha da capacitância foi tomada optando-se por um valor bastante elevado, tentando garantir o bom funcionamento do filtro elementar.

- Polyswitch 1 A : Um Polyswitch é o componente que atua na proteção do circuito, garantindo que em caso de sobrecarga devido a um curto circuito ou mal funcionamento, o circuito eletrônico não seja submetido a altas correntes que poderiam causar danos aos componentes eletrônicos. Um Polyswitch é um componente de proteção térmico resetável, isto é, ao atingir a corrente limítrofe, o dispositivo atua como se estivesse abrindo o circuito, devido a um grande aumento na resistência, causado pelo aumento na temperatura, protegendo assim contra correntes excessivas. Tão logo o surto de corrente cessa, este dispositivo diminui sua temperatura e a resistência diminui fortemente, voltando ao regime normal de condução. Dada essa característica de "abertura" e "fechamento" do circuito, é denominado resetável.

- Regulador de tensão LM7805 5V: O regulador de tensão tem a função de converter a tensão de entrada em uma tensão pré-determinada, além de garantir a regulação da tensão, isto é, a sua constância ao longo do tempo. Em nosso circuito eletrônico, alimentamos o regulador de 5V com a tensão provida diretamente da bateria, de 12 V, porém no estágio posterior ao filtro implementado pelo capacitor de 4700 µF, o que garante que o componente receba uma tensão mais uniforme possível. A tensão de 12 V na entrada do regulador é adequada, uma vez que pelas características elétricas constantes no datasheet, verificamos que a tensão máxima de entrada é de 35 V. Ainda, a utilização desse componente é de fundamental importância, já que é compatível com os limites de tensão de entrada do Microcontrolador, do Acelerômetro e do Giro-Direcional

- Regulador de Tensão LM 1117 T 3,3V: Utilizamos também um outro regulador de tensão em nosso circuito, desta vez um dispositivo que fornece a tensão de 3,3 V na saída, tensão essa necessária para a alimentação do módulo XBee, que possui tensão máxima de entrada de 3,4 V, como já referenciado acima nas especificações deste componente.

No circuito, alimentamos o regulador de saída 3,3 V com a tesão provida pelo regulador de tensão de 5V , de forma que garantimos com esta topologia uma tensão de entrada uniforme.

- Módulo XBee: O módulo XBee (ZigBee) foi alimentado com a tensão de 3,3 V provida do regulador. Quanto à interface para a comunicação, utilizamos as portas DOUT e DIN, que são as portas que contam com o controlador UART (*Universal Asynchronous Receiver/Transmiter*). Como o módulo ZigBee possui uma tensão máxima permitida de 3,4 V e o Microcontrolador PIC funciona com tensão 5V para nível lógico "1", tivemos que adicionar um resistor entre a entrada DIN do resistor e o pino de saída da interface do PIC, para que conseguíssemos abaixar a tensão um valor adequado.

- Cristal de 20 MHz: Utilizamos um oscilador externo conectado ao Microcontrolador PIC. Estão ligados a este Oscilador dois capacitores de 15pF, configuração esta típica de um Oscilador e prevista no datasheet.

- Microcontrolador PIC18F4331: Como já dito anteriormente, o PIC foi alimentado com a tensão de 5 V provida do regulador de tensão. Conectamos as saídas dos sensores (Acelerômetro e Giro-Direcional) bem como a saída do potenciômetro, utilizado para a realização das curvas, em entradas providas de conversor A/D. A saída e entrada USART foi ligada com o módulo ZigBee para a realização da comunicação serial. Como já dito anteriormente, utilizamos um Oscilador externo e o conectamos na entrada apropriada para este afim. Ainda, temos 6 pinos destinados à saída do sinal PWM aos Speed Controllers Victor 884, sendo 3 pinos para cada controlador. Por fim, ainda temos umachave ligada a uma das portas que tem a função de "ligar" e "desligar" os Motores. Implementamos a desativação dos motores através da transmissão do sinal PWM "neutro" ao Speed Controller.

- Acelerômetro: O Acelerômetro juntamente com o Giro-Direcional foram colocados em uma placa de circuito impresso de trilhas disposta perpendicularmente à placa de circuito impresso com os outros

componentes, disposição esta devido à necessidade de alinhar um dos eixos do Acelerômetro com o solo (na situação da base com inclinação 0°), sendo que o outro eixo deveria ser perpendicular e portanto mediria a aceleração da gravidade. O Acelerômetro foi alimentado com tensão de 5V e a sua saída, analógica, foi ligada a uma das entradas providas de conversor A/D no PIC.

- Giro-Direcional: Como já explicado anteriormente, o Giro-Direcional foi disposto na placa de trilhas perpendicular ao circuito impresso dos outros componentes. Com essa disposição garantimos que este sensor mediria a razão angular em relação ao eixo transversal de nosso veículo.

- Potenciômetro Stick: Utilizamos um potenciômetro do tipo stick, com um dispositivo mecânico que garante o seu retorno à posição central quandonão manipulado. Tal potenciômetro foi extraído de um controle de vídeo game Playstation 2. A saída deste potenciômetro foi ligada a uma das entradas providas de conversor A/D, com o fim de se implementar o sistema de curvas do veículo.

- Chave: Utilizamos uma chave para podermos ligar e desligar os motores, método este conseguido através da manipulação do sinal PWM enviado aos Speed Controllers.

Uma vez realizado o esquemático interligando todos os componentes acima descritos, prosseguimos com a confecção dos arquivos "Board", que consistem do desenho das trilhas a serem utilizados na impressão do circuito na placa de fenolite, ainda toda coberta pela camada de cobre. Visto o grande número de ligações entre os componentes do circuito e a necessidade de criarmos uma placa com dimensões compactas, para que fosse compatível com as dimensões de nosso veículo, optamos por fazer a placa usando a técnica de duas camadas (*double layer*), o que permitiu que conseguíssemos uma boa disposição das trilhas sem enfrentar problemas de trilhas cruzadas no roteamento.

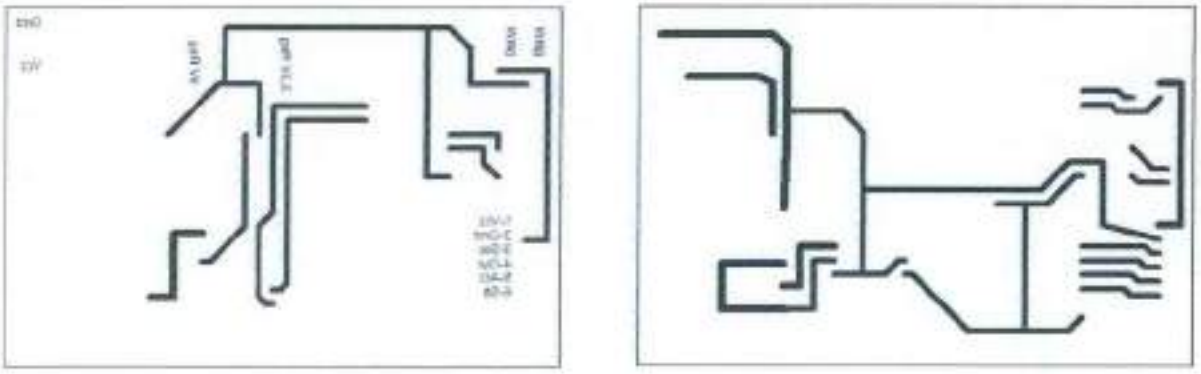


Figura 11 – Placa de circuito impresso (Controle)

Com as trilhas já impressas no papel Transfer, utilizamos a máquina de Transfer para que as mesma fossem transferidas para a placa de Fenolite e após este processo iniciamos a corrosão da placa em solução contendo Percloreto de Ferro.

As fotos abaixo mostram a placa com os componentes já devidamente instalada no nosso protótipo:

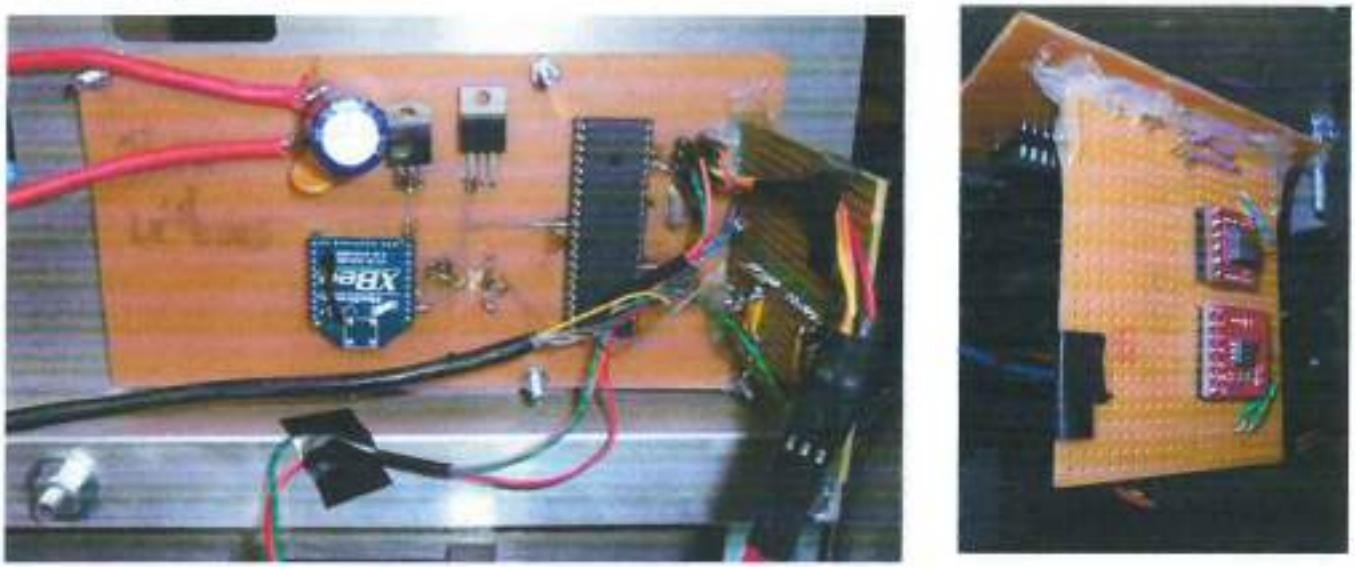


Figura 12 – Placa de circuito impresso (Controle) e placa com sensores, já instaladas no protótipo.

Circuito Elétrico

O circuito elétrico de nosso veículo é composto pela bateria de 12 V, pelos fusíveis de proteção, pelos Speed Controllers e por fim pelos dois motores elétricos de corrente contínua.

A imagem abaixo mostra a configuração de nosso circuito elétrico:

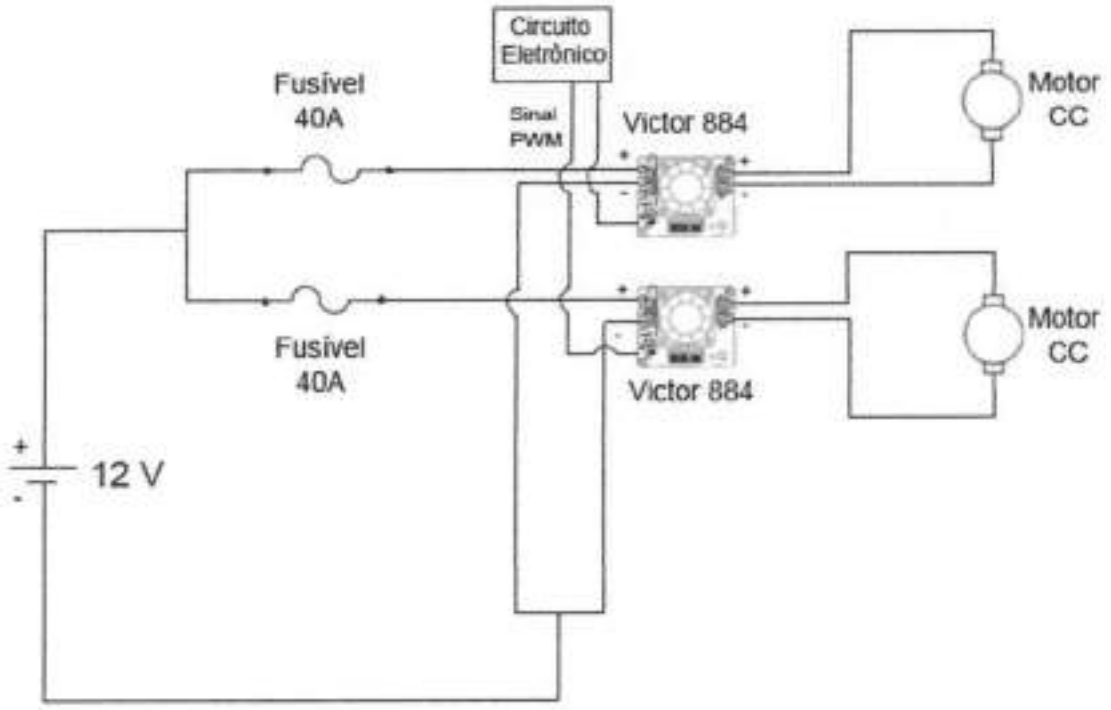


Figura 13 – Circuito de potência

Os fusíveis utilizados são fusíveis de ação retardada, isto é, permitem que uma corrente maior que a corrente de desarme passe pelo circuito por um curto intervalo de tempo. Isto é bastante apropriado na em circuito contendo motores elétricos, já que durante a utilização destes é prevista uma sobrecarga durante curtos períodos de tempo na partida e em situações de extrema carga. A corrente de desarme destes fusíveis é de 40 A.

Abaixo encontra-se a curva característica deste fusível:

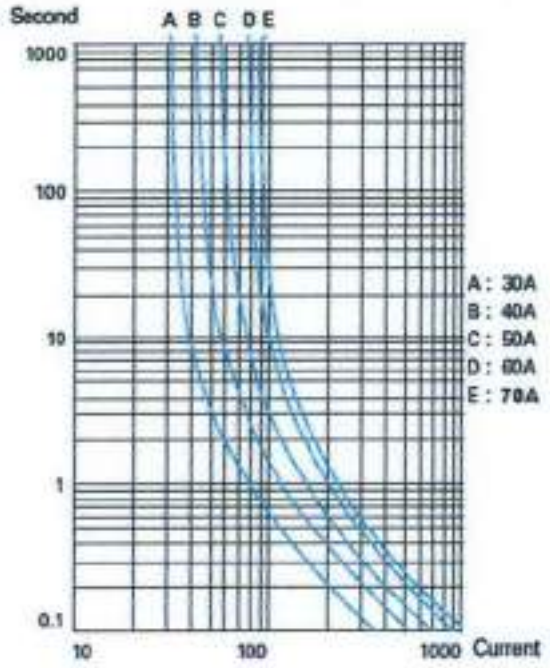


Gráfico 4 – Curva I-T do fusível

Como podemos notar do gráfico, este fusível pode suportar uma corrente de até aproximadamente 110 A por até 1 segundo, o que é válido já que temos uma corrente de *Stall* do motor valendo 133 A.

Já, a escolha dos fios se deu pela máxima corrente que previamos em condição de operação normal, valor este de 80 A, limitado pela corrente de operação normal máxima dos *Speed Controllers*.Com isso,tendo a norma NBR5410como referência,que regula instalações elétrica de baixa tensão, decidimos por utilizar um fio de cobre de secção nominal de 16 mm^2 no segmento ligando a bateria até o ponto de divisão da distribuição para os dois *Speed Controllers* Victor 884/ Motores CIM, como pode ser visto no diagrama do circuito elétrico acima. Já para o segmento possuindo os dois fios em paralelo que levam até os *Speed Controllers*, utilizamos fios de secção nominal de 10 mm^2. Consultando a tabela 28 da norma NBR 5410, e tendo em vista que o método de instalação que mais se aproxima do utilizado em nosso protótipo é o 4F, que usa fios unipolares encostados, verificamos que o valor nominal para o fio de 16 mm^2 é de 99 A e o valor nominal para o fio de 10 mm^2 é de 73 A. Como se pode notar tivemos como opção superdimensionar os fios de maneira que não encontrássemos problemas de aquecimentoem situações de cargas elevadas.

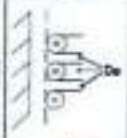
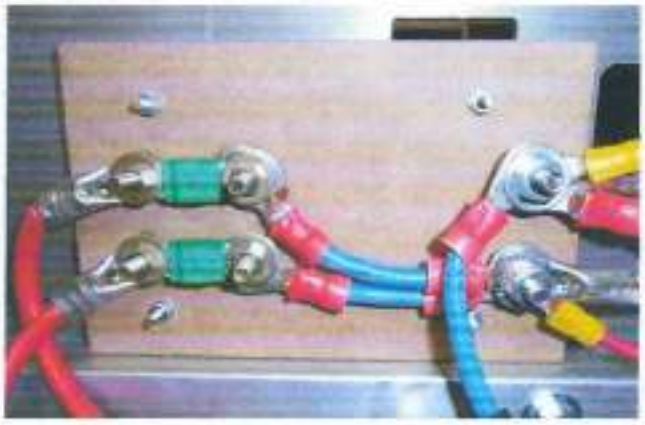
Seções nominais mm²	Métodos de instalação definidos na tabela 28						
	E	E	F	F	F	G	G
							
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Cobre							
0,5	11	9	11	8	9	12	10
0,75	14	12	14	11	11	16	13
1	17	14	17	13	14	19	16
1,5	22	18,5	22	17	18	24	21
2,5	30	25	31	24	25	34	29
4	40	34	41	33	34	45	39
6	51	43	53	43	45	59	51
10	70	60	73	60	63	81	71
16	94	80	99	82	85	110	97

Tabela 5 – Corrente máxima vs. Seção do fio (Bitola)

Seguem abaixo algumas fotos do circuito elétrico:



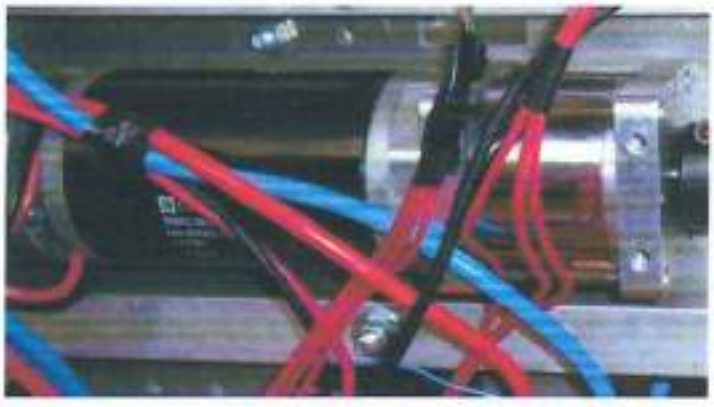


Figura 14 – Circuito elétrico

2.2 Componentes mecânicos

O nosso projeto mecânico foi baseado fortemente no projeto do MIT, referência bibliográfica [1]. Tal decisão foi tomada com o intuito de simplificarmos a tomada de decisão referente a esta parte, nos poupando tempo para que pudéssemos nos dedicar ao projeto elétrico/eletrônico, que o foco deste projeto.

Segue abaixo um descritivo dos componentes utilizados:

Rodas com eixo acoplado

As rodas que utilizamos em nosso projeto possuem dimensão de 12". Consistem de rodas de carrinho de mão que foram adaptadas para receberem o eixo compatível com o acoplador e rolamentos de fixação presentes em nossa estrutura mecânica.

Figura 15 – Rodas de 12" juntamente com o eixo.

Caixa de Redução

Utilizamos uma caixa de redução ligada ao eixo do motor com o objetivo de realizar a transformação mecânica de velocidade do motor em torque. Em nosso projeto utilizaremos uma redução 1:16.

Com tal redução conseguimos uma velocidade máxima, para a condição sem carga:

Usamos rodas de diâmetro 300mm (12"), logo:

$V_{max} = RPS.r.C_r = \frac{5280}{60} * \frac{1}{16}$ *0,30*π = 5,18 m/s = 18,66km/h

Sendo:

RPS – Velocidade Rotacional [Hz]

r – Razão de redução de rotação

C_r – Comprimento da roda

Figura 16 – Redução 16:1.

Chassi

Para a confecção do chassi, composto pela base e por uma haste semelhante a um guidão, utilizamos uma liga de Alumínio de alta resistência, o que garante leveza e robustez ao nosso veículo. Inicialmente planejávamos utilizar a liga 6061, por ser uma liga com resistência adequada a aplicações que exijam grandes esforços, sendo esta mesma utilizada na produção de bicicletas e pequenos veículos mas,dada a impossibilidade de acharmos tal liga em forma de chapas, utilizamos a liga 7075, que possui resistência mecânica ainda mais elevada que a liga 6061, sendo utilizada em uma ampla gama de produtos, que incluem até mesmo a produção de peças aeronáuticas.

A chapa de Alumínio que compramos possuia as dimensões 600x600x7 mm. A densidade desta liga é de 2,81 g/cm^3.Logo, com um volume de 2520 cm^3 tínhamos uma massa de 7,081 kg.

Corte e modelagem

Para a confecção da base de Alumínio, decidimos por utilizar o método de corte por jato d'água, que consiste na realização do corte por uma máquina que utiliza jato d'água em alta pressão. Tal máquina faz referência a um arquivo CAD em 2 dimensões que contém todas as medidas e entidades geométricas que ditarão o seu funcionamento,de forma que ao final do processo obtenhamos o design desejado.

O arquivo CAD de 2 dimensões que utilizamos foi o mesmo disponibilizado no projeto do MIT[1].

A edição deste arquivo foi realizada no *Software SolidWorks*, a partir do qual obteve-se um arquivo de extensão "dwg", compatível com a máquina de corte.

O corte foi realizado na empresa JetCorte, especializada neste método.

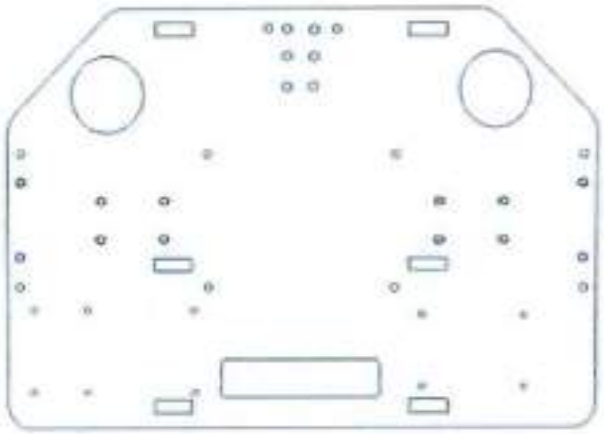


Figura17 – Representação do arquivo CAD*[1]* utilizado no Corte

Acopladores

Utilizamos um acoplador para interligarmos o eixo da redução ao eixo ligado nas rodas. Essa peça possui a função, além de obviamente interligar os eixos, de prover também um amortecimento quando há aceleração no motor, suavizando variações bruscas.

Figura 18 – Acoplador

Suporte com Rolamento

Os suporte com rolamento tem importância fundamental na estrutura mecânica do protótipo. Eles encontram-se fixados na extremidade da base, junto às rodas e tem a função de suportar grande parte das cargas impostas pelo usuário na estrutura, além de garantir o correto alinhamento do eixo com o segmento Acoplador – Redução –

Motor, alinhamento este importantíssimo para garantir que os componentes não sejam danificados, particularmente a redução que possui limitações nesse quesito.

Figura 19 – Suporte com rolamento

Estrutura Mecânica Consolidada

A estrutura mecânica consolidada juntamente com a baterias, e todos os componentes do circuito ficou com massa final de 24,20 kg.

O desenho 3 visões com todas as dimensões encontra-se nos Apêndices.

Abaixo encontram-se algumas fotos do protótipo já consolidado:

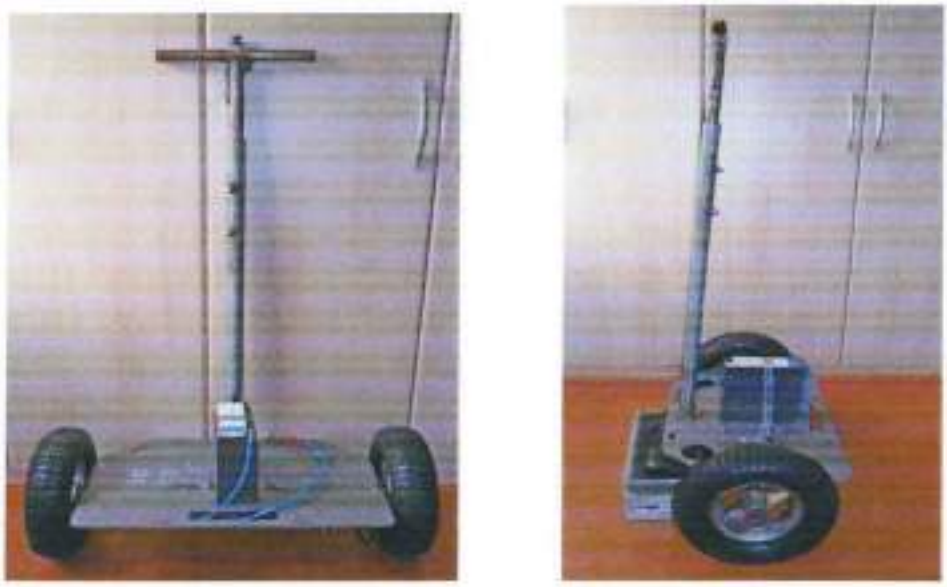


Figura 20 – Visão lateral e frontal do veículo

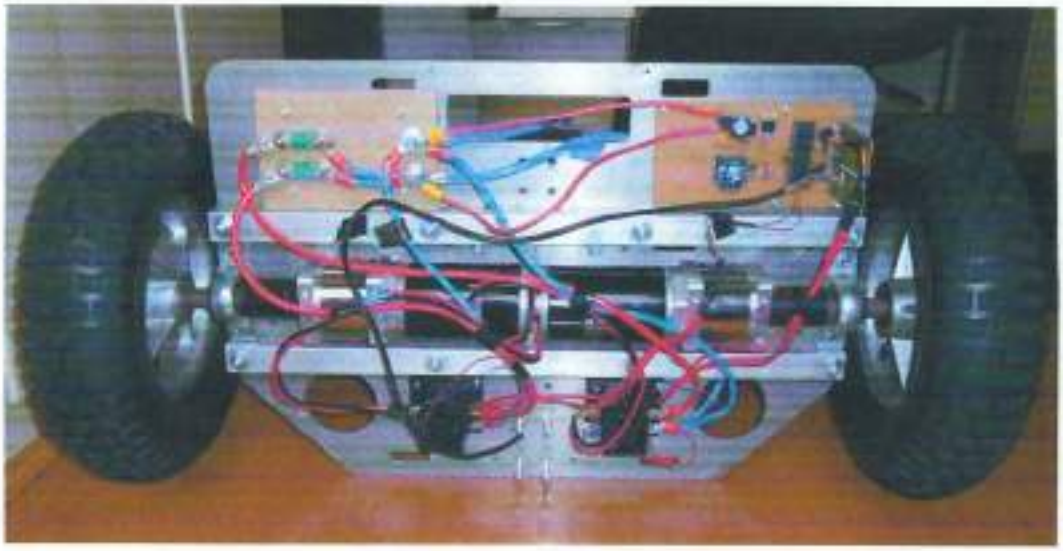


Figura 21 – Visão inferior do veiculo

2.3 Firmware de controle

Como referência para o desenvolvimento do código de controle, foram utilizados conceitos presentes em [1] e [8].

Partindo da análise do problema em conjunto com as referências, concluiu-se que o ideal seria fazer a leitura de duas portas analógicas que continham os sensores, aplicar o filtro complementar e então aplicar a equação geral do controlador PID.

A parte supracitada, em conjunto com a escala dos valores de sensor encontra-se no código a seguir:

```
acelerometro= ADC_Get_Sample(2);
giroscopio = ADC_Get_Sample(3);
stick = ADC_Get_Sample(1);
acelerometro_graus = (acelerometro - 514) * 0.4375;
stick_curva = (stick  - 512) * 0.4375;
giro_velocidade = (giroscopio - 515) * 0.34;
```

Não há grandes dificuldades no entendimento do código acima. Apenas foram feitas aferições das portas analógicas e transformadas em unidades físicas de acordo com a descrição dos respectivos manuais.

A seguir, o ângulo em que o veículo se encontrava foi estimado através da aplicação do já discutido filtro complementar.

```
angulo = 0.996 *(angulo + giro_velocidade * dt)+0.004 *(acelerometro_graus);
```

É interessante notar que a velocidade do giroscópio é integrada pela multiplicação por dt. A variável dt foi estimada através de medidas acerca da velocidade em que o código rodou. Após extensivos testes, chegamos ao valor de 0.001, ou seja, o código principal de controle completava um looping a cada 0.001 segundos.

De posse do ângulo em que se encontra o veículo, basta aplicar a equação básica do controle:

```
motor += ((KP * angulo) + (KD * giro_velocidade);
```

Cabe apenas um comentário a respeito da equação acima. Não foi utilizada a parcela relativa a integração do ângulo. Após repetidos testes, notou-se que a contribuição dessa parcela não era significativa para o funcionamento do sistema. Causando até mesmo mais dificuldade no equilíbrio. Portanto, não foi utilizado um PID propriamente dito, e sim, um PD.

Após essas manipulações dos resultados dos sensores, garantimos também que o valor da variável motor teria a variação compreendida entre -512 e +512.

Após a aplicação do controle para a variável motor, esta foi espelhada para duas variáveis distintas, relativas ao motor direito e ao esquerdo, que variam de 0 a 1024.Essa imposição ficará clara mais adiante quando serão comentados os sinais PWM.

Tais sinais (PWM) podem ser gerados em um PIC de duas maneiras distintas. A primeira utiliza como base uma função específica do compilador em que é possível alterar a frequência base e o duty cycle do pwm de forma simplificada, através de funções próprias. A segunda consiste na utilização de temporizadores cuidadosamente calculados combinados com interrupções de overflow. No presente trabalho, foi utilizada a segunda maneira, que encontra-se apresentada a seguir.

Primeiro, segue a parte relativa a interrupção do temporizador dos motores:

```
if (PIR1.TMR1IF) { /*Conta ate 65535 contador de 16 bits*/

TMR1H = 15535 >> 8;
```

```
                TMR1L = 15535 & 0xFF;
                PIR1.TMR1IF=0;
if(pwm==0)
                {
                PORTC.RC1 = 1;
                PORTC.RC2 = 1;
pwm=1;
                }
                if(pwm==1)
                {
                pwm=0;
                }
        }
```

Como TMR1 é um contador de 16 bits, assim que for atingido o valor de 65535, essa interrupção será chamada. Como primeiro passo, a variável TMR1 será zerada em 15535, ou seja, o contador vai contar 50000 vezes, o que representa 10ms, já que o chip roda a 20MHz (O que significa 5MHz para os contadores). Porém, o sinal PWM aceito pelo Victor é especificado como tendo 20ms. Para contornar a situação, foi adicionada uma variável auxiliar (pwm) que permite que o PWM seja colocado em nível positivo apenas uma vez a cada dois ciclos. Gerando assim um sinal de 20ms.

Também há uma segunda fase de interrupção, para deixar RC1 e RC2 em nível baixo, completando assim o sinal PWM.

```
if (PIR1.CCP1IF) { /
PORTC.RC1 = 0;
PIR1.CCP1IF = 0;
        }
```

É nas variáveis relativas a essa interrupção que a variável de cada motor vai trabalhar, no sentido de alterar o tamanho do pulso em 1. Essa interrupção é chamada quando o contador atingir o valor especificado em CCPR1.

Esse valor é especificado de acordo com o seguinte código:

```
CCPR1 = (20535 + motor_esq*5);
```

Ou seja, quando o motor for igual a zero. O duty cycle do pwm será de 5000 ciclos de clock, ou seja, 4ms.

Isso conclui a parte de controle do presente trabalho.

Juntamente com o código de controle, há o código de comunicação serial. O principal propósito dessa parte é depurar o sistema e assim conseguir fazer um bom ajuste das constantes.

A parte de comunicação serial funciona em conjunto com o software de comunicação, que, por sua vez, roda em um computador com Windows. A comunicação funciona por polling, ou seja, a cada vez que o Software de comunicação requer algum dado o microcontrolador envia através da serial e no máximo um dado por passagem no looping.

Abaixo segue a parte relativa à comunicação serial do valor atual do acelerômetro, meramente a título exemplificativo, já que há a mesma estrutura de código para as outras variáveis:

```
if (UART1_Data_Ready())
     {
        UART1_Read_Text(text,"#@",255);
(...)
if(strncmp(text,"accel",5)==0)
        {
temp_res = acelerometro;
                        UART1_Write('a');
        UART1_Write((temp_res>>5));
        UART1_Write((temp_res&0b00011111));
}
```

O protocolo de comunicação é o seguinte. O Software de comunicação envia uma mensagem do tipo "sensor#@". Quando o código do microcontrolador detecta uma mensagem terminada com #@, é feita a comparação do começo da mensagem com uma série de possibilidades (No caso acima, com 'accel'). Caso a comparação retorne verdadeiro, o PIC envia um caractere de confirmação e depois os dados. No caso do acelerômetro, o PIC envia a letra 'a' e depois o dado relativo ao

acelerômetro. Esse envio da letra permite que o Software de monitoramento saiba a variável na qual armazenar o dado recebido e evita erros de transmissão.

No que tange a velocidade do loop principal de controle, medições foram feitas. Através de um led intermitente, que ligava e desligava com um milésimo da velocidade do looping principal, foi feita uma aferição com um cronômetro comum para se obter uma medida aproximada da velocidade em que o código corre.

O resultado para tal medida foi de cerca de um segundo, ou seja, o looping de controle roda a cerca de 1000Hz. Ressalta-se que este procedimento foi feito logo nos primeiros testes com o programa.

Chegando a esse valor, consultamos a biografia para verificar se era ou não suficiente. Comparando com o resultado de [1], se constatou que a velocidade era mais que suficiente, já que o trabalho da biografia rodava a apenas 100Hz e obteve resultados satisfatórios.

Resta agora analisar o código do Software de monitoramento e comunicação, que será feito no próximo item.

2.4Software de Comunicação

Como já explicado anteriormente, dada a natureza do veículo e a impossibilidade/dificuldade em se realizar medições dos parâmetros do circuito, optamos por implementar um sistema de telemetria utilizando um rádio trasmissor Zig-Bee, que recebe e envia dados de acorda com as solicitações do software implementado no computador.

O software foi implementado utilizando-se a linguagem de programação C# e o protocolo Serial (RS232) para a comunicação externa. A vantagem da utilização do nosso módulo Zig-Bee é que o mesmo possui um modo "transparente", o que significa que a comunicação se dá da mesma forma que seria executada caso estivéssemos utilizando um fio acoplado à saída serial, sendo portanto desnecessário que tenhamos que manipular e lidar com um novo protocolo.

A imagem a seguir apresenta uma *screenshot* mostrando a interface de nosso Software:

Figura 22 – Interface de telemetria

Como é possível verificar da imagem, o Software foi concebido para apresentar informações sobre os parâmetros do circuito e também do sistema de controle, tais como tensão dos sensores, ângulo estimado, etc.

O código do programa é extenso e não está no escopo deste relatório descrevê-lo em seus pormenores.

Não obstante, serão comentadas a seguir as partes mais interessantes do código e do desenvolvimento da interface.

Primeiramente, com foco no que tange a interface gráfica serão explicitadas as principais funções do programa. Há quatro gráficos na janela principal. Cada um deles corresponde a saída de um sensor e o último corresponde à variável relativa ao motor direito. Também está presente na interface o ângulo em que a unidade se encontra, bem como opções da conexão serial (Porta, *baud rate*, paridade e número de bits).

O código pode ser dividido em duas grandes frentes, a conexão serial e o protocolo de mensagens.

A conexão serial foi feita com base a classe Communication Manager [11].Essa classe trouxe uma abstração na conexão com a porta serial em C#.

Uma vez estabelecida uma conexão com a porta serial, era necessário desenvolver o protocolo de mensagens do software com o microcontrolador do veículo.

O protocolo, por sua vez, como já descrito anteriormente, funciona por polling. A cada 50ms um timer é ativado no programa e faz a requisição de um certo dado pela porta serial. O PIC responde com uma mensagem de 'ack', que consiste em uma letra previamente escolhida. Após o caractere, são enviados dois bytes que contém efetivamente o dado a ser transmitido.

Abaixo segue um trecho, a título exemplificativo do código que implementa o protocolo supracitado (Os comentários explicam o código):

1)Envio da mensagem para o PIC

```
case 4: /*Uma mensagem por looping*/
            if (checkBox4.Checked) /*Se o monitoramento do motor esta
ligado*/
{
comm.WriteData("rmotor#@");
                this.progressBar2.Value = comm._motord.Value;
this.progressBar2.Refresh();
            }
poll ++;
break;
```

2)Após o recebimento do ack ('r', para o motor direito)

```
case 'r':

  DisplayData(MessageType.Incoming, "Motor direito \n");
(...)

      rmotor = (int)((uint)msg[1] << 5);  /*remonta dado a partir dos
dois bytes*/
rmotor = rmotor | (int)((uint)msg[2]);/*remonta dado a partir dos dois
bytes*/

motord.Value = rmotor; /*Atualiza o valor da barra de progressao*/

}
```

```
/*O codigo a seguir monta a serie a ser mostrada no grafico*/
                                    motor_d[amostramo] = _rmotor;
                                    amostramo++;
                                    if (amostramo == 100)
                                    { amostramo = 0; }
                                    Grafico3.RemoveSeries("Motor");

    Grafico3.AddSeries("Motor", Color.Lime, 1.5f, false, true,
TickMark.Dot, accel);
```

Após esse pequeno protocolo de envio, o dado é remontado no software e então exibido na tela do usuário.

Por fim, cabe comentar acerca da geração dos gráficos. Para que os dados pudessem ser exibidos na forma de gráfico foi necessário um componente de Visual Studio chamado SeaBeckEnterprises [10] que proveu as funcionalidades necessárias para a criação de gráficos

2.5 Bootloader

Em conjunto com o sistema de monitoramento, fez-se necessária a utilização de algum meio que permitisse a re-gravação do firmware do veículo; de forma simplificada e, de preferência, sem fio.

Após inúmeras pesquisas, chegou-se a conclusão que o melhor método para cumprir os requisitos seria o Bootloader.

O Bootloader é um pequeno software executado antes do código principal. Desenvolvido em assembly, ocupa pouco espaço na memória e tem total controle sobre o hardware, não constituindo, assim,nenhum entrave à sua implementação.

Levando em conta que o desenvolvimento de tal software fugiria ao escopo deste projeto, foi decidido que seria utilizada uma solução já consolidada e amplamente testada.

Novamente, após procura, foi encontrado o Tiny Bootloader [12]. O suporte oferecido por tal software incluía o nosso modelo de PIC, exigindo apenas algumas alterações em seu código (Apêndice D).

A comunicação com o Tiny Bootloader é feita através da porta serial. Ou seja, em conjunto com o Zigbee seria possível que o sistema de gravação de Firmware fosse todo feito de forma Wireless.

Para que a gravação seja feita, basta seguir os seguintes passos:

1-Desligar o PIC no veículo.

2-Pressionar o botão 'Write Flash'.

3-Rapidamente, ligar o PIC.

É interessante notar que esse método é especialmente útil para o ajuste de constantes, já que de forma rápida é possível alterá-las, sem que seja necessária a remoção do circuito integrado.

3 RESULTADOS E DISCUSSÃO

3.1Gastos

Na tabela abaixo encontram-se discriminados os valores gastos no projeto.

Quantidade	*Descrição*	*Preço*	*Total*
2	Acopladores rígidos (1/2" para 5/8")	R$ 55,00	R$ 110,00
2	Braçadeiras do eixo (5/8")	R$ 17,00	R$ 34,00
2	Eixos chavetados (5/8")	R$ 34,00	R$ 68,00
2	Motores Elétricos CIM	R$ 47,60	R$ 95,20
2	Caixa de Redução 16:1	R$ 187,00	R$ 374,00
2	Rodas com pneu	R$ 40,00	R$ 80,00
1	Acelerômetro de 2 eixos iMEMS	R$ 61,20	R$ 61,20
1	Giroscópio iMEMS	R$ 95,20	R$ 95,20
1	PIC 18f4331	R$ 35,00	R$ 35,00
1	MaxStream Xbee-PRO Digital Radio	R$ 54,50	R$ 54,50
1	Bateria 12V de motocicleta	R$ 120,00	R$ 120,00
2	Victor Speed Controller	R$ 195,50	R$ 391,00
1	Conjunto de fios	R$ 30,00	R$ 30,00
1	Componentes eletrônicos	R$ 40,00	R$ 40,00
1	Base de alumínio	R$ 295,00	R$ 295,00
1	Tubos diversos e parafusos	R$ 60,00	R$ 60,00
N/A	Trabalho torneiro	R$ 200,00	R$ 200,00
			R$ 2.143,10

Tabela 6 – Gastos do projeto

O total, cerca de dois mil reais, é considerado aceitável, na medida que um veículo desse tipo, em sua versão comercial, não sai por menos de dez mil reais[14].

Cabe ressaltar que o projeto desenvolvido, ao qual se refere este relatório, de maneira alguma pode ser comparado ao veículo comercial.

Será visto mais adiante que a duração da bateria, a falta de redundância e a potência dos motores são fatores limitantes do projeto.

3.2 Histórico

Primeiramente, cabe aqui contar e discutir o histórico de desenvolvimento do projeto, de forma que futuros trabalhos tenham conhecimento do processo de implementação e desenvolvimento do veículo.

Testes preliminares apenas considerando os sensores e a medição do ângulo (estimado versus real) -mostrados através do software de aquisição de dados – fizeram com que a equipe decidisse que os testes com os motores poderiam começar.

Não houve uma metodologia clara para esses testes, foi decidido que os primeiros seriam no sentido de ligar e construir o veículo e vê-lo equilibrar-se. Uma vez equilibrado, testes mais criteriosos e metódicos seriam feitos no sentido de deixar a experiência para o usuário o mais suave possível.

Os motores foram conectados aos *Speed Controllers*, os quais tinham como referência o recebimento do sinal PWM para o controle da velocidade. Notou-se que o comportamento estava dentro do esperado, com erros pequenos e aceitáveis.

Após este pequeno ensaio preliminar, foi feita a montagem da base, dos motores acoplados à redução e às rodas. Feito isso, ligamos os motores diretamente nos 12V para que pudéssemos testara condição da caixa de redução, verificando se as peças estavam bem fixadas e de que não havia escorregamento entre as mesmas.

Dando seguimento aos testes, finalmente ligamos todo o sistema e fizemos várias observações.

De primeira, o sistema apresentou um comportamento excessivamente oscilatório e não conseguiu equilibrar-se. Após uma série de checagens minuciosas descobriu-se que os sensores estavam ligados de forma invertida (O giroscópio no lugar do acelerômetro e vice-versa).

Ficou claro o motivo principal do não funcionamento, mas também foi identificado que as constantes geravam uma saturação rápida nos motores, fazendo com que o sistema chegasse rapidamente à sua velocidade final. Em conjunto com a mudança de sensores, diminui-se a ordem de grandeza das constantes por um valor de 10.

Mesmo após essas medidas, o comportamento ainda era um pouco oscilatório e brusco, restando agora a análise mais detalhada das constantes. Contudo, mesmo com esse comportamento oscilatório o sistema conseguiu equilibrar a base, provando que o conceito do projeto estava correto e que tudo encaminhava-se bem para que o sistema apresentasse o comportamento desejado

Após levar o sistema a sua forma definitiva, com tudo no seu devido lugar, foi feito um grande trabalho no sentido do ajuste de constantes. O trabalho foi muito facilitado devido a presença do Bootloader, que permitia alterações complexas de forma rápida.

Muitos testes foram feitos e chegou-se aos valores finais, que representaram o melhor compromisso entre equilíbrio, velocidade máxima e oscilação.

Ao fazer um balanço, pode-se dizer que o projeto foi bem sucedido.

3.3Resultados

Para a obtenção das características e respostas de nosso sistema, fizemos alguns testes com nosso protótipo e obtivemos os resultados através dos gráficos providos pelo nosso programa de aquisição de dados. A metodologia para a realização destes testes foi a de tentar verificar e demonstrar o funcionamento do nosso protótipo em situações quer permitiriam a obtenção de dados relevantes na análise. Seguem abaixo os resultados bem como as análises:

Teste 1 – Verificação da resposta dos sensores:

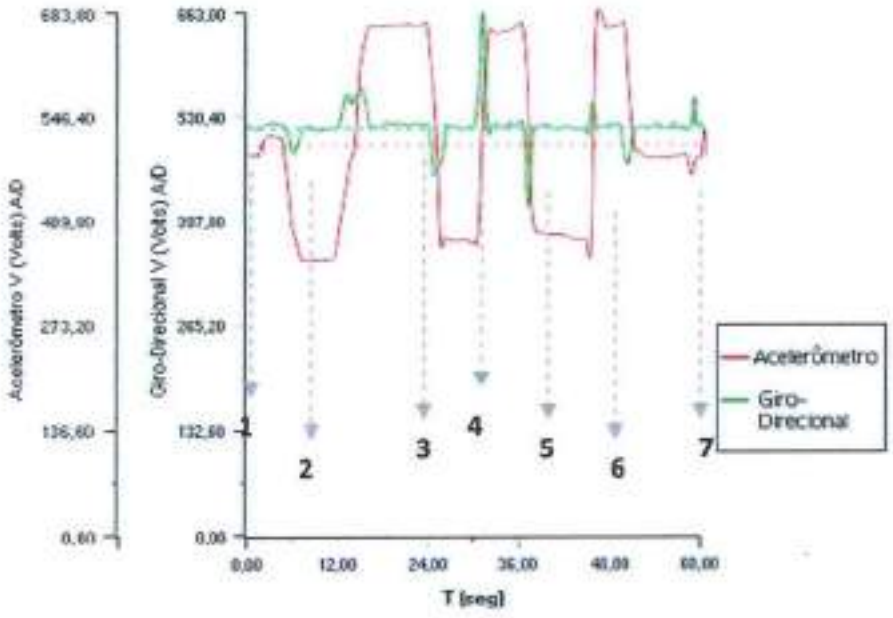


Gráfico 5 – Resposta dos sensores (Movimento)

A partir do Teste 1, pudemos comprovar o correto funcionamento dos sensores que servem de referência para a realização do controle de nosso veículo.

Portanto, o gráfico X mostra as respostas dos sensores mediante a variação do ângulo da base de nosso protótipo. O método utilizado para a realização deste teste foi através do desligamento dos motores através da chave implementada com este fim, possibilitando assim que inclinássemos a base de nosso veículo conforme desejássemos.

A representação no gráfico de dá conforme a legenda de cores. As linhas pontilhadas horizontais representam o valor médio na escala do sensor, isto é, para o Giro-Direcional este valor representa uma variação angular nula da plataforma e para o Acelerômetro este valor representa um ângulo de inclinação nulo da plataforma.

As coordenadas do gráfico apresentam no eixo Y, vertical, sendo uma escala para o Acelerômetro e outra para o Giro-Direcional, os valores de tensão convertido para os valores A/D do Microcontrolador. Como este Microcontrolador possui seu conversor analógico com resolução de 10 bits, ele consegue representar os valores numa escala de 0 a 1024, com o 0 representando 0 V e o 1024 representando 5V.

Para a apresentação das informações constantes no gráfico, dividimo-los em pontos notáveis. Para a explicação dos resultados, utilizaremos a convenção de um observador situado à direita do veículo e portanto visualizando a variação positiva do ângulo da plataforma, girando ao redor de um eixo imaginário, no sentido horário.

O segmento a partir do ponto 1 do gráfico representa a primeira variação angular da base do veículo e como podemos verificar, a saída do Giro-Direcional adquire o valor negativo, indicando que uma inclinação no sentido de diminuir o ângulo da plataforma, isto é, no sentido anti-horário, foi executada. Os pontos 3 e 5 apresentam o mesmo comportamento.O Giro-Direcional proporciona a leitura da razão de variação do ângulo da plataforma, logo, a derivada do gráfico ângulo, que é dado pela saída do Acelerômetro. Podemos notar que o comportamento está conforme o esperado, já que o gráfico do Acelerômetro a partir deste ponto mostra uma declividade negativa, de acordo com o esperado.O fato de a curva do Acelerômetro estar abaixo da linha média trás o significado de que este sensor está medindo ângulos negativos da plataforma.

Verificamos também a partir do ponto 1e de outros pontos do gráfico que a resposta do Giro-Direcional é mais rápida que a do Acelerômetro, verificação esta feita através da pequena defasagem entre as a indicação de variação angular pelo Giro-Direcional e a alteração do valor de saída do Acelerômetro.

No segmento a partir doponto 2,verificamos o resultado inverso ao apresentado no ponto 1, com o Giro-Direcional apresentando um saída positiva, significando que a

base está com um variação angular no sentido horário, o que se verifica no gráfico do Acelerômetro, com uma declividade positiva e que passa de valores abaixo do ponto médio para valores acima do ponto médio, evidenciando uma inclinação angular positiva da base.

Com a curva em torno dos pontos 4,5 e 7, verificamos a resposta do sistema a variações bruscas na plataforma, isto é, utilizando uma razão de inclinação maior em relação às usada em torno dos pontos 1 e 2. Tal comportamento é evidenciado pela maior amplitude na resposta do Giro-Direcional, bem como na maior inclinação da curva do sinal do Acelerômetro, evidenciado assim o funcionamento correto dos sensores.

Por fim, no ponto 7 conseguimos verificar uma situação em o ângulo da plataforma permanece praticamente constante, resultado este evidenciado pela saída do Acelerômetro, a menos de pequenas variações. A saída do Giro-Direcional apresenta valor perto do ponto médio, o que é correto, já que o ângulo permanece constante. Mas como se pode notar neste ponto e também nos demais pontos em que ângulo permanece constante, a saída do Giro-Direcional fica a todo momento mostrando pequenas variações, que serão discutidas nos resultados do próximo teste abaixo.

Teste 2 – Saída dos sensores com veículo em repouso

O objetivo deste teste era de verificar o comportamento dos sensores na situação em que o protótipo encontrava-se em repouso, além de verificarmos o valor a ser colocado como *set-point* no código.

Para o acelerômetro, o valor do *set-point* determina o ângulo verdadeiro da base em que o controlador detectará que a plataforma é nula. Portanto, para a realização deste teste buscamos manter a plataforma alinhada o máximo possível com o solo, inclinação esta verificada através de um inclinômetro de bolha. Já para o caso do Giro-Direcional, o valor do *set-point* indica o valor em que o controlador considerará que a plataforma não esteja com uma variação angular. Portanto, o valor do *set-point* no caso do Giro-Direcional infuencia diretamente no controle, já que um valor diferente do valor em que realmente não há inclinação resultaria no sinal incorreto para acelerar os motores. Porém, mais abaixo na discussão deste teste veremos que o valor de saída do Giro-Direcional é bastante ruidoso e portanto um valor mínimo de erro no *set-point* não traria tantos prejuízos à ação de controle.

Abaixo encontram-se os gráficos

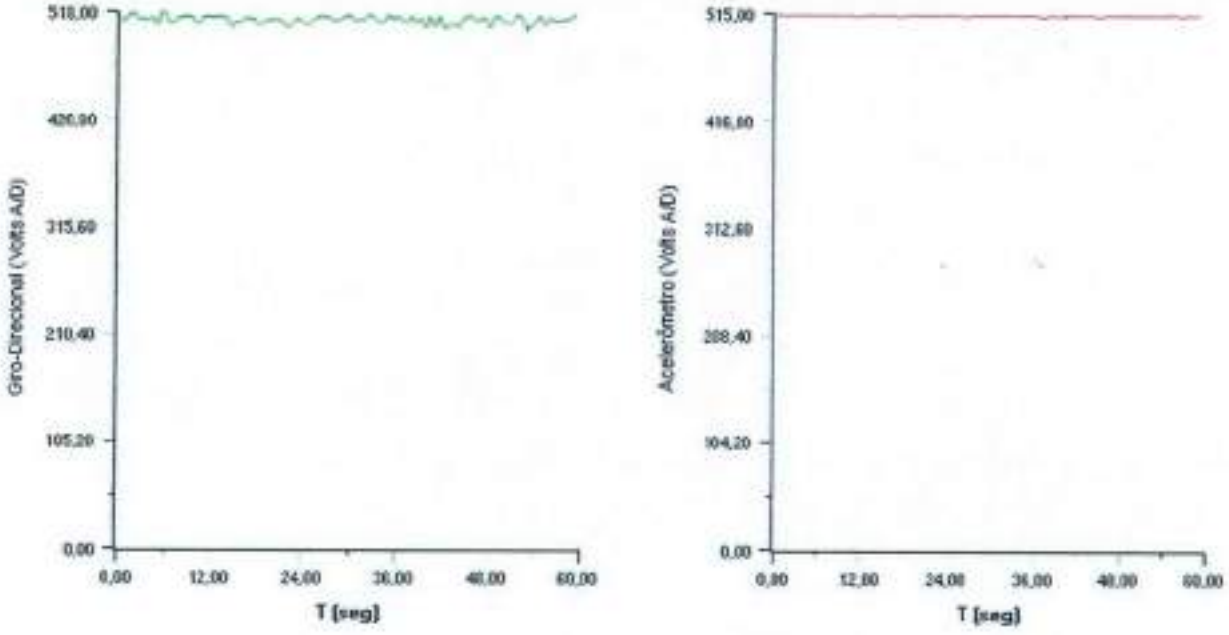


Gráfico 6 – Resposta dos sensores (Repouso)

Logo, através dos gráficos das saídas do Giro-Direcional e do Acelerômetro, pudemos constatar que o *set-point* do primeiro situa-se na faixa do valor 515 em Volts A/D e o valor do segundo situa-se na faixa de 514 Volts A/D.

O que é possível verificar também é a ausência de ruídos no caso do Acelerômetro e presença deles no caso do Giro-Direcional. Isto se deve ao fato de, como já explicamos nas seções acima, o Acelerômetro possuir perfil de baixa freqüência e possuir respostas mais lentas , ao contrário do Giro-Direcional, que apresenta um perfil de alta-frequência, com respostas rápidas. A partir deste resultado, verificamos o quão importante é a adoção de um filtro ligado à saída dos sensores, principalmente do Giro-Direcional. Tendo isto em mente, e como pudemos comprovar pelos gráficos acima, implementamos o Filtro Complementar.

Teste 3 – Resposta dos Motores em bancada

No teste 3, verificamos a correta resposta dos motores de acordo com as variações nos sensores, ora com a base inclinada em um ângulo negativo, ora com a base inclina em ângulo positivo.Para a realização deste teste, posicionamos o nosso protótipo apoiado em uma bancada de forma que as rodas ficassem suspensas e portanto não tivessem contato com o solo. Isto significa que diante da impossibilidade

de corrigir a variação angular na plataforma, o sistema de controle aumentaria progressivamente a velocidade dos motores de forma que atingiria a saturação.

Abaixo está o gráfico mostrando os resultados:

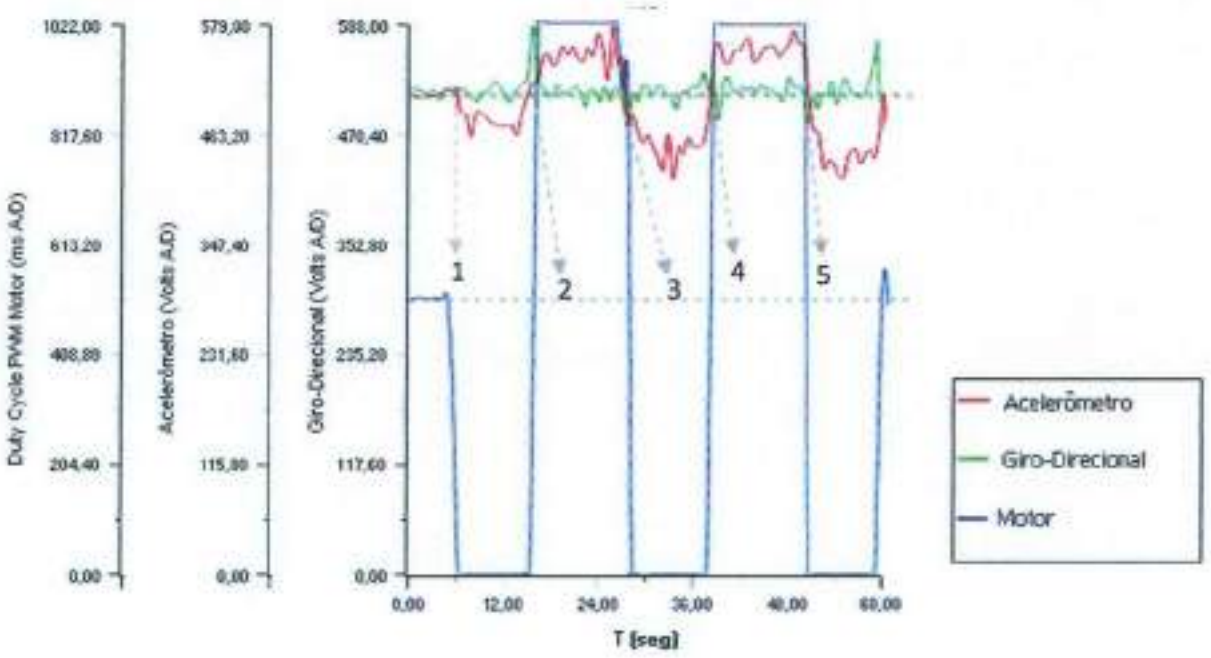


Gráfico 7 – Sensores e PWM do motor (Sem carga)

O gráfico segue o mesmo padrão do teste 1, com a linha pontilhada horizontal representando o valor médio da entidade medida, e , no caso do gráfico correspondente ao *Duty cycle* do sinal PWM enviado ao *Speed Controler*, a linha pontilhada representa o ponto neutro do motor, isto é, o ponto em que o motor não está acelerando progressivamente nem retardadamente.

Nos pontos 1, 3 e 5, verificamos o comportamento do motor na situação de inclinação da base para um ângulo negativo. Como podemos constatar, a resposta do motor foi realizada no sentido de acelerá-lo retardadamente, isto é, no sentido girar acelerá-lo no sentido anti-horário, considerando o mesmo referencial que foi determinado no teste 1.

Já, nos pontos 2 e 4, verificamos o comportamento do motor na situação de inclinação da base para um ângulo positivo.Nesta situação, a resposta do motor foi realizada no sentido de acelerá-lo retardadamente, isto é, no sentido de acelerá-lo no sentido horário.

Estas respostas estão de acordo com o que se espera do sistema de controle, já que em uma situação de ângulo negativo, espera-se que o motor tente acelerar retardadamente com, gerando assim um torque no sentido horário que aproximará a plataforma do ponto de ângulo zero. O mesmo é verdade para a situação de ângulo positivo, que acarreta na aceleração progressiva do motor, gerando o torque no sentido anti-horário.

Teste 4 – Protótipo na situação normal de uso

Por fim, verificamos agora o teste do protótipo na situação de uso por um indivíduo.

Abaixo encontra-se o gráfico representando esta situação:

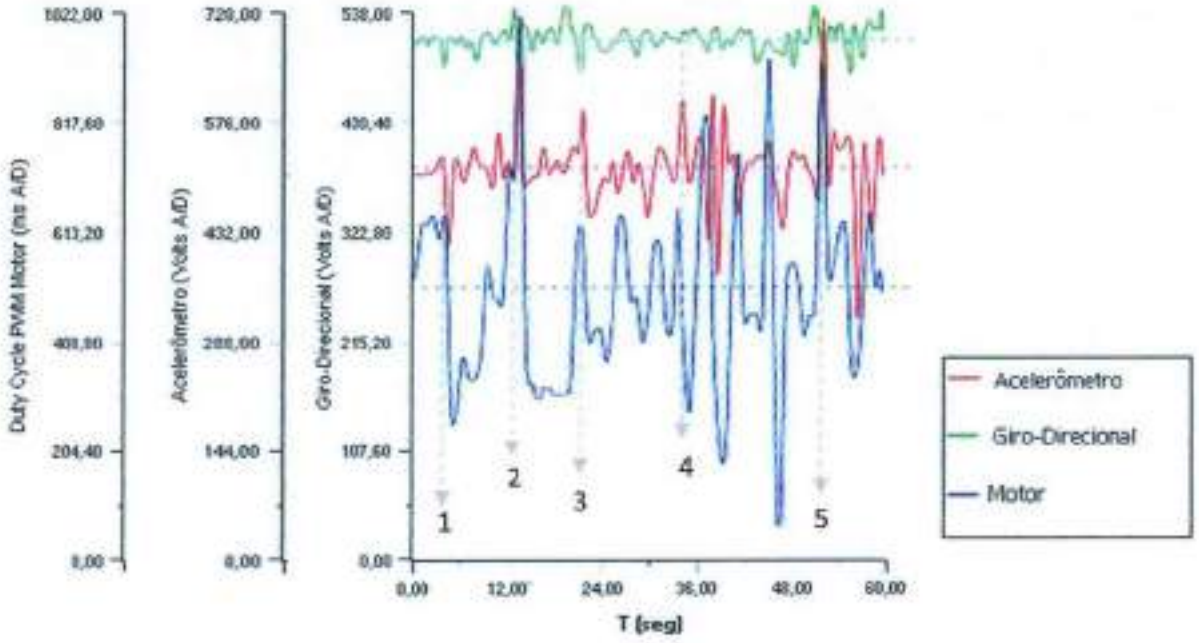


Gráfico 8 – Sensores e PWM do motor (Situação Normal)

Como é possível constatar no gráfico, o sistema nunca consegue ficar perfeitamente equilibrado com um ângulo de 0 graus na base, evidenciado pela saída do acelerômetro, uma vez que o mesmo está sempre realizando ajustes de forma a compensar os torques que tendem a tirá-lo da situação de equilíbrio. Esse comportamento fica bastante evidenciado no trecho entre os pontos 3 e 4, trecho este em que fizemos o teste buscando manter o equilíbrio e manter a base o mais perto possível da angulação nula.

As ações de controle que buscam o equilíbrio do sistema são dadas pela tabela abaixo, nas possíveis condições que o sistema pode enfrentar:

	Ângulo Positivo	**Ângulo nulo**	**Ângulo Negativo**
Razão Angular Positiva	Acelerar Motor sentido positivo RAPIDAMENTE	Acelerar Motor sentido Positivo	Mantém Aceleração do Motor ou Acelera Motor no sentido negativo
Razão Angular Nula	Acelerar Motor no sentido positivo	Não alterar condição do Motor	Acelerar Motor no sentido Negativo
Razão Angular Negativa	Mantém Aceleração do Motor ou Acelera Motor no sentido positivo	Acelerar Motor no sentido Negativo	Acelerar Motor no sentido Negativo RAPIDAMENTE

Tabela 7 – Ações de Controle

A tabela acima é bastante intuitiva e demonstra as ações do sistema de controle no sentindo de retornar ao equilíbrio. As situações que merecem destaque são as de Ângulo Negativo e Razão Angular Positiva e a de Ângulo Positivo e Razão Angular Negativa. Nestas situações, a ação no do sistema de controle vai depender das constantes utilizadas no sistema, influenciando o tempo de resposta. Por exemplo no caso de um Ângulo Positivo e Razão Angular Negativa, se o ganho do derivativo multiplicado pela razão Angular for igual ao ganho do proporcional multiplicado pelo ângulo , o motor manteria praticamente o mesmo regime. No entanto, se utilizássemos uma constante da componente proporcional maior, o sistema de controle aceleraria o motor no sentido positivo, já que o valor dessa parcela seria maior que a subtração dada pelo derivativo, resultaria em um incremento do sinal do motor . Tal ação é garantida pela fórmula utilizada no código, já citada anteriormente:

```
motor += ((KP * angulo) + (KD * giro_velocidade);
```

Logo, tendo em vista essas ações, podemos analisar as respostas no gráfico deste teste.

No ponto 1, verificamos a situação de Ângulo Negativo e Razão Angular Negativa, e segundo a tabela acima, deveríamos ter uma resposta rápida do motor no sentido negativo, de forma a restabelecer o equilíbrio em torno da posição de Ângulo

nulo na base, o que é comprovado pelo gráfico, através da aceleração do motor no sentido negativo, o que resulta na plataforma retornando à situação de ângulo 0 e portanto em sucesso na realização do equilíbrio.

Já nas imediações do ponto 2 vemos a situação simétrica, com um Ângulo Positivo e Razão Angular Positiva, devendo o sistema de controle atuar de forma a acelerar o motor. O resultado mostrado no gráfico também está de acordo com o que se espera, demonstrando o correto funcionamento do sistema de controle.

Já o trecho situado entre os pontos 4 e 5 representa a situação de movimentação do veículo para a frente e para trás, movimento este conseguido pelo usuário ao mudar a inclinação de seu corpo e com isso alterar a distribuição de momentos que atuam no sistema. Como se pode notar no gráfico, uma situação deste tipo exige grandes cargas e alterações bruscas do regime do motor, já que teremos uma condição de inversão de movimento, o que exige que o motor forneça um grande torque de forma a manter o equilíbrio e desta forma "atender" às solicitações do usuário. Nota-se a ainda grande amplitude no sinal enviado aos motores, o que resulta em uma situação de grande consumo da carga da bateria.

Através da análise dos gráficos, conseguimos comprovar o correto funcionamento de nosso protótipo, que atende às especificações e às expectativas quanto a sua performance. Todas as premissas e condições que tínhamos em mente durante a concepção do projeto foram atendidas pelo protótipo, comprovando o sucesso de nossa construção.

4 CONCLUSÕES

Nesta seção, serão analisadas, parte a parte as peças que formam o conjunto do processo. Temos também sugestões que os autores julgam interessantes para quem quiser dar continuidade ao trabalho.

4.1 Filtro complementar

Os resultados obtidos pela utilização do filtro complementar são comprovados pelos testes que realizamos em nosso protótipo, que apresentaram resultados satisfatórios, tanto em acurácia quanto em tempo de resposta, no que diz respeito à determinação do ângulo da base, valor este utilizado como referência para a realização do controle. Além do que, com o equilíbrio do sistema, fica praticamente demonstrado que o filtro é suficiente para esta estimativa de ângulo.

Portanto, o filtro complementar provou ser uma decisão acertada de projeto. Evitando a complexidade computacional e matemática do filtro de Kalman, obteve-se um filtro muito estável e com respostas adequadas ao problema (e.g erros baixos na medida do ângulo).

Em um sistema comercial, a aplicação do filtro de Kalman e redundâncias nos sensores se justificaria, pois a resposta necessitaria de uma grande precisão. Contudo, em um projeto de formatura, em que temos menos recursos e até mesmo tempo disponível para o desenvolvimento, o filtro complementar é uma boa solução.

4.2 Controlador PID

O controlador do tipo PID representa uma teoria consagrada nas matérias de controle. Neste projeto não poderia ser diferente. Após o ajuste das constantes, se obteve um sistema muito próximo do esperado.

Cabe o comentário a respeito de que neste trabalho foi utilizado um controlador do tipo PD, em detrimento do PID completo. Isso se deu devido ao fato de que a resposta já é satisfatória apenas com as duas componentes – Proporcional e

derivativa. Soma-se a esta constatação o fato de que temos de forma fácil a saída derivativa do sistema, através do giroscópio, que mede a velocidade angular do sistema. Ao contrário da componente derivativa, a componente relativa a integral necessitaria de cálculos por parte do PIC, tornando mais difícil a sua implementação. Este também foi mais um fato que nos fez optar apenas pelas componentes P e D.

O ajuste de constantes foi feito de forma empírica. Com ajuda do bootloader, altera-se o código do projeto e, portanto, as constantes - através da reprogramação do firmware.

A cada alteração de constantes, era feita uma bateria de teste. Tais testes visavam medir o caráter oscilatório do sistema e a velocidade de sua resposta. Após obter a ordem de grandeza das constantes, o ajuste fino foi feito com base nos gráficos do sistema de telemetria.

4.3 Sensores

Os sensores, apesar de possuírem um custo relativamente baixo, mostraram um perfomance boa. Como mencionado no filtro complementar, a estimativa do ângulo se mostrou muito adequada, tanto em relação à acurácia quanto em relação aos tempos de resposta às variações de ângulo da plataforma.

Cabe ressaltar que, no caso de um projeto comercial, dever-se-ia adicionar mais sensores como forma de redundância. Afinal, se um dos dois sensores parar de funcionar, o equilíbrio deixa de acontecer e o usuário pode sofrer um acidente.

A impressão que fica é a de que um par de sensores extra resolveriam possíveis problemas de falhas de sensor. O par inicial –Giroscópio e acelerômetro- demonstrou ser muito confiável, portanto, julga-se que apenas um par extra de sensores em paralelo daria cabo ao problema.

4.4 Mecânica

A decisão de basear a parte mecânica em um trabalho já consagrado como o do MIT*[1]*poupou os autores de muito tempo de desenvolvimento de um modelo CAD adequado para a aplicação.

A base de alumínio utilizada é composta por uma liga extremamente dura e resistente, liga esta utilizada também em aplicações aeronáuticas. Temos,

portanto, que sua resistência atende e até mesmo ultrapassa os requisitos para nosso projeto, o que é positivo, já que com isso garantimos a robustez da estrutura mecânica.

Uma de nossas preocupações quanto a parte mecânica era a do perfeito alinhamento do conjunto Motor – Redução – Eixo – Rolamento - Roda, já que um grande desvio em um desses componentes poderia comprometer o balanceamento do veículo bem como poderia trazer danos aos componentes, principalmente à caixa de redução, que tem especificada em seu manual uma tolerância máxima de 5 graus de desvio de alinhamento no eixo ligado a ela. Esse problemas, no entanto, não se apresentaram, já que o corte com a máquina de jato d'água foi preciso o suficiente para garantir que os furos saíssem alinhados conforme estabelecido no desenho CAD.

Enfim, a idéia era poupar o trabalho na estrutura mecânica e manter o foco no controlador do sistema. Esta idéia provou-se eficaz. A montagem da estrutura apesar de não ser trivial, exigindo trabalhos que necessitam de cuidado e atenção como a utilização de furadeiras de bancada e máquinas de corte, não tomou um tempo muito extenso, nos permitindo manter o já mencionado foco na eletrônica.

4.5 Bateria

A bateria é um ponto fraco do projeto e portanto um dos componentes que poderia ser melhorado em uma atualização. Sua duração é muito limitada e permite rodagens de, no melhor caso, 25 minutos, visto que a autonomia dada pela bateria depende de diversos fatores como inclinação do terreno em que é realizado o teste, estilo de condução do veículo, dentre outros.

A sugestão dos autores é a de procurar por uma bateria especifica para veículos elétricos.

A bateria que foi usada foi do tipo utilizada em motocicletas. Esse tipo de bateria é concebida para que tenha a capacidade de fornecer correntes altas durante pequenos intervalos de tempo, como na partida do motor. Temos ainda, também, o problema do tempo de carregamento da bateria, causado tanto pela utilização de um carregador de tomada, quanto pela característica da bateria, que somente permite o uso de correntes de até 10 % da capacidade nominal de carga da bateria, que no nosso caso, sendo a bateria de 18Ah, permite o carregamento com no máximo 1,8 A. Como nosso carregador fornece ao máximo 0,9 A, temos que o tempo de carregamento previsto de 20 horas, para a situação da bateria totalmente descarregada.

4.6 Motor

O motor foi escolhido através de um cálculo simples de potência requerida e também pelo fato de estarmos baseando nossa mecânica no consagrado projeto da referência bibliográfica*[1]*, a qual já estava adaptada àquele tipo de motor.

Entretanto, verificamos que o desempenho ficou aquém do esperado, mesmo com a utilização de uma caixa de redução com razão de redução maior do que a utilizada no projeto do MIT, o que nos trouxe um aumento no torque disponível .Caso o usuário se incline de maneira muito brusca, a força do motor pode não ser suficiente para reequilibrá-lo, como verificamos em algum de nossos testes. Esta condição, porém, verifica-se no uso do veículo em situações extremas, situações essas que estão fora do que esperávamos para o nosso projeto.

Um motor mais parrudo seria o ideal para esse tipo de aplicação. Tal mudança no entanto não seria trivial, já que precisaríamos de uma grande reformulação do da estrutura mecânica do projeto, através da realização de um novo design para a base, com novos furos e espaçamento adequado para comportar os novos equipamentos, que além dos motores incluiriam também novas caixas de redução que se adaptassem aos mesmos.

5 BIBLIOGRAFIA

[1]DIY Segway Technical Documentation. Disponível em: <web.mit.edu/first/segway>. Acesso em: 10 maio 2010.

[2]OGATA, K. Modern Control Engineering, 3ª ed. Prentice Hall, 1997.p 803 – 812.

*[3]Building a Balancing Scooter.*Disponívelem: <http://tlb.org/scooter.html>. Acessoem: 10 maio 2010.

*[4]LPY5150ALDatasheet,*STMicroelectronics. Ano 2009.

[5]ADXL320Datasheet , Analog Devices. Ano 2010.

[6] S.W, Ahmad M.N, Osman J.H.S, Husain Controller Design for Two-wheels Inverted Pendulum Vehicle Using PISMC A.R, 2006.

[7]PID Controller Theory. Disponível em: <http://team358.org/files/programming/PIDControlTheory_rev3.pdf>. Acesso em: 01/10/2010.

[8]RIBEIRO, Ricardo. Implementação de um sistema decontrole de um pêndulo invertido. Dissertação do programa de pós-graduação. Itajubá, 2007

[9]http://diysegway.blogspot.com/ Acesso em: 13 maio 2010

[10]http://www.csharphelp.com/2007/03/2d-charting-user-control/Acesso em: 13 novembro2010.

[11]http://www.dreamincode.net/forums/topic/37361-serial-port-communication-in-vbnet/Acesso em: 15 novembro 2010.

*[12]*http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm*Acesso em: 15 novembro 2010.*

6 APÊNDICES

Apêndice A – Listagem do código no microcontrolador

```
        float KP,KD,KS;

        signed int motor_esq = 512, motor_dir = 512;

        signed int turn = 0;

        int freq=0;

        signed int acelarometro, stick, giroscopio,acelerometro_graus, stick_curva,
giro_velocidade;

        int pwm=0;

        char text[20];

        int angulo;

        int numero;

        int i=0;

        float angulo = 0, motor = 0;

        void interrupt () {

        if (PIR1.TMR1IF) {

        TMR1H = 15535 >> 8;

                  TMR1L = 15535 & 0xFF;

                        PIR1.TMR1IF=0;

        if(pwm==0)

                        {

                        PORTC.RC1 = 1;

                        PORTC.RC2 = 1;

        pwm=1;

                        }

        if(pwm==1)

                        {

        pwm=0;

                        }

                }
```

```
if (PIR1.CCP1IF) {
                PORTC.RC1 = 0;
                PIR1.CCP1IF = 0;
        }
if (PIR2.CCP2IF) {
                PORTC.RC2 = 0;
                PIR2.CCP2IF = 0;
        }
}
void main() {
KP=0.05;
KD=0.15;
KS=0.4;
TRISD=0;
ANSEL0=0x07;
TRISA=0xFF;
TRISB=0;
ADCON0=0x01;
T0CON.T0CS = T0CON.PSA = T0CON.T0PS0 = 0;
T0CON.T0PS2 = T0CON.T0PS1 = 1;
ADC_Init();
UART1_Init(19200);
while (1) {

acelerometro = ADC_Get_Sample(2);
giroscopio = ADC_Get_Sample(3);
stick = ADC_Get_Sample(1);
acelerometro_graus = (acelerometro - 512) * 0.4375;

                stick_curva = (stick - 512) * 0.4375;
                giro_velocidade = (giroscopio - 522) * 0.34;
```

```
                angulo = 0.996 * (angulo + giro_velocidade * 0.001);
                angulo += 0.004 * acelerometro_graus;

                motor += ((KP * angulo) + (KD * giro_velocidade);
if (motor <= -511) { motor = -511; }
else if (motor >= 511) { motor = 511; }
turn = stick_curva * KB;
                motor_esq -= turn;
motor_dir += turn;
                CCPR1 = (20535 + motor_esq*5);

                CCPR2 = (20535 + motor_dir*5);
if (PORTA.RA4 == 1) {
                    motor_dir = motor_esq = 512;
motor = 0;
                }
if (UART1_Data_Ready())
    {
        UART1_Read_Text(text,"##",255);
if(strncmp(text,"gyro1",5)==0)
        {
         temp_res = giroscopio;

                    UART1_Write('g');
        UART1_Write((temp_res>>5));

        UART1_Write((temp_res&0b00011111));
        }
if(strncmp(text,"accel",5)==0)
        {
         temp_res = acelerometro;
                    UART1_Write('a');
```

```
        UART1_Write((temp_res>>5));
        UART1_Write((temp_res&0b00011111));
        }
 if(strncmp(text,"stick",5)==0)
        {
         temp_res = stick;
                    UART1_Write('h');
        UART1_Write((temp_res>>5));
        UART1_Write((temp_res&0b00011111));
        }
 if(strncmp(text,"motor_dir",6)==0)
        {
        UART1_Write('r');
        UART1_Write((motor_dir>>5));
        UART1_Write((motor_dir&0b00011111));
        }
 if(strncmp(text,"motor_esq",6)==0)
        {
        UART1_Write('l');
        UART1_Write((motor_esq>>5));
        UART1_Write((motor_esq&0b00011111));
        }
 if(strncmp(text,"dtmr",4)==0)
        {
        UART1_Write('d');
        UART1_Write((dtmr>>8));
        UART1_Write((dtmr&0b11111111));
        }
 if(strncmp(text,"angulo",6)==0)
        {
 if(angulo<0)
          {
```

```
            temp_res = -(signed int)angulo;
            temp_res = temp_res+512;
            }
else
            {
            temp_res = (unsigned int)angulo;
            }
          UART1_Write('v');
          UART1_Write(temp_res>>5);
          UART1_Write(temp_res&0b00011111);
          }
if(strncmp(text,"ctes",4)==0)
          {
          UART1_Write(KP);
          UART1_Write(KD);
          UART1_Write(KS);
          }
if(strncmp(text,"kp",2)==0)
          {
          KP=(text[4]-48)+(text[3]-48)*10+(text[2]-48)*100;
          }
if(strncmp(text,"kc",2)==0)
          {
          KD=(text[4]-48)+(text[3]-48)*10+(text[2]-48)*100;
          }
if(strncmp(text,"ki",2)==0)
          {
KS=(text[4]-48)+(text[3]-48)*10+(text[2]-48)*100;
}
          }
  }
}
```

Apêndice B – Listagem frmMain.cs do software de monitoramento

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using PCComm;
using SeabeckEnterprises.SeabeckCharts;
namespace PCComm
{
public partial class frmMain : Form
    {
CommunicationManager comm = new CommunicationManager();
int poll = 0;
string transType = string.Empty;
double[] plot = new double[101];
public frmMain()
        {
                InitializeComponent();
//LineChart Data
double[] x1 = new double[101];
double[] y1 = new double[101];
double[] y2 = new double[101];

//ScatterChart Data
double[] x3 = new double[11];
double[] y3 = new double[11];
double[] y4 = new double[11];
//Values for LineChart
for (int i = 0; i < x1.Length; i++)
            {
                x1[i] = Math.PI + (((double)i) * (Math.PI / 100.0));
y1[i] = Math.Sin(x1[i]);
                y2[i] = Math.Cos(x1[i]);
}
//Values for ScatterChart
for (int j = 0; j < y3.Length; j++)
            {
                x3[j] = j;
                y3[j] = Math.Exp(x3[j]) + 1000;
                y4[j] = -Math.Exp(x3[j]) - 1000;
            }
this.lineChart1.SetXValues(x1);
this.lineChart2.SetXValues(x1);
this.lineChart3.SetXValues(x1);
this.lineChart4.SetXValues(x1);
this.lineChart1.AxisColor = System.Drawing.Color.Black;
this.lineChart1.BGColor = System.Drawing.SystemColors.Control;
this.lineChart1.DrawClicks = true;
this.lineChart2.DrawClicks = true;
this.lineChart3.DrawClicks = true;
this.lineChart4.DrawClicks = true;
// this.lineChart1.DrawGrid = true;
this.lineChart1.DrawLegend = true;
this.lineChart1.LegendAlpha = ((System.Byte)(0));
this.lineChart1.LegendBG = System.Drawing.SystemColors.Control;
this.lineChart1.LegendSizeX = 65F;
this.lineChart1.LegPosition = SeabeckEnterprises.SeabeckCharts.LegendPosition.NW;
this.lineChart1.Name = "lineChart1";
this.lineChart2.Name = "lineChart2";
this.lineChart3.Name = "lineChart3";
this.lineChart3.Name = "lineChart4";
this.lineChart1.Size = new System.Drawing.Size(300, 300);
this.lineChart2.Size = new System.Drawing.Size(300, 300);
this.lineChart3.Size = new System.Drawing.Size(300, 300);
this.lineChart4.Size = new System.Drawing.Size(300, 300);
```

```
this.lineChart1.SubTitle = "A/D";
this.lineChart1.SubTitleColor = System.Drawing.Color.Black;
this.lineChart1.TabIndex = 0;
this.lineChart1.Title = "Gyro 1 ";
this.lineChart2.Title = "Gyro 2 ";
this.lineChart3.Title = "Accel  ";
this.lineChart4.Title = "Motor Direito  ";
this.lineChart1.TitleColor = System.Drawing.Color.Red;
this.lineChart2.TitleColor = System.Drawing.Color.Red;
this.lineChart3.TitleColor = System.Drawing.Color.Red;
this.lineChart4.TitleColor = System.Drawing.Color.Red;
this.lineChart1.TitleFont = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Pixel);
this.lineChart1.XLabel = " T (seg)";
this.lineChart1.YLabel = " V (Volts)";
this.lineChart2.XLabel = " T (seg)";
this.lineChart2.YLabel = " V (Volts)";
this.lineChart3.XLabel = " T (seg)";
this.lineChart3.YLabel = " V (Volts)";
this.lineChart4.XLabel = " T (seg)";
this.lineChart4.YLabel = " PWM (Volts)";
        }

privatevoid frmMain_Load(object sender, EventArgs e)
        {
            LoadValues();
            SetDefaults();
            SetControlState();
        }

privatevoid cmdOpen_Click(object sender, EventArgs e)
        {
            comm.Parity = cboParity.Text;
            comm.StopBits = cboStop.Text;
            comm.DataBits = cboData.Text;
            comm.BaudRate = cboBaud.Text;
            comm.DisplayWindow = rtbDisplay;
            comm.Grafico = lineChart1;
comm.Grafico2 = lineChart2;
            comm.Grafico3 = lineChart3;
comm.Grafico4 = lineChart4;
            comm.PortName = cboPort.Text;
            comm.OpenPort();

            cmdOpen.Enabled = false;
            cmdClose.Enabled = true;
            cmdSend.Enabled = true;
        }

///<summary>
/// Method to initialise serial port
/// values to standard defaults
///</summary>
privatevoid SetDefaults()
        {
            cboPort.SelectedIndex = 0;
            cboBaud.SelectedText = "9600";
            cboParity.SelectedIndex = 0;
            cboStop.SelectedIndex = 1;
            cboData.SelectedIndex = 1;
        }

///<summary>
/// method to load our serial
/// port option values
///</summary>
privatevoid LoadValues()
        {
            comm.SetPortNameValues(cboPort);
            comm.SetParityValues(cboParity);
            comm.SetStopBitValues(cboStop);
```

```
        }

        ///<summary>
        /// method to set the state of controls
        /// when the form first loads
        ///</summary>
        privatevoid SetControlState()
        {
            rdoText.Checked = true;
            cmdSend.Enabled = false;
            cmdClose.Enabled = false;
        }

        privatevoid cmdSend_Click(object sender, EventArgs e)
        {
            comm.WriteData(txtSend.Text);
        }

        privatevoid rdoHex_CheckedChanged(object sender, EventArgs e)
        {
        if (rdoHex.Checked == true)
            {
                comm.CurrentTransmissionType = PCComm.CommunicationManager.TransmissionType.Hex;
            }
        else
            {
                comm.CurrentTransmissionType = PCComm.CommunicationManager.TransmissionType.Text;
            }
        }

        privatevoid timer1_Tick(object sender, EventArgs e)
        {
        switch(poll)
            {
        case 0:
        if (checkBox1.Checked)
                    {
                        comm.WriteData("gyro1$0");

                        lineChart1.Refresh();
                    }

            poll++;
        break;


        case 1:
        if (checkBox3.Checked)
            {
                comm.WriteData("accel$0");

                lineChart3.Refresh();
            }
            poll++;
        break;

        case 2:
        if (checkBox2.Checked)
            {
                comm.WriteData("stick$0");

                lineChart2.Refresh();
            }

            poll++;
        break;
        case 3:
        if (checkBox4.Checked)
```

```
            {
                comm.WriteData("lmotor##");
this.progressBar1.Value = comm._motore.Value;
this.progressBar1.Refresh();
            }
           poll++;

break;


case 4:
if (checkBox4.Checked)
            {
                comm.WriteData("rmotor##");
this.progressBar2.Value = comm._motord.Value;
this.progressBar2.Refresh();
            }
            poll ++;
break;

case 5:
if (checkBox5.Checked)
            {
                comm.WriteData("angulo##");
                label8.Text = ((int)comm.angle).ToString();
            }
            poll ++;
break;

case 6:
// comm.WriteData("dtmr##");
// label8.Text = ((int)comm.angle).ToString();
            poll = 0;
break;
        }

        }

privatevoid cmdClose_Click(object sender, EventArgs e)
        {
            timer1.Enabled = !timer1.Enabled;

        }

privatevoid button1_Click(object sender, EventArgs e)
        {
            timer1.Enabled = false;

            cmdClose.Enabled = false;
            cmdSend.Enabled = false;
            cmdOpen.Enabled = true;
            comm.ClosePort()
      }
    }
}
```

Apêndice C – Listagem de CommunicationManager.cs

```
using System;
using System.Text;
using System.Drawing;
using System.IO.Ports;
using System.Windows.Forms;
using System.Collections.Generic;
using SeabeckEnterprises.SeabeckCharts;
//*********************************************************************************
//                          LICENSE INFORMATION ABOUT COMMUNICATION MANAGER
//*********************************************************************************
//   PCCom.SerialCommunication Version 1.0.0.0
//   Class file for managing serial port communication
//
//   Copyright (C) 2007
//   Richard L. McCutchen
//   Email: richard@psychocoder.net
//   Created: 20OCT07
//
//   This program is free software: you can redistribute it and/or modify
//   it under the terms of the GNU General Public License as published by
//   the Free Software Foundation, either version 3 of the License, or
//   (at your option) any later version.
//
//   This program is distributed in the hope that it will be useful,
//   but WITHOUT ANY WARRANTY; without even the implied warranty of
//   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//   GNU General Public License for more details.
//
//   You should have received a copy of the GNU General Public License
//   along with this program.  If not, see <http://www.gnu.org/licenses/>.
//*********************************************************************************
//*********************************************************************************//**************
CÓDIGO DO SOFTWARE DE MONITORAMENTO DO FARA-LÉGUAS INSERIDO //DENTRO DA CLASSE
namespace PCComm
{
classCommunicationManager
    {
        #region Manager Enums
///<summary>
/// enumeration to hold our transmission types
///</summary>
publicenumTransmissionType { Text, Hex }

privateint _gyro1 = 0;
privateint _gyro2 = 0;
privateint _accel = 0;
privateint _rmotor = 0;
privateint dtmr = 0;
privateint rmotor = 0;
privateint lmotor = 0;
publicint angle = 0;
privateint amostrag1 = 0;
privateint amostrag2 = 0;
privateint amostraac = 0;
privateint amostramo = 0;
privatedouble[] gyro1 = newdouble [101];
privatedouble[] gyro2 = newdouble[101];
privatedouble[] accel = newdouble[101];
privatedouble[] motor_d = newdouble[101];
///<summary>
/// enumeration to hold our message types
///</summary>
publicenumMessageType { Incoming, Outgoing, Normal, Warning, Error };
        #endregion

        #region Manager Variables
//property variables
privatestring _baudRate = string.Empty;
```

```
privatestring _parity = string.Empty;
privatestring _stopBits = string.Empty;
privatestring _dataBits = string.Empty;
privatestring _portName = string.Empty;
publicProgressBar _motord = newProgressBar();
publicProgressBar _motore = newProgressBar();
privateTransmissionType _transType;
privateRichTextBox _displayWindow;
private SeabeckEnterprises.SeabeckCharts.LineChart _grafico;
private SeabeckEnterprises.SeabeckCharts.LineChart _grafico2;
private SeabeckEnterprises.SeabeckCharts.LineChart _grafico3;
private SeabeckEnterprises.SeabeckCharts.LineChart _grafico4;
//global manager variables
privateColor[] MessageColor = { Color.Blue, Color.Green, Color.Black, Color.Orange, Color.Red };
privateSerialPort comPort = newSerialPort();
        #endregion

        #region Manager Properties
///<summary>
/// Property to hold the BaudRate
/// of our manager class
///</summary>
publicstring BaudRate
        {
get { return _baudRate; }
set { _baudRate = value; }
        }

///<summary>
/// property to hold the Parity
/// of our manager class
///</summary>
publicstring Parity
        {
get { return _parity; }
set { _parity = value; }
        }

///<summary>
/// property to hold the StopBits
/// of our manager class
///</summary>
publicstring StopBits
        {
get { return _stopBits; }
set { _stopBits = value; }
        }

///<summary>
/// property to hold the DataBits
/// of our manager class
///</summary>
publicstring DataBits
        {
get { return _dataBits; }
set { _dataBits = value; }
        }

///<summary>
/// property to hold the PortName
/// of our manager class
///</summary>
publicstring PortName
        {
get { return _portName; }
set { _portName = value; }
        }

///<summary>
/// property to hold our TransmissionType
/// of our manager class
```

```
        ///</summary>
        publicTransmissionType CurrentTransmissionType
        {
        get { return _transType; }
        set { _transType = value; }
        }

        ///<summary>
        /// property to hold our display window
        /// value
        ///</summary>
        publicRichTextBox DisplayWindow
        {
        get { return _displayWindow; }
        set { _displayWindow = value; }
        }

        public SeabeckEnterprises.SeabeckCharts.LineChart Grafico
        {
        get { return _grafico; }
        set { _grafico = value; }
        }

        public SeabeckEnterprises.SeabeckCharts.LineChart Grafico2
        {
        get { return _grafico2; }
        set { _grafico2 = value; }
        }

        public SeabeckEnterprises.SeabeckCharts.LineChart Grafico3
        {
        get { return _grafico3; }
        set { _grafico3 = value; }
        }

        public SeabeckEnterprises.SeabeckCharts.LineChart Grafico4
        {
        get { return _grafico4; }
        set { _grafico4 = value; }
        }

        publicProgressBar Motord
        {
        get { return _motord; }
        set { _motord = value; }
        }

        #endregion

        #region Manager Constructors
        ///<summary>
        /// Constructor to set the properties of our Manager Class
        ///</summary>
        ///<param name="baud">Desired BaudRate</param>
        ///<param name="par">Desired Parity</param>
        ///<param name="sBits">Desired StopBits</param>
        ///<param name="dBits">Desired DataBits</param>
        ///<param name="name">Desired PortName</param>
        public CommunicationManager(string baud, string par, string sBits, string dBits, string name,
        RichTextBox rtb)
        {
            _baudRate = baud;
            _parity = par;
            _stopBits = sBits;
            _dataBits = dBits;
            _portName = name;
            _displayWindow = rtb;
        //now add an event handler
            comPort.DataReceived += newSerialDataReceivedEventHandler(comPort_DataReceived);
        }
```

```
///<summary>
/// Constructor to set the properties of our
/// serial port communicator to nothing
///</summary>
public CommunicationManager()
        {
            _baudRate = string.Empty;
            _parity = string.Empty;
            _stopBits = string.Empty;
            _dataBits = string.Empty;
            _portName = "COM4";
            _displayWindow = null;
            _motord.Minimum = 0;
            _motord.Maximum = 1024;
            _motore.Minimum = 0;
            _motore.Maximum = 1024;
//add event handler
            comPort.DataReceived += new SerialDataReceivedEventHandler(comPort_DataReceived);
        }
        #endregion

        #region WriteData
public void WriteData(string msg)
        {
switch (CurrentTransmissionType)
            {
case TransmissionType.Text:
//first make sure the port is open
//if its not open then open it
if (!(comPort.IsOpen == true)) comPort.Open();
//send the message to the port
                    comPort.Write(msg);
//display the message
                    DisplayData(MessageType.Outgoing, msg + "\n");
break;
case TransmissionType.Hex:
try
                    {
//convert the message to byte array
byte[] newMsg = HexToByte(msg);
//send the message to the port
                        comPort.Write(newMsg, 0, newMsg.Length);
//convert back to hex and display
                        DisplayData(MessageType.Outgoing, ByteToHex(newMsg) + "\n");
                    }
catch (FormatException ex)
                    {
//display error message
                        DisplayData(MessageType.Error, ex.Message);
                    }
finally
                    {
                        _displayWindow.SelectAll();
                    }
break;
default:
//first make sure the port is open
//if its not open then open it
if (!(comPort.IsOpen == true)) comPort.Open();
//send the message to the port
                    comPort.Write(msg);
//display the message
                    DisplayData(MessageType.Outgoing, msg + "\n");
break;

            }
        }
        #endregion

        #region HexToByte
///<summary>
```

```
/// method to convert hex string into a byte array
///</summary>
///<param name="msg">string to convert</param>
///<returns>a byte array</returns>
privatebyte[] HexToByte(string msg)
        {
//remove any spaces from the string
            msg = msg.Replace(" ", "");
//create a byte array the length of the
//divided by 2 (Hex is 2 characters in length)
byte[] comBuffer = newbyte[msg.Length / 2];
//loop through the length of the provided string
for (int i = 0; i < msg.Length; i += 2)
//convert each set of 2 characters to a byte
//and add to the array
                comBuffer[i / 2] = (byte)Convert.ToByte(msg.Substring(i, 2), 16);
//return the array
return comBuffer;
        }
        #endregion

        #region ByteToHex
///<summary>
/// method to convert a byte array into a hex string
///</summary>
///<param name="comByte">byte array to convert</param>
///<returns>a hex string</returns>
privatestring ByteToHex(byte[] comByte)
        {
//create a new StringBuilder object
StringBuilder builder = newStringBuilder(comByte.Length * 3);
//loop through each byte in the array
foreach (byte data in comByte)
//convert the byte to a string and add to the stringbuilder
                builder.Append(Convert.ToString(data, 16).PadLeft(2, '0').PadRight(3, ' '));
//return the converted value
return builder.ToString().ToUpper();
        }
        #endregion

        #region DisplayData
///<summary>
/// method to display the data to & from the port
/// on the screen
///</summary>
///<param name="type">MessageType of the message</param>
///<param name="msg">Message to display</param>
        [STAThread]
privatevoid DisplayData(MessageType type, string msg)
        {
            _displayWindow.Invoke(newEventHandler(delegate
        {
            _displayWindow.SelectedText = string.Empty;
            _displayWindow.SelectionFont = newFont(_displayWindow.SelectionFont, FontStyle.Bold);
            _displayWindow.SelectionColor = MessageColor[(int)type];
            _displayWindow.AppendText(msg);
            _displayWindow.ScrollToCaret();
        }));
        }
        #endregion
        #region ClosePort
publicbool ClosePort()
        {
try
            {
                comPort.Close();
returntrue;
            }
catch (Exception ex)
            {
                DisplayData(MessageType.Error, ex.Message);
```

```
returnfalse;
            }
        }
        #endregion

        #region OpenPort
publicbool OpenPort()
        {
try
            {
//first check if the port is already open
//if its open then close it
if (comPort.IsOpen == true) comPort.Close();

//set the properties of our SerialPort Object
                comPort.BaudRate = int.Parse(_baudRate);    //BaudRate
                comPort.DataBits = int.Parse(_dataBits);    //DataBits
                comPort.StopBits = (StopBits)Enum.Parse(typeof(StopBits), _stopBits);
//StopBits
                comPort.Parity = (Parity)Enum.Parse(typeof(Parity), _parity);    //Parity
                comPort.PortName = _portName;   //PortName
//now open the port
                comPort.Open();
//display message
                DisplayData(MessageType.Normal, "Port opened at " + DateTime.Now + "\n");
//return true
returntrue;
            }
catch (Exception ex)
            {
                DisplayData(MessageType.Error, ex.Message);
returnfalse;
            }
        }
        #endregion

        #region SetParityValues
publicvoid SetParityValues(object obj)
        {
foreach (string str inEnum.GetNames(typeof(Parity)))
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        #endregion

        #region SetStopBitValues
publicvoid SetStopBitValues(object obj)
        {
foreach (string str inEnum.GetNames(typeof(StopBits)))
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        #endregion

        #region SetPortNameValues
publicvoid SetPortNameValues(object obj)
        {

foreach (string str inSerialPort.GetPortNames())
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        #endregion

        #region comPort_DataReceived
///<summary>
/// method that will be called when theres data waiting in the buffer
///</summary>
```

```
///<param name="sender"></param>
///<param name="e"></param>
void comPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {


//determine the mode the user selected (binary/string)
switch (CurrentTransmissionType)
            {

//user chose string
caseTransmissionType.Text:
//read data waiting in the buffer
string msg = comPort.ReadExisting();
                    DisplayData(MessageType.Incoming, msg);
//display the data to the user
if (msg.Length > 0)
                    {
switch ((char)msg[0])
                        {
case'g':


                                DisplayData(MessageType.Incoming, msg);

if (msg.Length > 2)
                                {
                                    _gyro1 = (int)((uint)msg[1] << 5);
                                    _gyro1 = _gyro1 | (int)((uint)msg[2]);

                                    gyro1[amostrag1] = _gyro1;
                                    amostrag1++;
if (amostrag1 == 100)
                                    { amostrag1 = 0; }
                                    Grafico.RemoveSeries("Gyro1");

                                    Grafico.AddSeries("Gyro1", Color.Lime, 1.5f, false, true,
TickMark.Dot, gyro1);



                                }
                                msg = String.Empty;
break;



case'h':

DisplayData(MessageType.Incoming, "Sensor \n");
// msg = comPort.ReadExisting();
if (msg.Length > 2)
                                {
                                    _gyro2 = (int)((uint)msg[1] << 5);
                                    _gyro2 = _gyro2 | (int)((uint)msg[2]);

gyro2[amostrag2] = _gyro2;
                                    amostrag2++;
if (amostrag2 == 100)
                                    { amostrag2 = 0; }
                                    Grafico2.RemoveSeries("Gyro2");

Grafico2.AddSeries("Gyro2", Color.Lime, 1.5f, false, true, TickMark.Dot, gyro2);



                                }
```

```
                                msg = String.Empty;
break;


case'a':

//msg = comPort.ReadExisting();


                                DisplayData(MessageType.Incoming, msg);
if (msg.Length > 2)
                                {
                                    _accel = (int)((uint)msg[1] << 5);
                                    _accel = _accel | (int)((uint)msg[2]);

                                    accel[amostraac] = _accel;
                                    amostraac++;
if (amostraac == 100)
                                    { amostraac = 0; }
                                    Grafico3.RemoveSeries("Accel");

                                    Grafico3.AddSeries("Accel", Color.Lime, 1.5f, false, true,
TickMark.Dot, accel);



                                }
                                msg = String.Empty;
break;

case'd':


                                DisplayData(MessageType.Incoming, "DTMR: \n");
//msg = comPort.ReadExisting();
if (msg.Length > 1)
                                {

                                    dtmr = (int)((uint)msg[1] << 8);
                                    dtmr = dtmr | (int)((uint)msg[2]);


                                }
                                DisplayData(MessageType.Incoming, dtmr.ToString());
                                msg = String.Empty;
break;


case'v':


                                DisplayData(MessageType.Incoming, "Angulo: \n");
// msg = comPort.ReadExisting();



if (msg.Length > 2)
                                {

                                    angle = (int)(msg[1] << 5);

if (msg.Length > 2)
                                    {
                                        angle = angle + (int)(msg[2]);
                                    }
```

```
if (angle >= 512)
                                    {
                                        angle = angle - 512;
                                        angle = -angle;
                                    }
//angle = (int)ang - 1024;
/* int gz_vel = (_accel - 512) * 14 >> 5; ;
                                     int ax_deg = (_gyrol - 512) * 14 >> 5; ;

                                     angle = 0.97 * (angle + (double)gz_vel * (double)dtmr *
0.000128);
                                     angle += 0.03 * ((double)ax_deg);*/

                                    DisplayData(MessageType.Incoming, angle.ToString()); msg =
comPort.ReadExisting();
                                }

                                msg = String.Empty;
break;



case'r':

DisplayData(MessageType.Incoming, "Motor direito \n");

if (msg.Length > 2)
                                {
                                    rmotor = (int)((uint)msg[1] << 5);
                                    rmotor = rmotor | (int)((uint)msg[2]);

if (rmotor > 1024)
                                    {
                                        rmotor = 1024;
                                    }
                                        _motord.Value = rmotor;


}
                                motor_d[amostramo] = _rmotor;
                                amostramo++;
if (amostramo == 100)
                                { amostramo = 0; }
                                Grafico3.RemoveSeries("Motor");

Grafico3.AddSeries("Motor", Color.Lime, 1.5f, false, true, TickMark.Dot, accel);



                                msg = String.Empty;
break;



case'l':

DisplayData(MessageType.Incoming, "Motor Esquerdo \n");

if (msg.Length > 2)
                                {
```

```
                                lmotor = (int)((uint)msg[1] << 5);
                                lmotor = lmotor | (int)((uint)msg[2]);

 if (lmotor > 1024)
                                {
                                    lmotor = 1024;
                                }
                                    _motore.Value = lmotor;
                            }

                            msg = String.Empty;

 break;


                        }

                    }
 break;
 //user chose binary
 caseTransmissionType.Hex:
 //retrieve number of bytes in the buffer
 int bytes = comPort.BytesToRead;
 //create a byte array to hold the awaiting data
 byte[] comBuffer = newbyte[bytes];
 //read the data and store it
                    comPort.Read(comBuffer, 0, bytes);
 //display the data to the user
                    DisplayData(MessageType.Incoming, ByteToHex(comBuffer) + "\n");
 break;
 default:
 //read data waiting in the buffer
 string str = comPort.ReadExisting();
 //display the data to the user
                    DisplayData(MessageType.Incoming, str + "\n");
 break;
            }
        }
        #endregion
    }

}
```

Apêndice 4 – Listagem TinyBld - Bootloader

```
radix DEC
        LIST      P=18F4331     ; change also: Configure->Select Device from MPLAB
 xtal EQU 20000000               ; you may want to change: _XT_OSC_1H  _HS_OSC_1H  _HSPLL_OSC_1H

baud EQU 19200          ; standard TinyBld baud rates: 115200 or 19200
        ; The above 3 lines can be changed and built a bootloader for the desired frequency (and
PIC type)

        ;*****************************************************************
        ;       Tiny Bootloader                 18F series              Size=100words
        ;       claudiu.chiculita@ugal.ro
        ;       http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm
        ;       Modified by Nam Nguyen-Quang for testing different PIC18Fs with tinybldWin.exe
v1.9
        ;       namqn@yahoo.com
        ;*****************************************************************

;       This source file is for PIC18F242, 252, 442, 452, 248, 258, 448, 458, 2220, 2320,
```

```
;       4220, 4320, 1220, 1320, 2331, 2431, 4331, 4431, 2439, 2539, 4439, and 4539

;       Copy these include files to your project directory (i.e. they are in the same
;       directory with your .asm source file), if necessary

        #include "icdpictypes.inc"     ; Takes care of: #include "p18fxxx.inc",  max_flash,
IdTypePIC
        #include "spbrgselect.inc"     ; RoundResult and baud_rate

                #define first_address max_flash-200          ;100 words

;       For different PICs, uncomment the appropriate lines of CONFIG directives
;       as indicated, and comment out all the other lines, if necessary
;       For example, the following configuration is for PIC18F4580, with 8 MHz crystal
;       You could find the symbol names for the chip in its include file
;       (in the Microchip\MPASM Suite directory)


;----- CONFIG1H Options -----
;       For 18F242, 248, 252, 258, 442, 448, 452, and 458 (xx2/xx8)
;               CONFIG  OSC = HS, OSCS = OFF

;       For 18F1220, 1320, 2220, 2320, 4220, and 4320 (x220/x320)
;               CONFIG  OSC = HS, FSCM = OFF, IESO = OFF
;               CONFIG  OSC = INTIO2, FSCM = OFF, IESO = OFF ; Use internal oscilator, xtal =
8000000

;       For 18F2331, 2431, 4331, and 4431
;               CONFIG  OSC = HS, FCMEN = OFF, IESO = OFF
;               CONFIG  OSC = IRCIO, FCMEN = OFF, IESO = OFF

;       For 18F2439, 2539, 4439, and 4539
;               CONFIG  OSC = HS

;       For 2480, 2580, 4680, and 4580
;               CONFIG  OSC = HS, FCMENB = OFF, IESOB = OFF
;               CONFIG  OSC = IRCIO67, FCMENB = OFF, IESOB = OFF      ; Use internal oscilator,
xtal = 8000000


;----- CONFIG2L Options -----
;       For 18F242, 248, 252, 258, 442, 448, 452, and 458 (xx2/xx8)
;       For 18F1220, 1320, 2220, 2320, 4220, and 4320 (x220/x320)
;       and for 18F2439, 2539, 4439, and 4539 as well
;               CONFIG  PWRT = ON, BOR = ON, BORV = 27

;       For 18F2331, 2431, 4331, and 4431
                CONFIG  PWRTEN = ON, BOREN = ON, BORV = 27

;       For 2480, 2580, 4680, and 4580
;               CONFIG  PWRT = ON, BOR = BOHW, BORV = 27


;----- CONFIG2H Options -----
;       For all of the chips associated with this source file, except xx31 as follows
;               CONFIG  WDT = OFF, WDTPS = 128

;       For 18F2331, 2431, 4331, and 4431
                CONFIG  WDTEN = OFF, WINEN = OFF, WDPS = 128

;----- CONFIG3L Options -----
;       For 18F2331, 2431, 4331, and 4431
                CONFIG  T1OSCMX = ON, HPOL = HIGH, LPOL = HIGH, PWMPIN = OFF


;----- CONFIG3H Options -----
;       For 18F242, 252, 442, and 452
;               CONFIG  CCP2MUX = OFF

;       For 18F1220, 1320, 2220, 2320, 4220, and 4320
;               CONFIG  MCLRE = ON, PBAD = DIG, CCP2MX = OFF
```

```
;       For 18F2331, and 2431
;             CONFIG  MCLRE = ON

;       For 18F4331, and 4431
;             CONFIG  MCLRE = ON, EXCLKMX = RD0, PWM4MX = RD5, SSPMX = RD1, FLTAMX = RD4

;       For 2480, 2580, 4480, and 4580
;             CONFIG  MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF


;----- CONFIG4L Options -----
;       For 18F242, 248, 252, 258, 442, 448, 452, and 458
;       For 18F1220, 1320, 2220, 2320, 4220, and 4320
;       and for 18F2439, 2539, 4439, and 4539 as well
;             CONFIG  STVR = ON, LVP = OFF, DEBUG = OFF

;       For 18F2331, 2431, 4331, and 4431
;             CONFIG  STVREN = ON, LVP = OFF, DEBUG = OFF

;       For 2480, 2580, 4480, and 4580
;             CONFIG  STVREN = ON, LVP = OFF, DEBUG = OFF, XINST = OFF, BBSIZ = 1024


;------------------------------ PROGRAM ---------------------------------
        cblock 0
        crc
        i
        cnt1
        cnt2
        cnt3
        counter_hi
        counter_lo
        flag
        endc
        cblock 10
        buffer:64
        dummy4crc
        endc

SendL macro car
        movlw car
        movwf TXREG
        endm

;000000000000000000000000 RESET 000000000000000000000000

              ORG     0x0000
              GOTO    IntrareBootloader

;view with TabSize=4
;&&&&&&&&&&&&&&&&&&&&&&&&   START     &&&&&&&&&&&&&&&&&&&&&&&&
;----------------------- Bootloader -----------------------
;PC_flash:             C1h                             U               H               L
      K  ...  <64 bytes>   ...  crc
;PC_eeprom:            C1h                             40h     EEADR   EEDATA  0               crc

;PC_cfg                C1h                     U OR 80h        H               L               1
        byte    crc
;PIC_response:    type 'K'

        ORG first_address               ;space to deposit first 4 instr. of user prog.
        nop
        nop
        nop
        nop
        org first_address+8
IntrareBootloader
                                                ;init IntOSC, added by Nam Nguyen-Quang
        ;movlw 0x70
        ;movwf OSCCON
```

```
        ; the above 2 lines should be commented out for designs not using the internal oscilator
        ; or for the chips without the internal oscilator
                                                ;init serial port
        movlw b'00100100'
        movwf TXSTA
        movlw spbrg_value
        movwf SPBRG
        movlw b'10010000'
        movwf RCSTA

                                                ;wait for computer
        rcall Receive
        sublw 0xC1                      ;Expect C1h
        bnz way_to_exit
        SendL IdTypePIC                 ;send PIC type
MainLoop
        SendL 'K'                       ; "-Everything OK, ready and waiting."
mainl
        clrf crc
        rcall Receive           ;Upper
        movwf TBLPTRU
             movwf flag                 ;(for EEPROM and CFG cases)
        rcall Receive           ;Hi
        movwf TBLPTRH
             movwf EEADR                ;(for EEPROM case)
        rcall Receive           ;Lo
        movwf TBLPTRL
             movwf EEDATA       ;(for EEPROM case)

        rcall Receive           ;count
        movwf i
        incf i
        lfsr FSR0, (buffer-1)
rcvoct
             movwf TABLAT               ;read 64+1 bytes
        rcall Receive           ;prepare for cfg; => store byte before crc
        movwf PREINC0
        decfsz i
        bra rcvoct

        tstfsz crc                      ;check crc
        bra ziieroare
             btfss flag,6       ;is EEPROM data?
             bra noeeprom
             movlw b'00000100'  ;Setup eeprom
             rcall Write
             bra waitwre
noeeprom
             btfss flag,7       ;is CFG data?
             bra noconfig
             tblwt*                     ;write TABLAT(byte before crc) to TBLPTR***
             movlw b'11000100'  ;Setup cfg
             rcall Write
             bra waitwre
noconfig

                                                ;write
eraseloop
       movlw  b'10010100'       ; Setup erase
       rcall Write
       TBLRD*-                          ; point to adr-1

writebigloop
       movlw 8                          ; 8groups
       movwf counter_hi
       lfsr FSR0,buffer
writesloop
       movlw 8                          ; 8bytes = 4instr
       movwf counter_lo
writebyte
       movf POSTINC0,w                  ; put 1 byte
       movwf TABLAT
       tblwt+*
```

```
        decfsz counter_lo
        bra writebyte

        movlw   b'10000100'             ; Setup writes
        rcall Write
        decfsz counter_hi
        bra writesloop
waitwre
        ;btfsc EECON1,WR                ;for eeprom writes (wait to finish write)
        ;bra waitwre                    ;no need: round trip time with PC bigger than 4ms

        bcf EECON1,WREN                         ;disable writes
        bra MainLoop

ziierosare                                      ;CRC failed
        SendL 'N'
        bra main1

;******** procedures ******************

Write
        movwf EECON1
        movlw 0x55
        movwf EECON2
        movlw 0xAA
        movwf EECON2
        bsf EECON1,WR                   ;WRITE
        nop
        ;nop
        return


Receive
        movlw xtal/2000000+1   ; for 20MHz => 11 => 1second delay
                                                ; for 18F2xxx chips, this should be
xtal/1000000+1
        movwf cnt1
rpt2
        clrf cnt2
rpt3
        clrf cnt3
rptc
              btfss PIR1,RCIF                   ;test RX
              bra notrcv
           movf RCREG,w                         ;return read data in W
           addwf crc,f                          ;compute crc
              return
notrcv
        decfsz cnt3
        bra rptc
        decfsz cnt2
        bra rpt3
        decfsz cnt1
        bra rpt2
        ;timeout:
way_to_exit
        bcf     RCSTA,  SPEN                    ; deactivate UART
        bra first_address
;*****************************************************************
; After reset
; Do not expect the memory to be zero,
; Do not expect registers to be initialised like in catalog.

          END
```

Apêndice D – Desenho 3 Visões