unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

GUSTAVO BUZZATTO SIQUEIRA

TELEMETRIA AUTOMOTIVA USANDO MÓDULO DE TRANSMISSÃO WIRELESS ZIGBEE

Guaratinguetá

2014

GUSTAVO BUZZATTO SIQUEIRA

TELEMETRIA AUTOMOTIVA USANDO MÓDULO DE TRANSMISSÃO WIRELESS ZIGBEE

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. José Feliciano Adami

Guaratinguetá
2014

Siqueira, Gustavo Buzzatto
S618t Telemetria automotiva usando módulo de transmissão Wireless Zigbee/ Gustavo Buzzatto Siqueira – Guaratinguetá : [s.n], 2014.
113 f : il.
Bibliografia: f. 73-74

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2014.
Orientador: Prof. Dr. José Feliciano Adami

1. Microcontroladores 2. Sistemas de comunicação sem fio I. Título

CDU 621.381

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

TELEMETRIA AUTOMOTIVA USANDO MÓDULO DE TRANSMISSÃO WIRELESS ZIGBEE

GUSTAVO BUZZATTO SIQUEIRA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO EM ENGENHARIA ELÉTRICA"**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA MECÂNICA

Prof. Dr. LEONARDO MESQUITA
Coordenador

BANCA EXAMINADORA:

Prof. Dr. JOSÉ FELICIANO ADAMI
Orientador/UNESP-FEG

Prof. Dr. SAMUEL EUZÉDICE DE LUCENA
UNESP-FEG

Prof. Dr. FRANCISCO ANTÔNIO LOTUFO
UNESP-FEG

Dezembro de 2014

GUSTAVO BUZZATTO SIQUEIRA

NASCIMENTO	24.03.1989 - CACHOEIRA PAULISTA / SP
FILIAÇÃO	Fernando Antonio Rosa de Siqueira
	Alcineia Aparecida Buzzatto

dedico primeiramente este trabalho aos meus pais Alcineia Aparecida Buzzatto e Fernando Antonio Rosa de Siqueira que sempre me deram suporte para chegar onde estou hoje. Minha irmã Patrícia Buzzatto Siqueira e minha namorada Vanessa Gonçalves Fortes que souberam me apoiar quando eu precisava, a todos os integrantes, ou ex-integrantes, da Equipe Unesp Racing de Fórmula SAE e por último a todos os meus Irmãos de República IGLU, WCK, amigos da Engenharia Elétrica 08/09 e de infância que sempre estiveram comigo e viram minhas vitórias e derrotas. A todos vocês um muito obrigado a me ajudar a me tornar quem eu sou, Engenheiro Eletricista.

E FOI REALMENTE MUITO BOM TER IDO PRA IGLU...VAI IGLU!

AGRADECIMENTO

Agradeço o grande apoio do meu professor orientador, Dr. José Feliciano e ao professor Dr. Samuel pelo apoio e orientações na realização neste trabalho, aos amigos de Unesp Racing pelo aprendizado adquirido nestes 5 anos de equipe, também ao Capitão do Baja no ano de 2014, Samuel Guarnieri, e da FEG Robótica, Felipe Barbosa, pela troca de informações e paciência ao me ouvir. E em especial aos Srs. e caros colegas Pakita, Ceará, Bonsai, CG, Xitão, que nestes últimos dois anos se tornaram meus irmãos durante esta jornada na *Escola de Engenharia lá de Guará*.

"*Para conseguir a amizade de uma pessoa digna é preciso desenvolvermos em nós mesmos as qualidades que naquela admiramos."*

Sócrates

"*Você tem de assumir o compromisso de vencer, ou então nada.*"

Ayrton Senna

SIQUEIRA, G. B. **Telemetria automotiva usando módulo de transmissão wireless zigbee.** 2014.113 f. Trabalho de Graduação (Graduação em Engenharia Elétrica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2014

RESUMO

Este trabalho apresenta o projeto e a construção de um sistema de aquisição de dados com transmissão via *wireless* de alguns parâmetros importantes, como temperatura do motor, rotação, velocidade e outros do protótipo de um carro tipo fórmula, da equipe Unesp Racing da Universidade Estadual Paulista, Campus de Guaratinguetá. O desenvolvimento deste sistema apresenta uma contribuição para a equipe somar pontos importantes na competição com relação às características elétricas do carro, pois este protótipo foi desenvolvido dentro da própria faculdade. Além disso, possibilita manutenções preventivas e ajustes mais refinados do carro com os dados obtidos durante os treinamentos e corridas.

PALAVRAS-CHAVE: *ZigBee,* microcontrolador, aquisição de dados, telemetria.

SIQUEIRA, G. B**. Automotive telemetry using zigbee wireless module**. 2014. 113 f. Graduate Work (Graduate in Electrical Engeneering) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2014.

ABSTRACT

This paper presents a design and construct the system of data acquisition with wireless transmission of some important parameters like motor temperature, motor rotation and velocity of Unesp Racing´s formula car prototype from Universidade Estadual Paulista, Campus Guaratinguetá. This system development presents a contribution for the electrical technical features adding important points in the competition due car´s electric, in addition, enables preventive maintenance and fine adjustment of the car with the data obtained during training and racing.

KEYWORDS: ZigBee microcontroller, data acquisition, telemetry

.

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico/Digital
API	Application Programming Interface
CCP	Capture Compare Pulse
CI	Circuito Integrado
ECPA	Esporte Clube Piracicabano de Automobilismo
FFD	Full Function Device
GUIDE	Graphical User Interface Development Environment
HID	Human Interface Device
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
LED	Light Emitting Diode
PAN	Personal Area Network
PCB	Printed Circuit Board
PIC	Programmable Intelligent Computer
RC	Oscilador à Resistor e Capacitor
RF	Radio Frequência
RFD	Reduced Function Device
Rx	Receptor
SAE	Society of Automotive Engineers
Tx	Transmissor
UART	Universal Asynchronous Receiver Transmission
USB	Universal Serial Bus
XT	Oscilador à Cristal

LISTA DE SÍMBOLOS

Ω	**Ohm**
kΩ	**kilo Ohm**
MΩ	**Mega Ohm**
pF	**pico Faraday**
nF	**nano Faraday**
uF	**micro Faraday**
m	**metro**
km	**kilometro**
s	**segundo**
m/s	**metros por segundo**
km/h	**kilometros por hora**
Hz	**Hertz**
kHz	**kilo Hertz**
MHz	**Mega Hertz**
GHz	**Giga Hertz**
V	**Volts**
bps	**bits por segundo**
kbps	**kilo bits por segundo**
Mbps	**Mega bits por segundo**
W	**Watts**
mW	**mili Watts**
°	**Graus**
dB	**Decibel**

SUMÁRIO

1. INTRODUÇÃO

A equipe de Fórmula SAE (*"Society of Automotive Engineers"*) da Universidade Estadual Paulista do Campus de Guaratinguetá, denominada Unesp Racing, foi fundada em 2008 pelos alunos da própria faculdade, aplicando os conhecimentos adquiridos em seus respectivos cursos (Engenharia Elétrica, Mecânica, Produção e Materiais) para projetar e conceber um veículo protótipo tipo fórmula para competir com demais faculdades e universidades de todo o Brasil.

Uma competição é realizada anualmente na cidade de Piracicaba, interior do estado de São Paulo, no autódromo do Esporte Clube Piracicabano de Automobilismo, (**ECPA)**, por voluntários, mas muito bem capacitados. A competição reúne provas de consumo de combustível, custos envolvidos no desenvolvimento do veículo e projetos de marketing, além das tão esperadas provas dinâmicas, passando ainda por rigorosas provas de segurança para obter o direito de competir nestas provas.

Desde então a equipe vem evoluindo em todas as suas áreas, e a elétrica é uma delas, sendo uma das áreas que mais pontua, entretanto tem deixado de obter mais pontos em seu projeto por falta de um sistema de transmissão de dados em tempo real desenvolvido pela própria equipe. Este trabalho tem como objetivo apresentar o desenvolvimento de um sistema de telemetria para ser utilizado no carro. Com isso, além de somar mais pontos na competição para a equipe da UNESP, possibilita também ter mais informações dos parâmetros do veículo para melhorar a qualidade dos ajustes finos do mesmo e realizar manutenções preventivas ou corretivas do veículo. Tudo isso foi alcançado com a inclusão deste sistema de telemetria no sistema elétrico do protótipo.

Neste trabalho são apresentadas algumas características dos componentes que compõem o sistema de telemetria, como Microcontrolador, o ZigBee, Cristal Ressonador, e como são os sinais adquirido pelo sistema. Também são mostrados ambientes de desenvolvimento, como MATLAB, MPLAB, ISIS e ARES, explicando o que cada um deles é capaz de fazer, assim como o que foi desenvolvido passo a passo. Passando também por uma breve explicação do módulo de transmissão, apresentando suas topologias, e funcionalidades e mostrando detalhadamente a confecção da placa de circuito impressa utilizada neste sistema de telemetria, levando-se em conta projeto enlace de rádio para garantir a transmissão dos sinais ao computador que irá adquirir os

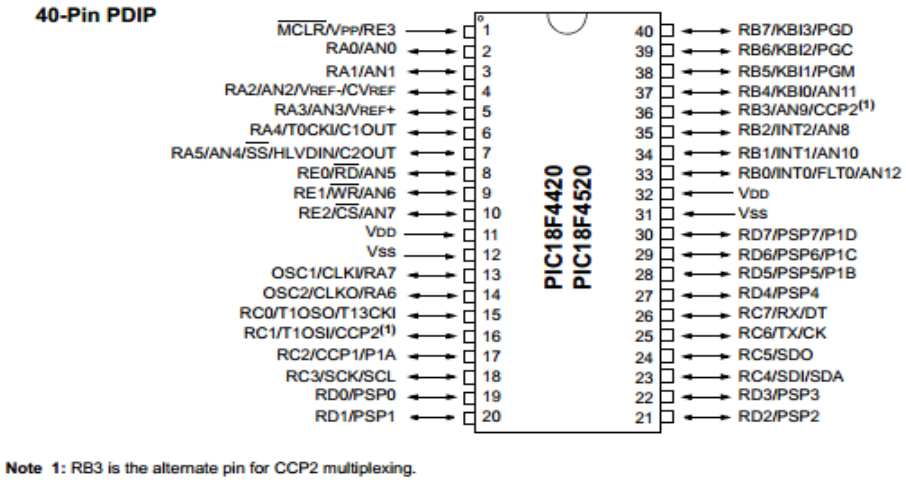
dados. Por fim são apresentados os testes dos dados transmitidos e captados no veiculo e a respectiva conclusão do trabalho.

2. MICROCONTROLADOR

O Microcontrolador é um dispositivo eletrônico usado para controle de sistemas periféricos a ele, que dependendo do nível lógico (**um** ou **zero**) do sinal recebido ou enviado pelo mesmo. Irá atuar de maneiras diferentes sobre o periférico em questão, dependendo ainda da lógica atribuída ao sistema. E este sistema somente irá funcionar a partir de um oscilador que irá atuar como um ***clock*** para que o microcontrolador possa executar todas as suas instruções do programa.

O microcontrolador utilizado neste trabalho é o **PIC18F4520**. A figura 1 mostra sua configuração:

Figura 1: Pinagem PIC 18F4520



Fonte: (MICROCHIP, 2008)

A funcionalidade dos pinos deste microcontrolador é apresentada na tabela 1.

Tabela 1: Funcionalidade dos pinos do PIC 18F4520

PINO	FUNÇÃO	TIPO	FUNCIONALIDADE
1	/MCLR / VPP	In-In	Reset externo e programação ICSP
2	RA0 / AN0	I/O e input A/D	I/O digital e entrada do AD0
3	RA1 / AN1	I/O e input A/D	I/O digital e entrada do AD1
4	RA2 / AN2/ Vref-	I/O e input A/D	I/O digital e entrada do AD1
5	RA3/AN3/ Vref+	I/O e input A/D	I/O digital, entrada do AD3 e entrada de referência alta do A/D
6	RA4 / **T0CKI**	I/O e Input TMR0	I/O digital e entrada do TMR0
7	RA5/AN4/SS/LVDIN	I/O e Inputs	I/O digital, entrada do AD4, entrada do SPI e Detector de LV
8	RE0 / RD / AN5	Fonte	I/O digital, Leitura da Porta Paralela e entrada do AD5
9	RE1 / WR / AN6	Fonte	I/O digital, Escrita da Porta Paralela e entrada do AD6
10	RE2 / CS / AN7	Fonte	I/O digital, Seleção da Porta Paralela e entrada do AD7
11,32	VCC	Fonte	Positivo da Fonte de Alimentação
12,31	GND	Fonte	Negativo da Fonte de Alimentação
13	OSC1 / CLK1	Input	Entrada do Cristal e entrada do Clock externo
14	Osc2 / CLK1 / **RA6**	I/O e Inputs	I/O digital, Saída do Cristal e saída do Clock externo
15	RC0/T10S0/T1Ck1	I/O Out e In	I/O digital, saída do 2° oscilador e entrada do contador externo Timer1/Timer3
16	RC1/T10S1/**CCP2**	I/O In e Out	I/O digital, entrada do 2° oscilador e saída do Módulo CCP2
17	RC2 / **CCP1**	I/O e Out	I/O digital e saída do Módulo CCP1
18	Rc3 / SCK / SCL	I/O, I/O e I/O	I/O digital, in e out do Clock serial para modo SPI e in/out do Clock serial para modo I^2C
19	RD0 / PSP0	I/O e I/O	I/O digital e Porta de Comunicação Paralela
20	RD1 / PSP1	I/O e I/O	I/O digital e Porta de Comunicação Paralela
21	RD2 / PSP2	I/O e I/O	I/O digital e Porta de Comunicação Paralela
22	RD2 / PSP3	I/O e I/O	I/O digital e Porta de Comunicação Paralela
23	RC4 / SDI / SDA	I/O e I/O	I/O digital e Porta de Comunicação Paralela
24	RC5 / SD0	I/O e I/O	I/O digital e saída de dados SPI
25	RC6 / TX / CK	I/O e I/O	I/O digital, Transmissão UART e Clock de sincronismo UART
26	RC7 / RX / DT	I/O e I/O	I/O digital, Recepção UART e Dados do UART
27	RD4 / PSP4	I/O e I/O	I/O digital e Porta de Comunicação Paralela
28	RD5 / PSP5	I/O e I/O	I/O digital e Porta de Comunicação Paralela
29	RD6 / PSP6	I/O e I/O	I/O digital e Porta de Comunicação Paralela
30	RD7 / PSP7	I/O e I/O	I/O digital e Porta de Comunicação Paralela
33	RD0 / **INT0**	I/O e In	I/O digital e entrada de Interrupção Externa 0
34	RD0 / **INT1**	I/O e In	I/O digital e entrada de Interrupção Externa 1
35	RD0 / **INT2**	I/O e In	I/O digital e entrada de Interrupção Externa 2
36	RB3 / **CCP2**	I/O e I/O	I/O digital Módulo CCP2
37	RB4	I/O e In	I/O digital e entrada de Interrupção por Mudança de Estado
38	RB5 / PGM	I/O e In	I/O digital, Interrupção por Mudança de Estado e Habilita ICSP baixa tensão
39	RB6 / PGC	I/O e In	I/O digital, Interrupção por Mudança de Estado e ICSP in-circuit Debuger
40	RB7 / PGD	I/O e In	I/O digital, Interrupção por Mudança de Estado e ICSP in-circuit Debuger

Fonte: (MICROCHIP, 2008)

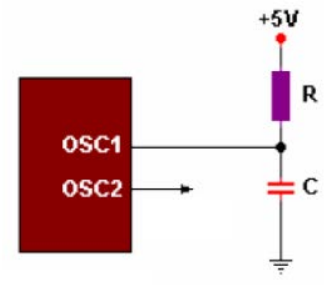
Como mencionado anteriormente, para o correto funcionamento do microcontrolador, é necessária a introdução de um circuito oscilador. Existem alguns tipos de osciladores, neste trabalho são apresentados 2 (**dois**) tipos, o **Oscilador XT** e o **Oscilador RC.**

2.1. OSCILADOR RC

Este oscilador é menos preciso do que o oscilador XT, pois este utiliza uma resistência R, a ser definida. Se R for inferior a 2,2 kΩ, o oscilador pode se tornar instável ou até mesmo parar de oscilar. Caso a resistência R for muito elevada (maior que 1 (MΩ) o oscilador fica muito susceptível a ruídos. A fim de ter uma melhor confiabilidade neste tipo de oscilador é então recomendado uma resistência R entre 3 kΩ e 100 kΩ e um capacitor de 20 pF para aumentar a estabilidade. A figura 2 ilustra este tipo de oscilador.

Este tipo de oscilador funciona a partir da diferença de tensão no ponto de conexão de **OSC1**, como pode ser observador na figura 2. Com o carregamento do capacitor C a diferença de tensão vai diminuindo segundo os valores de R e de C. Após o capacitor C estar totalmente carregado, ele se descarrega iniciando o processo novamente, assim gerando um sinal periódico para o microcontrolador.

Figura 2: Oscilador RC

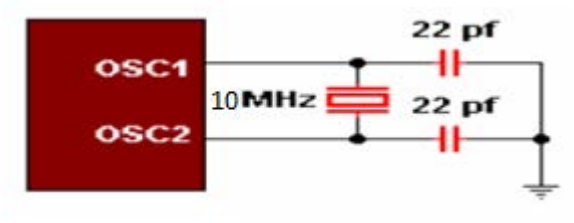


Fonte: (ANTONIO, 2006)

2.2. OSCILADOR XT

É um oscilador à cristal, que possui a seguinte configuração como ilustra a figura 3:

Figura 3: Oscilador XT



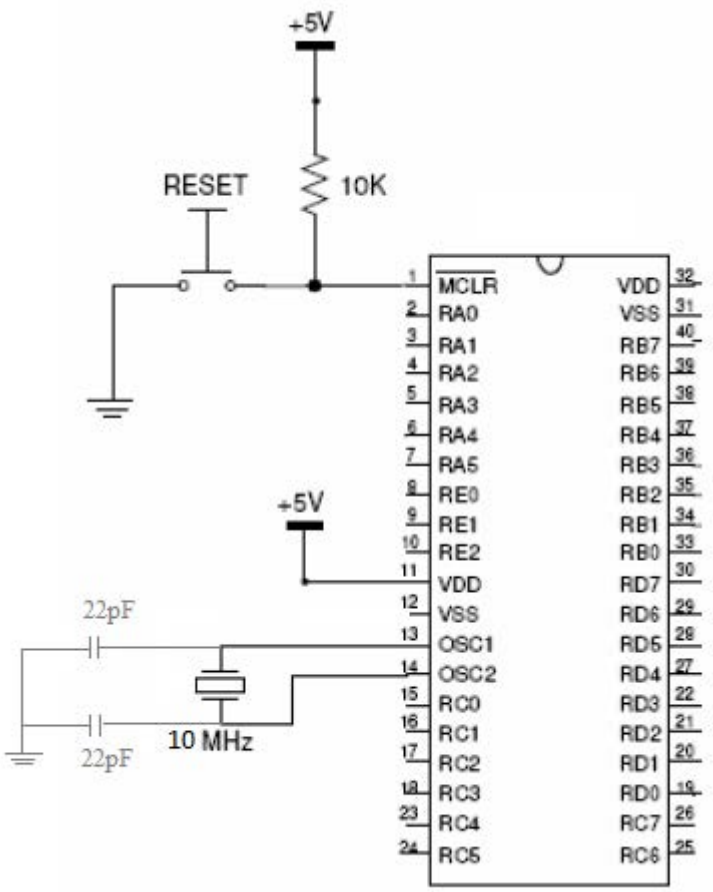
Fonte: (AUTOR, 2014)

Este tipo de oscilador utiliza a ressonância de um cristal de um material piezoeletrônico, como o quartzo, que possui frequência de oscilação bastante precisa quando submetido a uma tensão elétrica, pois ocorre uma deformação mecânica do cristal e quando se atinge uma determinada tensão, o circuito libera esta tensão e em seguida o cristal volta ao seu formato original e volta a se deformar, gerando assim a sua frequência de oscilação, que varia de cristal para cristal conforme seu corte e material piezoeletrônico.

Então com o intuito de se obter um sistema mais estável e confiável, neste trabalho é utilizada uma configuração de oscilador XT cuja frequência de oscilação é de 10 MHz.

Como os osciladores, ao serem alimentados, possuem certa instabilidade por um curto período de tempo e para sanar este pequeno problema deve-se manter o microcontrolador em estado de reset durante o tempo que o clock não se estabiliza. Para isto é usado a configuração mostrada na figura 4:

Figura 4: Esquemático de Reset



Fonte: (AUTOR, 2014)

Com estas informações do microcontrolador, basta agora programá-lo com as lógicas adequadas para este trabalho, elas são responsáveis pelo reconhecimento, tratamento, condicionamento e armazenamento dos sinais que são captados do veículo, como os sinais de Velocidade Instantânea, Rotação do Motor e Temperatura do Motor do protótipo.

3. SISTEMA DE AQUISIÇÃO DE DADOS

3.1. SINAIS PARA AQUISIÇÃO DE DADOS

3.1.1. Velocidade Instantânea

A velocidade instantânea do veiculo é obtida acoplando em uma das rodas dianteiras do veículo um sensor, pois se tal sistema fosse colocado em uma de suas rodas traseiras, o carro poderia “patinar” e assim fornecer uma velocidade diferente da qual o carro realmente se encontra. Este sistema é composto de um sensor indutivo e do disco de freio do carro, o qual está posicionado na frente do sensor e como o disco é um elemento vazado, gera um trem de pulsos a cada volta do disco, conforme a disposição dos furos, como se segue na figura 5. O sinal capturado pelo sistema é digital.

Figura 5: Sensor de Velocidade Instantânea

Fonte: (AUTOR, 2014)

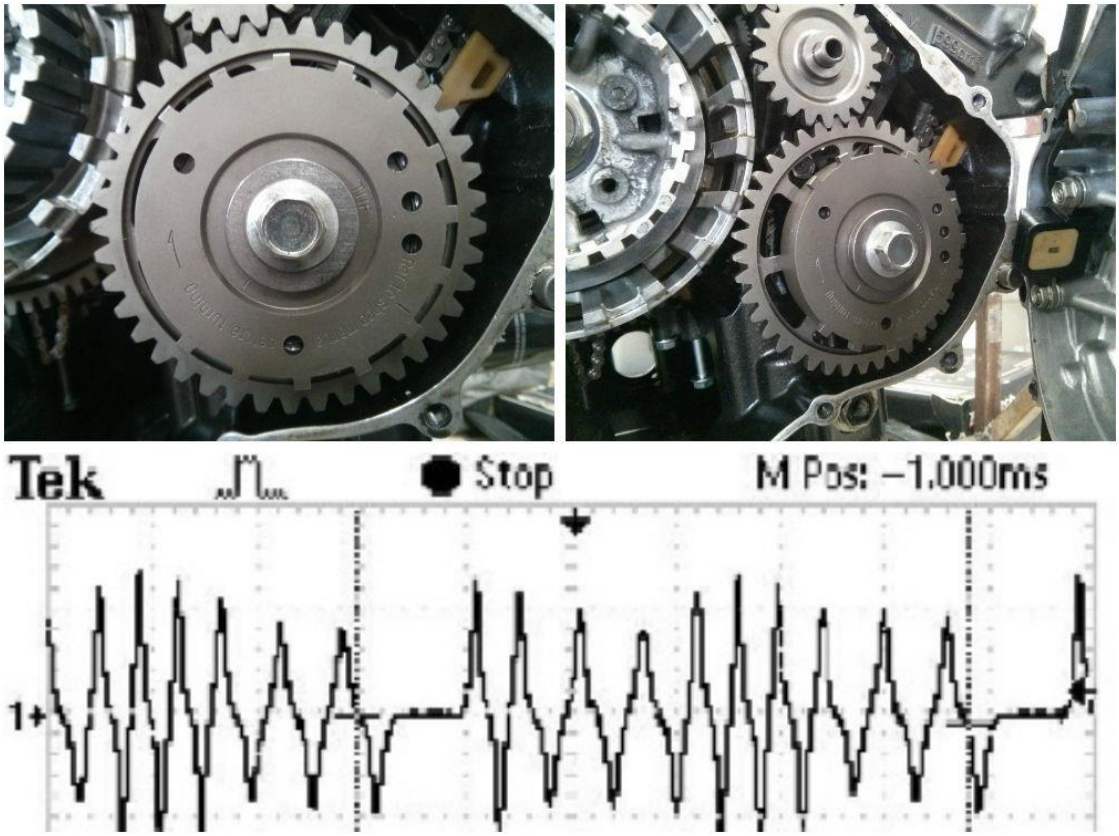
Gerando assim um pulso de tensão no sensor que é enviado ao microcontrolador para tratamento, armazenamento e transmissão do sinal para o módulo ZigBee que envia via wireless em tempo real o sinal para computador.

3.1.2. Rotação do Motor

O sinal referente à rotação do motor também é digital e original do motor do carro protótipo, composto por uma roda fônica, ou seja, um disco magnético dotado de

dentes, com um padrão de formação de 16-3 (13 dentes e 3 falhas consecutivas) e um sensor indutivo que interpreta as passagens dos dentes gerando um padrão de onda como mostra a figura 6:

Figura 6: Sinal Roda Fônica



Fonte: (AUTOR, 2014)

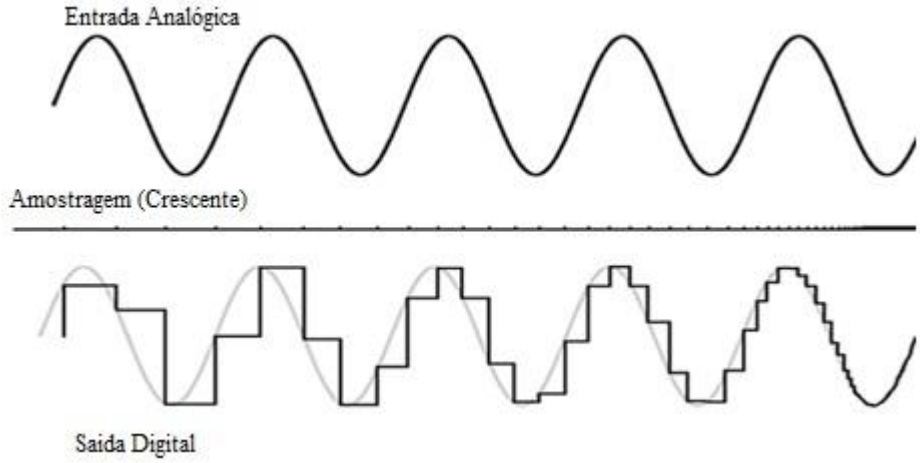
3.1.3. Temperatura do Motor

A temperatura do motor é obtida também por meio de um sensor. O sinal que este sensor apresenta é um sinal analógico, mas também original do motor, ou seja, é um sinal continuo no tempo e tem a necessidade de ser convertido em digital, para que o microcontrolador possa interpretá-lo e para isto é preciso que o mesmo passe antes por um conversor **A/D** (**Analógico/Digital**) para que seja enviado para o PIC.

3.2. O CONVERSOR A/D

É o dispositivo que converte sinais analógicos em sinais discretos de modo que o microcontrolador possa interpretar corretamente a informação a ser recebida e/ou transmitida. Consiste em transformar o sinal contínuo no tempo em alguns níveis de tensão, dependendo da resolução que o conversor possui, assim obtendo maior ou menor precisão de conversão. Outro fator importante para uma conversão A/D é a taxa de conversão, que indica quantas vezes por segundo o sinal analógico ou digital é quantificado. Isto pode ser observado na figura 7. Sendo o gráfico superior um exemplo de sinal analógico, o central mostra a taxa de conversão e o último representa o sinal convertido em um sinal digital. Neste trabalho é utilizado o conversar A/D interno ao microcontrolador que possui 10 bits para conversão e 13 canais para conversão.

Figura 7: Exemplo de conversão analógico-digital



Fonte: adaptado (DANTAS, 2014)

3.3. AMBIENTE DE DESENVOLVIMENTO

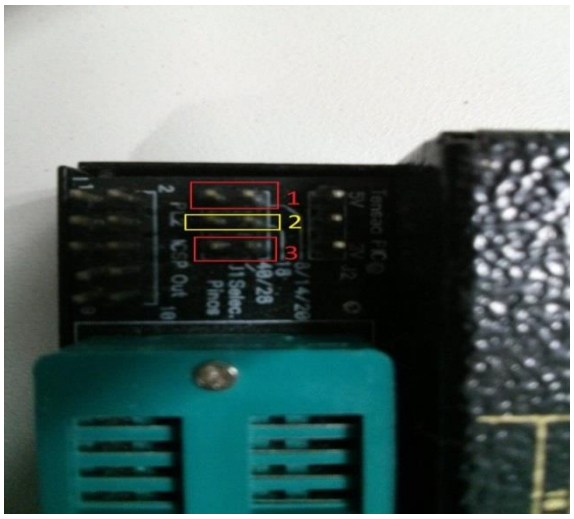
Para que o microcontrolador possa exercer suas tarefas é necessário programá-lo, para isto existem várias linguagens e ambientes de programação possíveis, para este trabalho foi escolhido a linguagem C e o ambiente de trabalho MPLAB, com o compilador *C18* que irá finalmente compilar a programação que é repassada ao microcontrolador usando um gravador de ***MultPROG Plus – Programador PIC – PicKit2***, que para ser utilizado basta colocar o PIC na posição indicada no gravador, como mostra a figura 8.

Figura 8: Posicionamento do Microcontrolador no gravado

Fonte: (AUTOR , 2014)

Dependendo da quantidade de pinos do microcontrolador deve-se setar algumas configurações no *hardware* do gravador de PIC, podendo-se optar por configurações de 8/14/20 para o par mais externo [1], 18 para o par central [2] e 28/40 para o par mais próximo ao microcontrolador [3], conforme mostra a figura 9:

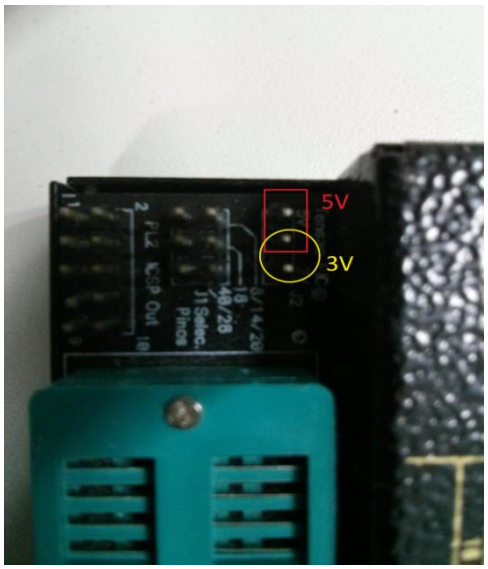
Figura 9: Seleção de pinagem do PIC



Fonte: (AUTOR , 2014)

Ainda pode-se ajustar a tensão de operação para gravação que pode ser de 3V ou 5V, visualizada na figura 10:

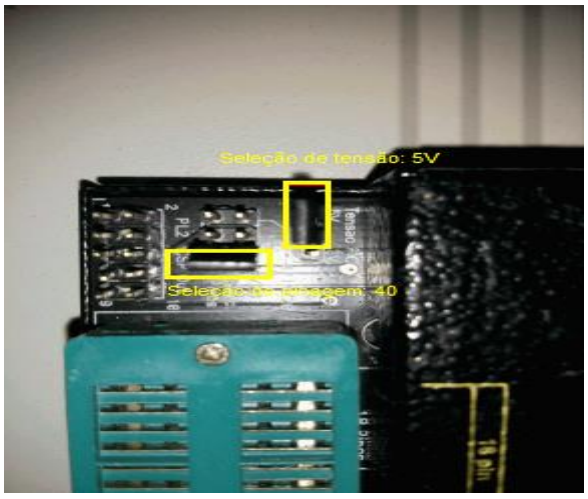
Figura 10: Setup do Ajuste de Tensão de gravação



Fonte: (AUTOR , 2014)

A partir disto, como o circuito desenvolvido tem nível lógico alto de 5V DC e o microcontrolador em questão possui 40 pinos, assim a configuração requisitada pelo projeto mostrada na figura 11:

Figura 11: Setup para Gravação de Microcontrolador de 40 pinos a 5Vdc



Fonte: (AUTOR , 2014)

Neste ambiente, foi desenvolvida a lógica para aquisição dos sinais descritos anteriormente, todavia ainda assim não é possível que um usuário possa ler os dados em tempo real, para isto foi desenvolvida uma interface gráfica que lê os valores obtidos em tempo real. Assim é possível interpretar os dados no mesmo momento

que são lidos pelo MATLAB, podendo então realizar os ajustes, caso sejam necessários, de maneira mais eficaz e de modo preventivo também.

Mas não é tão simples quanto parece, deve-se tomar alguns cuidados com a programação no ambiente MPLAB, para que a linguagem desenvolvida possa ser interpretada corretamente neste ambiente em conjunto com o *C18*. Para tal deve-se seguir estes passos:

3.3.1- Instalação do Ambiente de Desenvolvimento

Instalar o ambiente MPLAB e em seguida o *C18,* que são os dois arquivos executáveis.

3.3.2- Inserindo bibliotecas de compilação

No ambiente do MPLAB, na guia "*Project*" / "*Select Language Toolsuite*" e em "*Active Toolsuite*" selecionar a opção que contiver "*Microchip C18 Toolsuite*" e em "*Location*" que deve conter o executável do PIC que será utilizado para cada um das quatro opções disponíveis, cada uma com seu executável, basta procurar onde foi instalado o C18 no computador e escolher o executável correspondente.

3.3.3- Seleção de Gravação no Microcontrolador

Quando o código estiver pronto para ser compilado e escrito no PIC, deve-se ir no guia "*Programmer*" / *"Select Programmer"* / "*PICkit 2*".

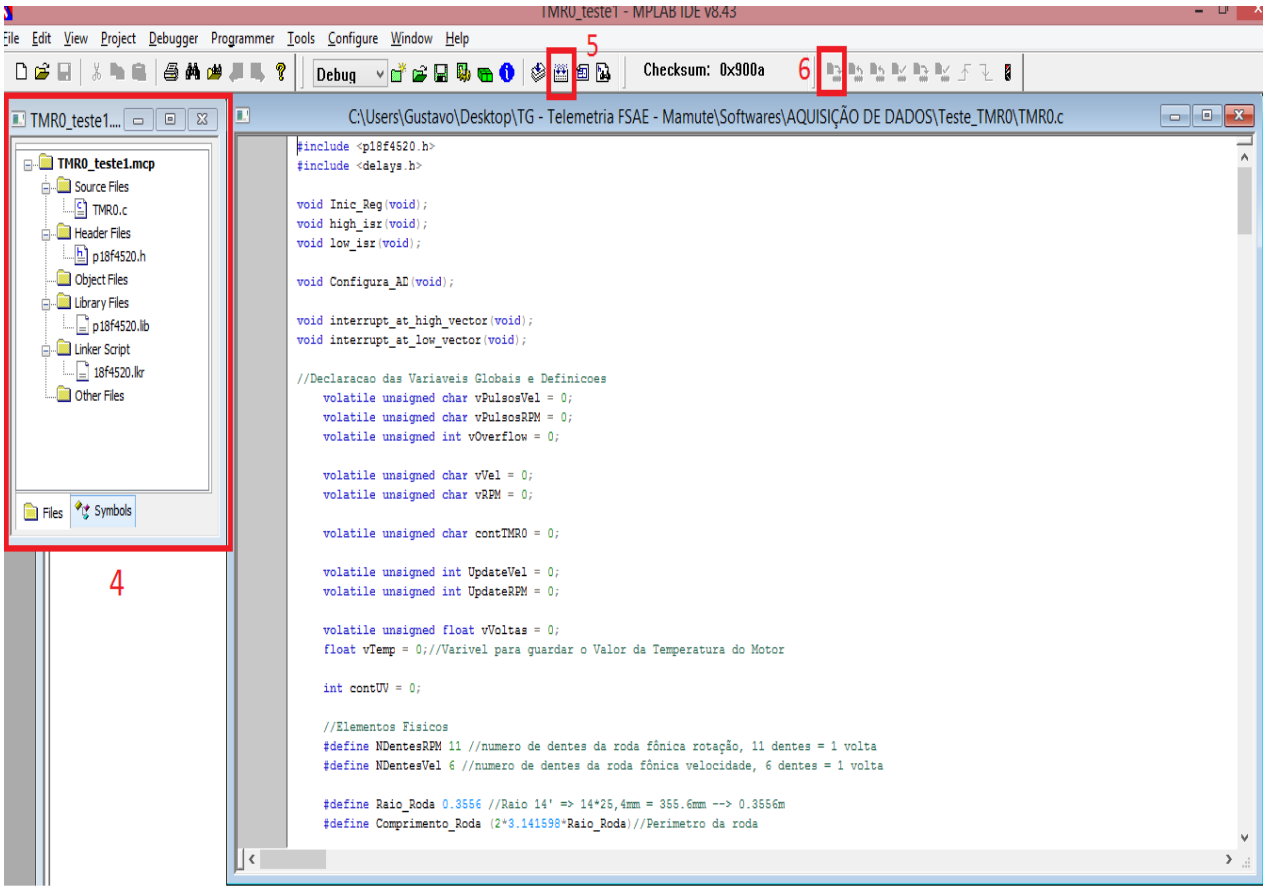
3.3.4- Verificação da Árvore de Projetos

Após os passos anteriores, deve-se verificar se os arquivos utilizados estão na árvore do projeto. É preciso que estejam na árvore de trabalho [4] para que a programação possa ser compreendida pelo ambiente de trabalho.

3.3.5- Programando o Dispositivo

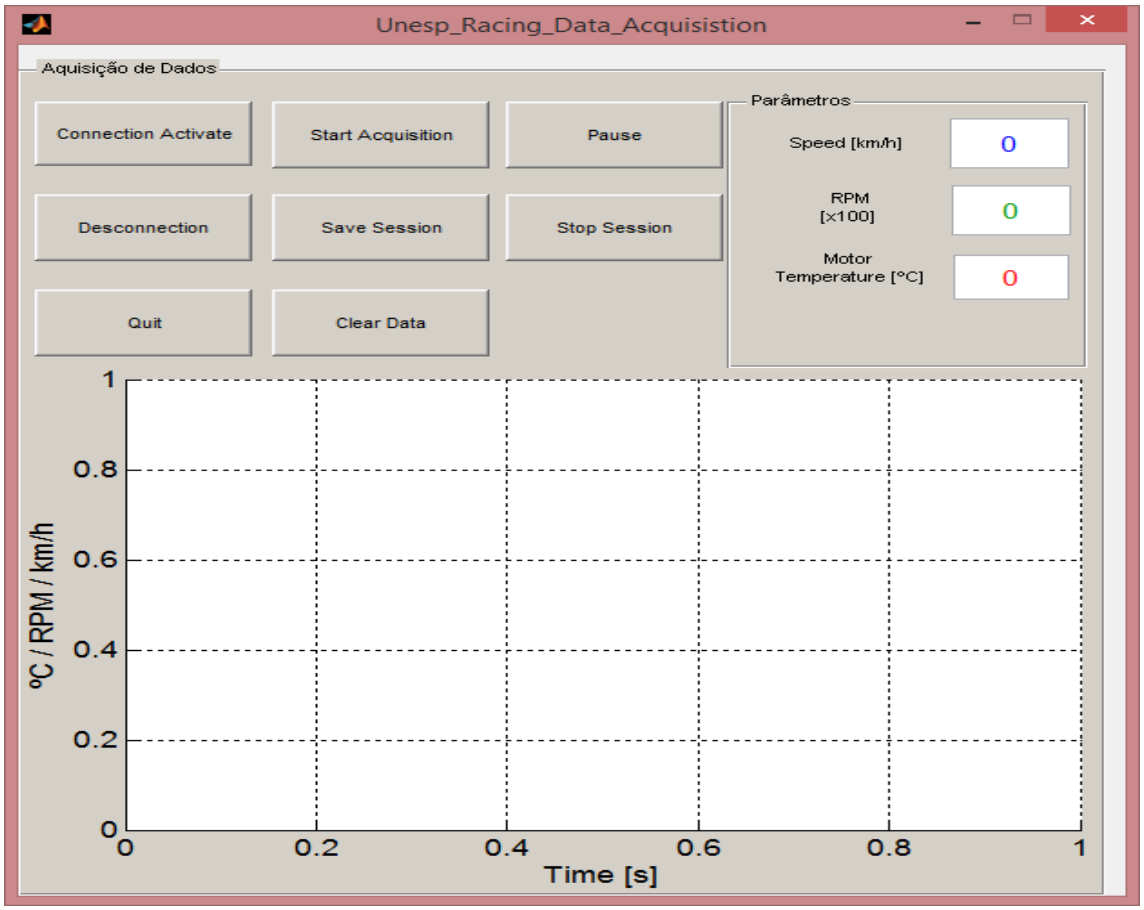
Agora basta compilar a programação clicando no botão branco com duas setas azuis para baixo do ambiente de trabalho do MPLAB, *Build All* [5], se necessário corrija os erros existentes e em seguida coloque o PIC no Gravador de PIC na posição correta e fixo ao gravador, conecte o cabo do gravador no gravador e no computador, a agora sim pode-se gravar as informações no PIC, clicando no primeiro botão amarelo chamado *Program the target device* [6], ou pode-se também ir na guia *Programmer / Program* como ilustra a figura 12.

Figura 12: Ambiente de desenvolvimento MPLAB



Fonte: (AUTOR , 2014)

Figura 13: Interface gráfica no Ambiente Matlab



Fonte: (AUTOR , 2014)

A interface da Figura 13 foi totalmente desenvolvida no MATLAB utilizando-se uma ferramenta interna denominada ***GUIDE***, *Graphical User Interface Development Environment*, que permite fazer de forma simples a interface. Nela então é preciso dar as devidas funcionalidades no código que é gerado automaticamente quando se termina o layout da interface.

Neste HID, *Human Interface Device*, é possível habilitar a comunicação da porta USB para que a HID possa ler os dados transmitidos, basta clicar no botão “Ativar Conexão” e o contrário, ou seja, desabilitar a comunicação com o computador, então deve-se clicar no botão “Desconectar”.

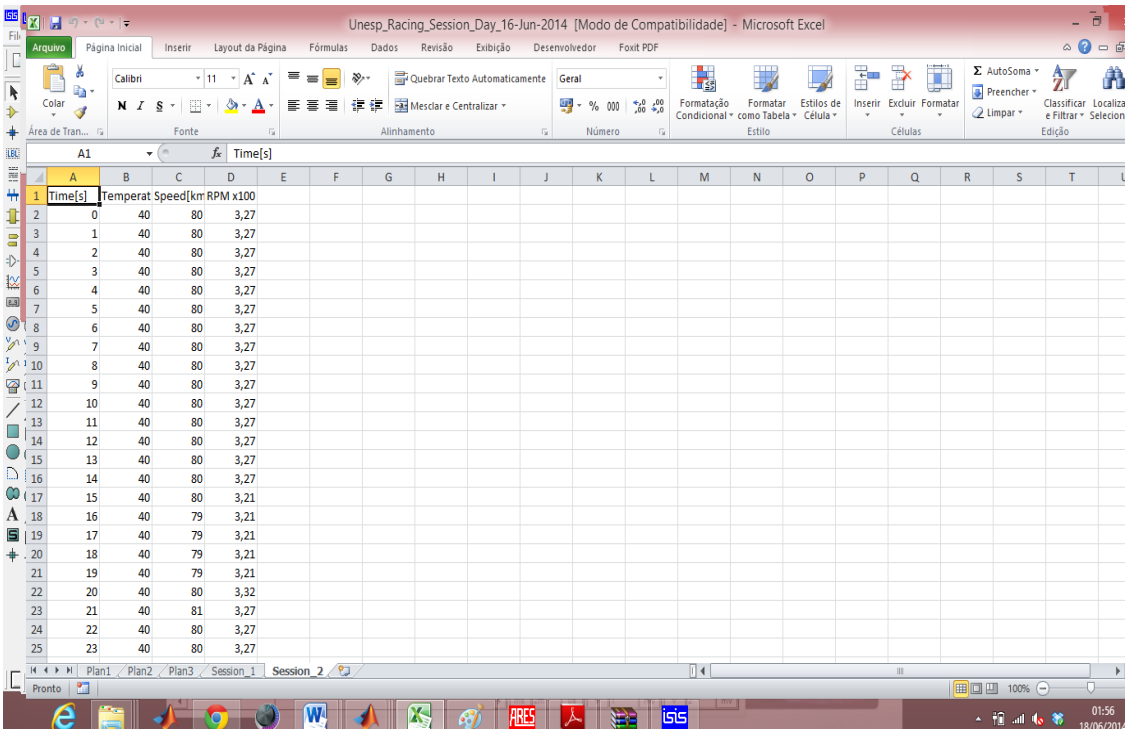
Já com a porta habilitada para transmissão para dar início à aquisição de dados é preciso então apertar o botão “Começar Aquisição” e em seguida dados são plotados no gráfico que o HID possui. Ainda é possível pausar a aquisição, que nada mais é parar momentaneamente a aquisição e retomá-la do ponto onde foi pausada, assim ao parar a aquisição, já aqui a aquisição dos dados é interrompida faz-se uma pergunta

ao usuário da HID se ele deseja salvar os dados ou descartá-los e em seguida limpa os dados capturados para se dar inicio a uma nova aquisição.

E ainda conta com um botão exclusivo para salvar quando se desejar fazê-lo, aqui é criado um diretório no computador, no caminho ***C:\FSAE Telemetry\Sessions*** mesmo que nele não exista. Em seguida é criado um arquivo Excel (extensão.xls) dos dados que foram capturados na sessão atual. Cada arquivo é criado de acordo com a data que o computador apresenta, ou seja, se no dia 16 de junho de 2014 foi solicitado uma gravação, a HID irá criar o arquivo **Unesp_Racing_Session_Day_16-Jun-2014**, então a primeira sessão irá criar uma planilha nesta pasta de trabalho excel chamada Session_1 e para cada gravação subsequente nesta mesma data irá gerar planilha Session_2, Session_3 e assim por diante. E caso seja solicitado uma gravação num dia diferente da data apresentada pelo computador é criada uma nova pasta de trabalho na data que se encontrar o computador. Assim no Excel é possível fazer qualquer tipo de tratamento destes dados para análises posteriores a sua aquisição.

A figura 14 mostra um exemplo de uma pasta de trabalho de uma aquisição de dados.

Figura 14: Pasta de trabalho de uma Aquisição de Dados



Fonte: (AUTOR , 2014)

3.4. LÓGICAS DE PROGRAMAÇÃO

3.4.1. Aquisição de dados

A lógica para aquisição de dados tem por finalidade tratar e aquisitar os sinais de Temperatura do Motor, Rotação do Motor e Velocidade do veículo, sendo que o sinal de temperatura é um sinal analógico, pois se trata de um sinal contínuo no tempo e precisa ser convertido em sinal digital, para que o microcontrolador possa entender os níveis de tensão, pré-determinados, correspondentes aos valores de temperatura.

Já o sinal de Velocidade e de Rotação são sinais discretos então é preciso somente que haja a conversão correta dos pulsos que são lidos no tempo para km/h e RPM. Para isto basta fazer uma conversão simples, com o perímetro do pneu, tem-se a distância que o carro percorre cada vez que a roda dá uma volta completa, logo é de suma importância saber a quantidade de pulsos gerados numa volta completa, desta forma, com a distância que foi percorrida em uma volta e o tempo que se levou para realizar tal movimento, tendo-se assim a Velocidade Instantânea do carro. Já para a Rotação, basta apenas saber o trem de pulsos de uma volta completa da árvore de engrenagens do motor obtido no tempo e converter este trem de pulsos para a razão de rotações por minuto.

3.4.2. Interface gráfica

A lógica com o intuito de ler, mostrar e salvar os dados que o sistema de aquisição envia para o computador através dos módulos Zigbees, é dada atenção especial no próximo tópico, a interface gráfica contempla a tela para plotar os gráficos dos dados adquiridos em função do tempo em tempo real, o botão para inicializar e sair da aquisição, o botão para salvar os dados para estudos posteriores, além de displays para visualizar os valores instantâneos dos dados.

4. O MÓDULO DE TRANSMISSÃO WIRELESS

4.1. ZIGBEE ALLIANCE

É um conjunto de empresas de vários ramos, que estão desenvolvendo protocolo de comunicação dos módulos ZigBee para que estes sejam confiáveis, de baixo consumo e de taxas de transmissão baixa, com um máximo próximo de 250 kbps e 200 kbps respectivamente nas frequências de operação de 2,4 GHz e 900 MHz, e que estes módulos sejam de uso global.

4.2. O ZIGBEE

Figura 15: Módulo ZigBee Xbee PRO S3B - RPSMA

Fonte: (X-Bee Store, 2013)

O ZigBee leva algumas vantagens quando comparado à outros sistemas de comunicação *wireless* como o *wi-fi* e *Bluetooth*, pois o ZigBee é o que possui um alcance maior de comunicação, consome menos energia dentre estes três, entretanto é o que tem menor taxa de transmissão, como a taxa de transmissão não é um fator de suma importância para o sistema que é apresentado neste trabalho e por ter as vantagens mencionadas anteriormente o módulo de transmissão de dados *wireless* escolhido foi o ZigBee. A tabela 2 mostra comparação entre estas tecnologias.

Tabela 2: Comparação de Tecnologias na frequência de 2,4GHz

Característica	IEEE 802.11B	Bluetooth	ZigBee
Bateria	Horas	Dias	Anos
Complexidade	Muito complexo	Complexo	Simples
Dispositivos	32	7	64000
Distância	100 metros	10 metros	70-300 metros
Extenção	Roaming possível	Não	Sim
Taxa Tranferência	11 Mbps	1 Mbps	250 Kbps

Fonte: (DESMONTA CIA, 2013)

A família ZigBee possui várias séries de fabricação e cada qual se comunica numa faixa de frequência específica, e uma delas com taxas de transmissões distintas em distâncias diferentes, para se ter uma ideia, quanto maior a taxa de transmissão menor é a distância de comunicação, uma destas séries até se comunica via *WI-FI*. A série escolhida foi a família da série 3, que são os módulos que trabalham numa frequência de 900 MHz, tendo assim uma distância de comunicação entre os transmissores, com antenas de alto ganho de aproximadamente 3000 m, porém quando utilizada uma antena de alto ganho podem chegar até um distância de 14000 m consumindo uma potência de 50 mW, com uma taxa de transferência de 10 kbps.

A tabela 3 ilustra as famílias de módulos XBee.

Tabela 3: Comparação de Famílias Zigbee

Módulos XBee™

802.15.4 / ZigBee / DigiMesh / Multiponto

Digi VK GRUPO VIKA CONTROLS
www.vikacontrols.com.br
Soluções em Wireless

Especificações

Protocolo	802.15.4		ZigBee		DigiMesh		DigiMesh/Multiponto
	XBee	XBee-PRO	XBee ZB	XBee-PRO ZB	XBee	XBee-PRO	XBee-PRO 900HP-S3B
Perfomance							
Taxa de dados em RF	250kbps		250kbps		250kbps		10 ou 200kbps
Alcance Indoor/Urbano	30m	90m	40m	90m	30m	90m	610m
Alcance em visada direta	90m	1,6km	120m	1,6km	90m	1,6km	14km
Potência de transmissão	1mW (0dBm)	63mW (18dBm)	1.25mW (1dBm)	63mW (18dBm)	1mW (0dBm)	63mW (18dBm)	250mW (24dBm)
Características							
Interface de dados serial	UART 3,3V CMOS		UART 3,3V CMOS		UART 3,3V CMOS		UART 3,3V CMOS, SPI
Método de configuração	via comandos AT ou API		via comandos AT ou API		via comandos AT ou API		via comandos AT ou API
Banda de frequência	2.4GHz		2.4GHz		2.4GHz		900MHz
Imunidade a interferências	DSSS		DSSS		DSSS		FHSS
Taxa de transferência serial	1200 bps - 250 kbps		1200 bps - 1 Mbps		1200 bps - 115,2 kbps		9600 bps - 230,4 kbps
Conversores AD	(6) 10 bits		(4) 10 bits		(6) 10 bits		(4) 10 bits
I/Os digitais	8		10		13		15
Opções de antena	fio, PCB, U.FL, RPSMA		fio, PCB, U.FL, RPSMA		fio, PCB, U.FL, RPSMA		fio, PCB, U.FL, RPSMA
Rede & Segurança							
Encriptação	128 bits - AES						
Confiabilidade	Retransmissões e confirmação de recebimento						
Requisitos de consumo							
Tensão de alimentação	2,8 - 3,4 Vcc		2,1 - 3,6 Vcc	2,7 - 3,6 Vcc	2,8 - 3,4 Vcc		3,0 - 3,6 Vcc
Consumo na transmissão	45 mA	215 mA	35 mA	205 mA	45 mA	250 mA	215 mA
Consumo no recebimento	50 mA	55 mA	38 mA	47 mA	50 mA	55 mA	29 mA
Modo de baixo consumo	<10 µA		<1 µA	<3,5 µA	<50 µA		<2,5 µA
Homologação							
Anatel	0991-10-1209	0591-10-1209	1028-10-1209	0229-10-1209	0991-10-1209	0591-10-1209	3727-12-1209

Fonte: (Vika Controls, 2013)

As redes de comunicação ZigBee tem seus módulos ocupando três distintas funções, a de **Coordenador** (*cordinator*), **Roteador** (*router*) e **Dispositivo Final** (*end device*).

4.2.1. Coordenador

Esta função possui um único módulo executando por rede, e este é capaz de criar uma rede, possui a maior variedade de funções a executar e ainda somente ele pode ser

o dispositivo de maior autonomia para comutar os dados, além de fazer o armazenamento das informações da rede, tornando-se assim a raiz.

4.2.2. Roteador

Estes dispositivos têm por finalidade repassar a informação que recebe, do coordenador ou mesmo de outro roteador, para os demais elementos da rede, tendo o máximo de módulos possíveis em torno de 64000 dispositivos interagindo numa mesma rede. Mas neste projeto são apenas 2 (**dois**), um coordenador e um dispositivo final.

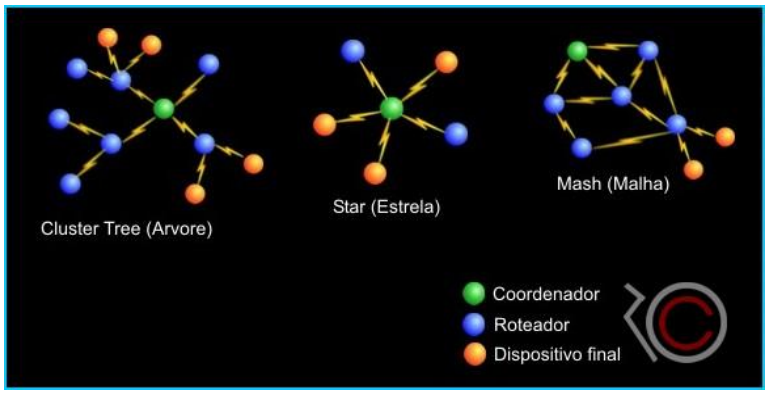
4.2.3. Dispositivo Final

Sua função é apenas receber informações advindas de um coordenador ou um roteador, uma de suas vantagens é ter uma memória com capacidade menor que os demais dispositivos da rede, diminuindo assim o seu custo.

4.3. TOPOLOGIAS DE REDE

As Redes de Zigbee podem assumir vários tipos de topologia de rede que podem ser do tipo **Estrela**, **Mesh**, **Árvore** como ilustra a figura 16.

Figura 16: Estruturas de Rede Zigbee



Fonte: (RogerCom, 2013)

4.3.1. Topologia Estrela

Pode ser denominada também como topologia *peer-to-peer*, neste tipo os módulos receptores de sinais transmitidos e que são os dispositivos finais da rede é apenas de comunicarem-se com o coordenador da PAN (**Personal Area Network**), desta forma sem a necessidade de usar roteadores para enviar o sinal.

Dependendo da organização de uma topologia *peer-to-peer*, pela razão de que o módulo FFD (Full Function Device) Zigbee poder se comunicar com dispositivo FFD mais próximo e sendo esta rede formada por módulos exercendo todos os papéis disponíveis pelo padrão, pode assumir uma topologia Mesh ou Árvore.

4.3.2. Topologia Mesh

É quando os dispositivos FFD passam a transmitir informações entre si em seu raio de alcance, os dispositivos finais da rede podem utilizar roteadores para se comunicar com o coordenador da rede.

4.3.3. Topologia Árvore

Esta topologia é a mais organizada das três, devido como os novos dispositivos tentam adentrar na PAN, estes necessitam averiguar se o dispositivo FFD está disponível para recebê-los na rede e este tipo de topologia cresce com a inclusão de novos roteadores em seus ”galhos” para se comunicar com as “folhas”, dispositivos finais da rede, RFD (**Reduced Function Device**).

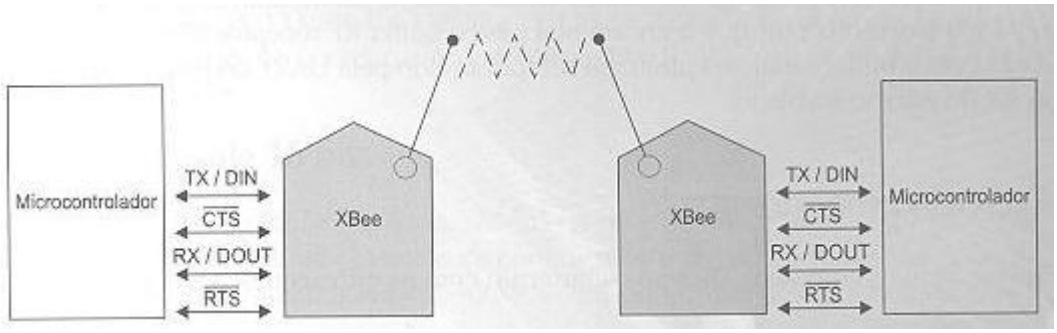
4.4. COMUNICAÇÃO DO MÓDULO ZIGBEE

A comunicação é feita de forma assíncrona com qualquer dispositivo que suporte a comunicação **UART – Universal Asynchronous Receiver Transmission**, ela se dá quando um dado UART enviado por um microcontrolador de seu pino Tx, ou seja, de seu transmissor para o pino **DIN**, que é o pino que recebe a informação do microcontrolador do módulo Zigbee, a informação é armazenada num buffer, neste caso

receptor serial que em seguida é armazenado no buffer RF transmissor, aguardando a disponibilização para a transmissão.

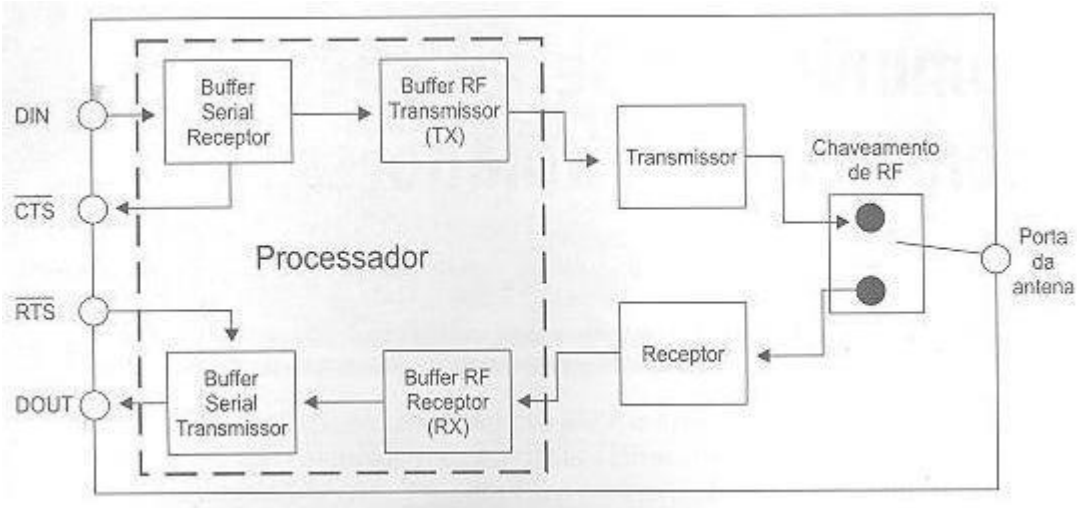
Quanto à recepção do sinal transmitido, o envio ao bloco receptor que por sua vez o passa para o buffer RF receptor, Rx. Na sequência o sinal é enviado ao buffer receptor serial para ser transmitido pelo **DOUT,** pino que passa a informação a um novo microcontrolador ou para o dispositivo que desejar, do Zigbee para o pino Rx, ou seja, receptor do microcontrolador. A figura 17 mostra o sistema comunicação serial ZigBee com microcontrolador, já a figura 18 ilustra o diagrama de fluxo de dados USART.

Figura 17: Comunicação Serial ZigBee – Microcontrolador



Fonte: (RAMOS, 2012)

Figura 18: Diagrama de Fluxo dos Dados USART



Fonte: (RAMOS, 2012)

Para módulos Zigbee não existem dois ou mais módulos com o endereçamento fixo de identificação de 64 bits, com a mesma numeração. Já o endereço de 16 bits é dinâmico que o módulo recebe para entrar em uma **PAN** e este endereço é fornecido

pelo coordenador ou pelo roteador mais próximo ao dispositivo que deseja entrar na PAN. O endereço 0x0000 é sempre reservado para o coordenador da PAN.

Os dados podem ser transmitidos de modo ***Unicast*** ou ***Broadcast***, a primeira tem sua comunicação feita entre um coordenador e um único destinatário.

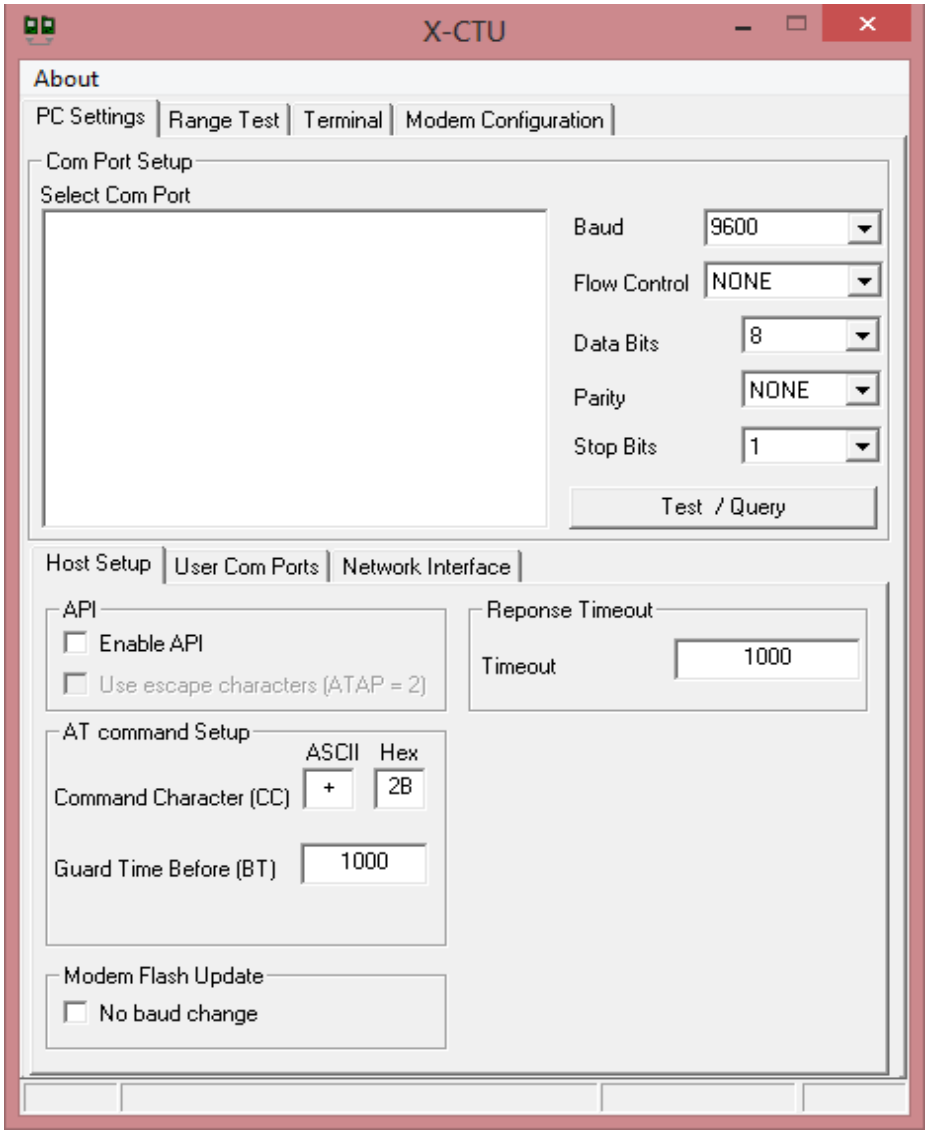
Pode-se usar dois modos de comunicação para se transmitir um sinal, o primeiro é o **Modo Transparente** e o segundo é o **Modo API -** ***"Application Programming Interface"***.

O modo transparente é o modo de comunicação entre apenas dois módulos, isto ocorre neste tipo de transmissão com o envio dos dados que recebeu, ou seja, o sinal é enviado do microcontrolador para o módulo XBee Tx para o XBee Rx, que por sua vez envia para o microcontrolador que faz parte da recepção do sinal.

No modo API de transmissão de dados, pode-se fazer uma rede de módulos XBee muito mais estruturada, pois pode-se fazer alguns comandos como escolher para qual módulo enviar a informação desejada, determinar o tamanho da palavra a ser enviada e o tipo de informação a transmitir.

Para realizar as configurações necessárias nos módulos XBee é necessário ter o software **X-CTU** e o dispositivo que permite a comunicação do módulo Zigbee com o computador para **CON-USBEE**. A figura 19 ilustra interface da X-CTU. Já a figura 20 mostra o dispositivo **CON-USBEE**.

Figura 19**: Ambiente X-CTU**



Fonte: (Digi International, 2008)

Figura 20: Dispositivo CON-USBEE

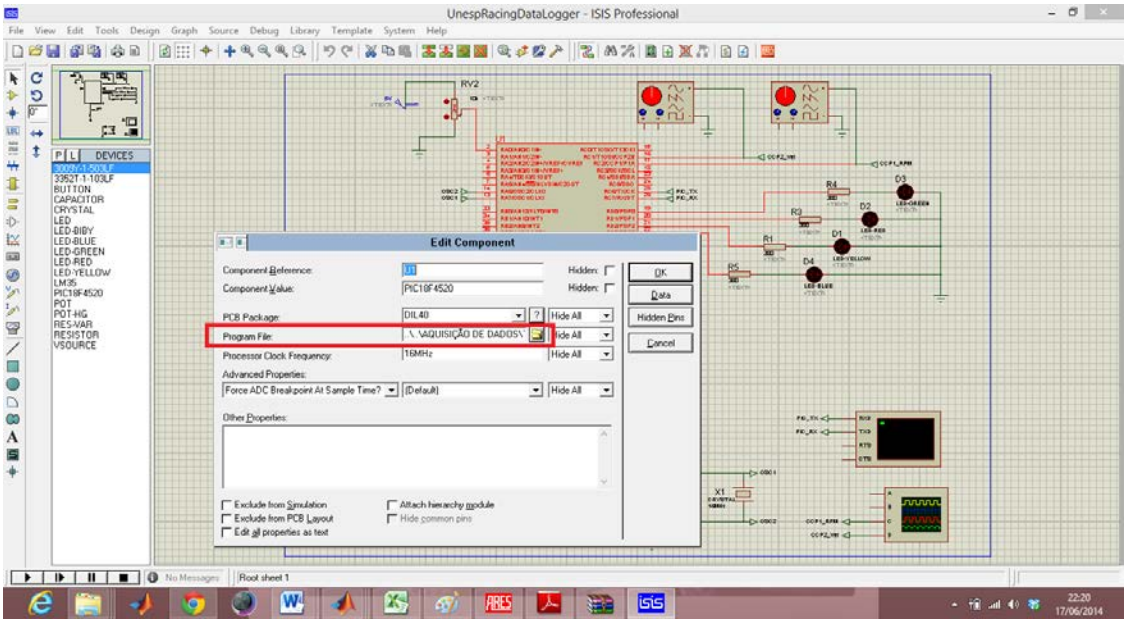
Fonte: (RogerCom, 2013)

Nesta parte do trabalho, é apresentado o processo de montagem, programação, implementação e testes do sistema de telemetria.

4.5. DESEMVOLVIMENTO EM SOFTWARE

Para antecipação de erros e facilitar a confecção de uma placa de circuito impresso foi desenvolvido no ambiente de simulação chamado ISIS, que nele é possível, além de montar virtualmente o circuito que foi implementado na placa de circuito impresso, pode-se simular a programação feita anteriormente no MPLAB, importando seu arquivo hexadecimal, gerado na compilação do programa, importando o arquivo para o microcontrolador que está no software. Isto pode ser feito clicando duas vezes com o botão esquerdo do *mouse* no microcontrolador e no ícone de uma pasta no campo *"Program File"* e selecionar o código fonte hexadecimal da programação desejada, vide o exemplo da figura 21:

Figura 21: Importando programação hexadecimal para o ISIS



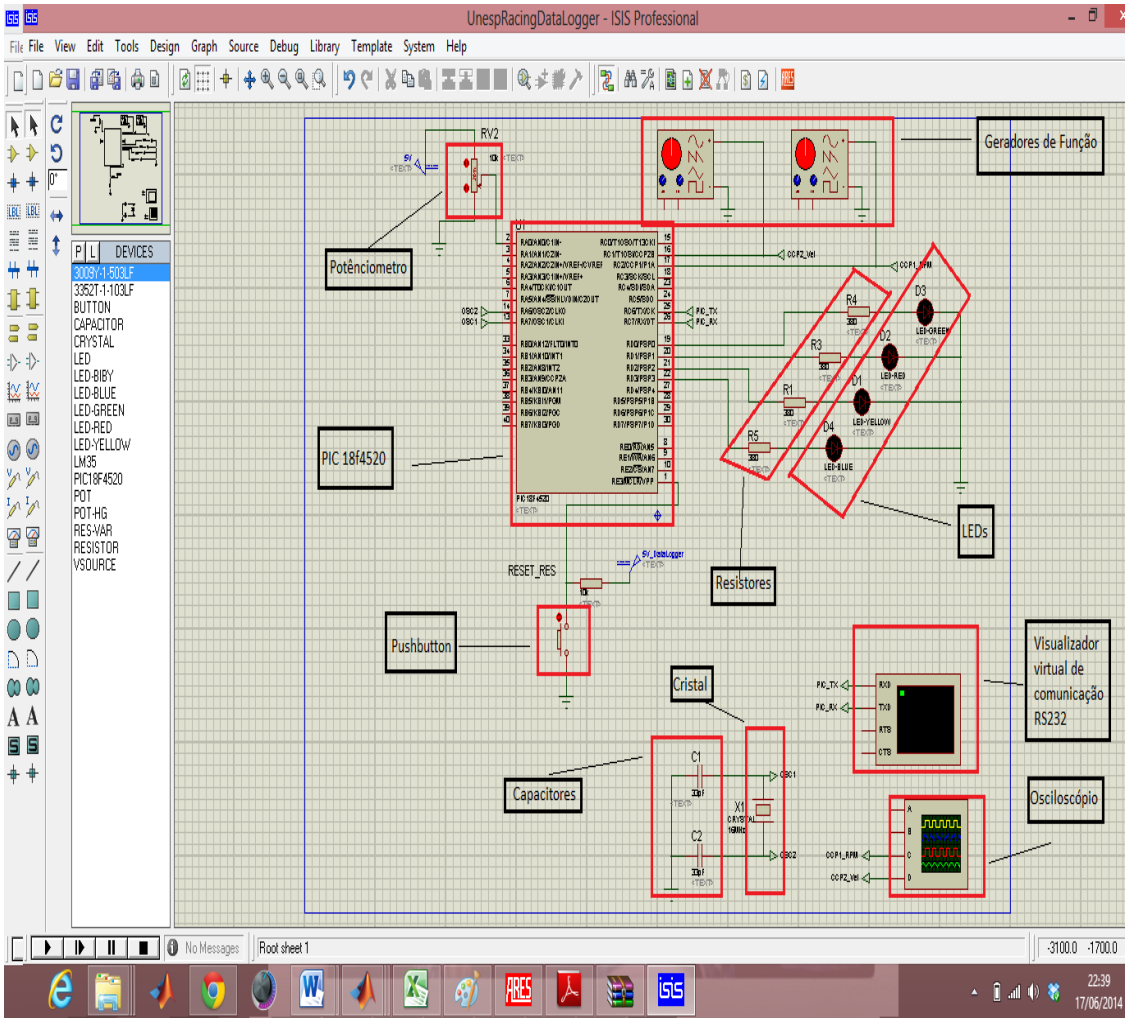
Fonte: (AUTOR , 2014)

Mas antes que isto seja feito é preciso montar seu esquemático no ISIS para que seja possível então simular seu programa. Para tal selecionou-se os componentes desejados na biblioteca de dispositivos e fazer as conexões necessárias para o correto funcionamento da simulação.

Neste protótipo foi utilizado 1(**um**) PIC18f4529, 1(**um**) resistor de 10 kΩ, 4(**quatro**) de 380 Ω, 4(**quatro**) LEDs coloridos para diferenciar as funções do pic, 1(**um**) cristal 16 MHz, 2(**dois**) capacitores de 33 pF, 1(**um**) *pushbutton* para reset do microcontrolador, 1(**um**) potenciômetro para simular o sensor de temperatura, 2(**dois**)

geradores de função virtuais, um para simular pulsos de velocidade e outro para rotação, além de 1(**um**) visualizador de comunicação serial para observar a palavra transmitida e do osciloscópio virtual para visualizar os pulsos gerados. A figura 22 mostra o circuito de simulação em ISIS.

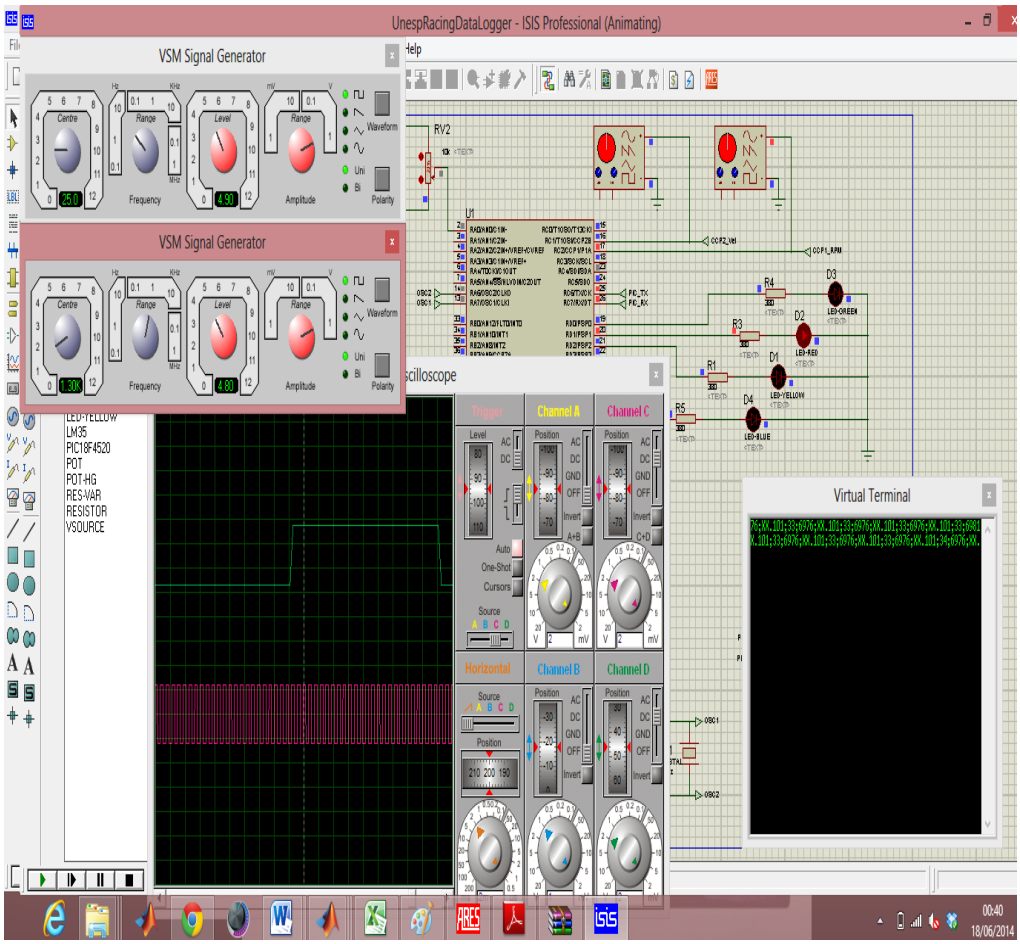
Figura 22: Circuito de simulação na ambiente ISIS



Fonte: (AUTOR , 2014)

A figura 23 mostra o circuito em simulação, com suas respectivas visualizações para alterar os valores dos geradores de função, visualizador do osciloscópio e terminal de comunicação RS232 e potenciômetro para simular o sensor de temperatura.

Figura 23: Circuito em Simulação no ambiente ISIS



Fonte: (AUTOR , 2014)

Terminada esta etapa, ou seja, todos os erros de programação solucionados e circuito respondendo como o esperado, pode-se então desenvolver no ambiente de trabalho ARES, que faz os projetos de placas de circuitos impressos a partir das conexões já feitas no ISIS, circuito final do projeto. Entretanto o ARES não entende alguns componentes que são utilizados no ISIS, estes componentes não possuem **PCB,** ***Printed Circuit Board***, que nada mais é que o componente desenhado em escala para confecção da placa de circuito impresso no ARES.

É importante lembrar que o ISIS e ARES possuem comunicação um com o outro, pois fazem parte do mesmo pacote denominado Proteus desenvolvido pela ***LabCenter***, sendo assim não é preciso montar tudo novamente do zero, basta apenas clicar no botão escrito ARES no *label* superior direito. Ao clicar neste botão o ISIS exporta todos os componentes utilizados com suas respectivas conexões para o ambiente de trabalho ARES, facilitando em muito o processo de fabricação da placa de circuito impresso, precisando então somente alocar os componentes da forma que for a

mais interessante para a confecção assim como suas conexões, podendo inclusive escolher níveis por onde as trilhas da placa poderão passar, desde uma camada superior passando por mais 14(**quatorze**) níveis internos e mais um nível inferior.

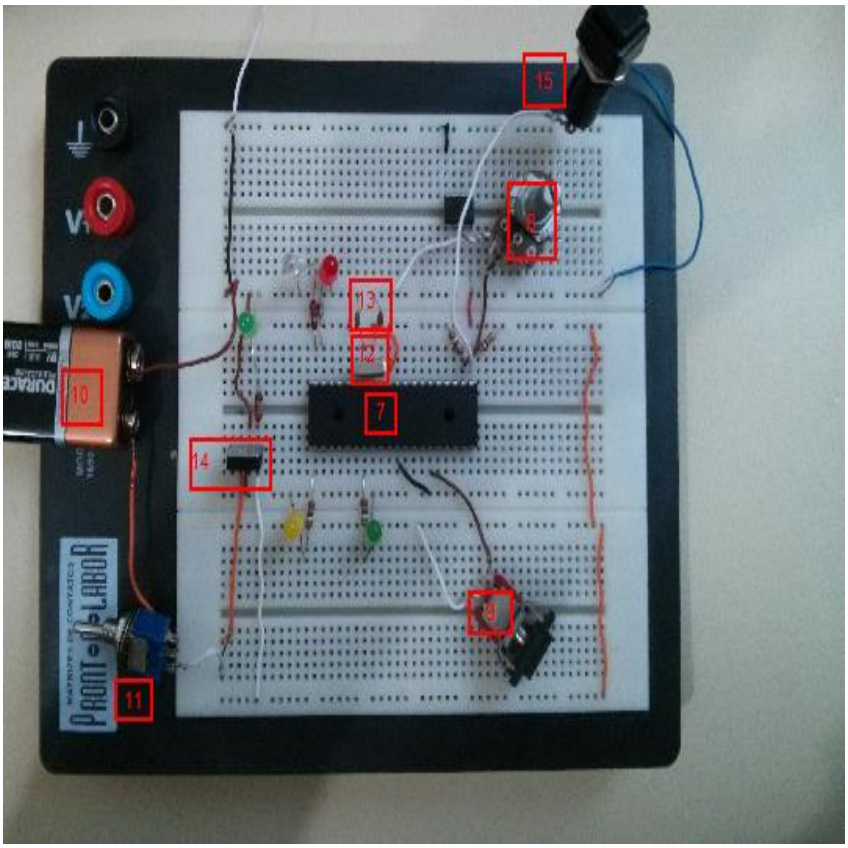
Como este projeto não exige tanta complexidade de conexões foi utilizado somente o nível inferior para as trilhas serem tratadas.

4.6. MONTAGEM DO CIRCUITO

Para a execução da montagem do circuito que irá coletar, enviar e ler os valores adquirido foi preciso, antes de tudo fazer testes, e para isto foi utilizado uma *protoboard* para facilitar montagem e desmontagem do circuito protótipo final a ser utilizado no carro. Com o intuito de facilitar a execução do projeto e minimizar erros na construção da placa de circuito impresso, este procedimento foi divido em 4 (**quatro**) partes.

A primeira adquire os valores de Velocidade Instantânea, Rotação do Motor e Temperatura do Motor que é composta por um microcontrolador **18F4520**, leds para ajudar a visualização do funcionamento do PIC [7], resistores para limitar correntes e divisores de tensão, potenciômetro [8] para simular um sensor de Temperatura do Motor e 2 **(dois)** cabos para simular pulsos de Velocidade Instantânea [9] e Rotação do Motor [9], cabos azul e cinza respectivamente, do pino 16 (CCP2 - Rotação do Motor) para o pino 17 (CCP1 - Velocidade Instantânea), fonte de 6 V de saída [10] para geração de tensão, chave de duas posições [11] para ligar/desligar circuito, cristal XT de 16 MHz [12] para o *"clock"* em conjunto com 2 (**dois**) capacitores de 33pF [13] (recomendação do manual do microcontrolador), LM7805 [14] para abaixar tensões até 12Vdc para 5Vdc e *push button* para reset manual [15] do PIC. A montagem da descrição acima pode ser vista na figura 24.

Figura 24: Montagem provisória no *"protoboard"*



Fonte: (AUTOR , 2014)

A segunda consiste em apenas receber tais sinais do microcontrolador e repassá-las ao ZigBee Coordenador da rede, composta neste caso de 2 (**dois**) módulos, um que irá transmitir para o segundo ZigBee, que é o Roteador, para que ele possa enviar os dados via UART para o computador. Nesta parte já que os computadores modernos não possuem mais o conector DB-9, que realiza a comunicação RS232 com perfeição, então aproveitando-se de um recurso disponível no dispositivo CON-USBEE que é o CI FT232, que consegue fazer uma conversão digital do UART diretamente para o USB. Para tal é necessário fazer o "*download*" e instalação do "*driver*" para a plataforma de operação em uso no site http://www.ftdichip.com/Drivers/VCP.htm, coluna "*Comments*" clicar em "*setup executable*".

Figura 25: Conexão CON-USBEE - Computador

Fonte: (AUTOR , 2014)

Já na terceira parte, que é responsável por receber as informações advindas do ZigBee Roteador, que tem a finalidade de ler no terminal virtual, **TeraTerm**. Nele é possível visualizar os dados que estão sendo encaminhados para porta USB, que esta conectada no computador para aquisição dos dados e em seguida são lidos pelo MATLAB, que em seu código e sua interface mostra em tempo real os valores aquisitados em forma de gráfico e salvar cada sessão quando for solicitado.

Na quarta etapa, já com os dados devidamente tratados no MATLAB, já lidos e recuperados na porta USB, a interface irá lê-los e mostrá-los em gráficos e em valores instantâneos.

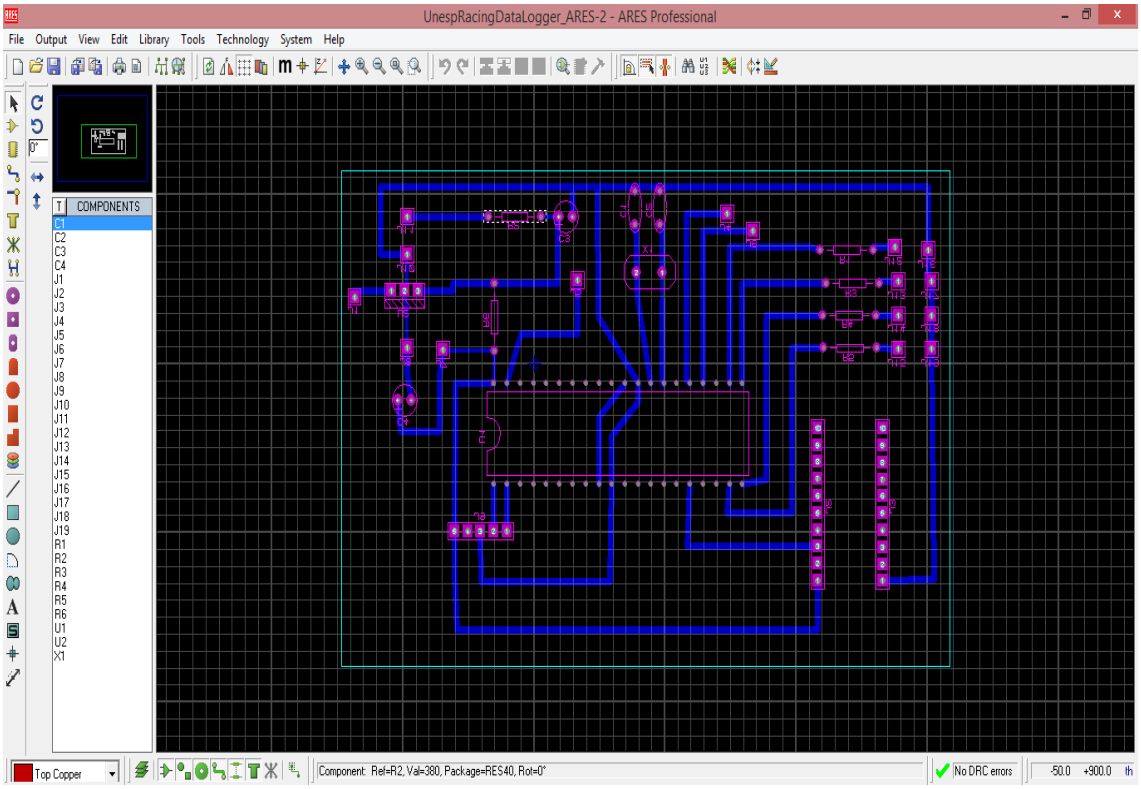
Após testes e ajustes nas 4 (**quatro**) etapas anteriores, pode-se então concluir o circuito que é feito no *software* ARES 7 para posterior fabricação em circuito impresso.

4.7. PROCESSO DE FABRICAÇÃO DO CIRCUITO IMPRESSO

Após a conclusão da montagem e ensaios do circuito no anterior, apresentado na figura 24, na matriz de pontos, para a implementação no veículo, foi desenvolvida uma placa de circuito impresso para a montagem final do circuito do sistema de telemetria.

Para se realizar a confecção desta placa, teve-se então a necessidade de fazer um molde em papel de foto, ou de retroprojetor. E para isto foi necessário utilizar um ambiente de desenvolvimento específico, já dito anteriormente, o ARES, ambiente o qual é possível fazer o desenho de um circuito eletrônico, montar os componentes e ainda realizar simulações no computador antes de começar a manufatura de um circuito eletrônico. A figura 26 ilustra o ambiente de desenvolvimento ARES a disposição dos componentes.

Figura 26: Ambiente de Desenvolvimento ARES



Fonte: (AUTOR , 2014)

Neste caso, como a biblioteca do módulo de transmissão sem fio, ZigBee, é inexistente, não foi possível realizar simulações no *software*, contudo o motivo de se usar este *software* não foi a realização de simulações e sim pela facilidade de desenvolvimento do circuito e pelo fato dele poder converter o circuito em uma figura, a qual é possível imprimi-la para repassar com maior fidelidade e melhor acabamento para a placa de cobre, onde o circuito final será desenhado.

Após a confecção do circuito no ambiente ARES, o circuito final deve ser espelhado, pois durante a confecção da placa de circuito impresso as peças são

colocadas no lado contrário às trilhas de cobre, por isso a necessidade de espelhar o circuito e por fim impresso em papel de foto ou papel de transparência de retroprojetor, tem-se o molde (Figura 27) para a placa de cobre.

A placa de cobre deve estar bem limpa para que as trilhas da impressão possam aderir de forma adequada na placa, para isto é necessário limpar bem com palha de aço e água corrente para que saia todas as impurezas da placa, como gordura dos dedos, oxido de cobre e demais sujeiras. Após a limpeza da placa, deve-se alinhar a impressão com a placa, esticando-se bem o papel e fixar com fita crepe, pois durante o processo se for utilizado algum material para fixação como uma fita isolante, o material irá derreter.

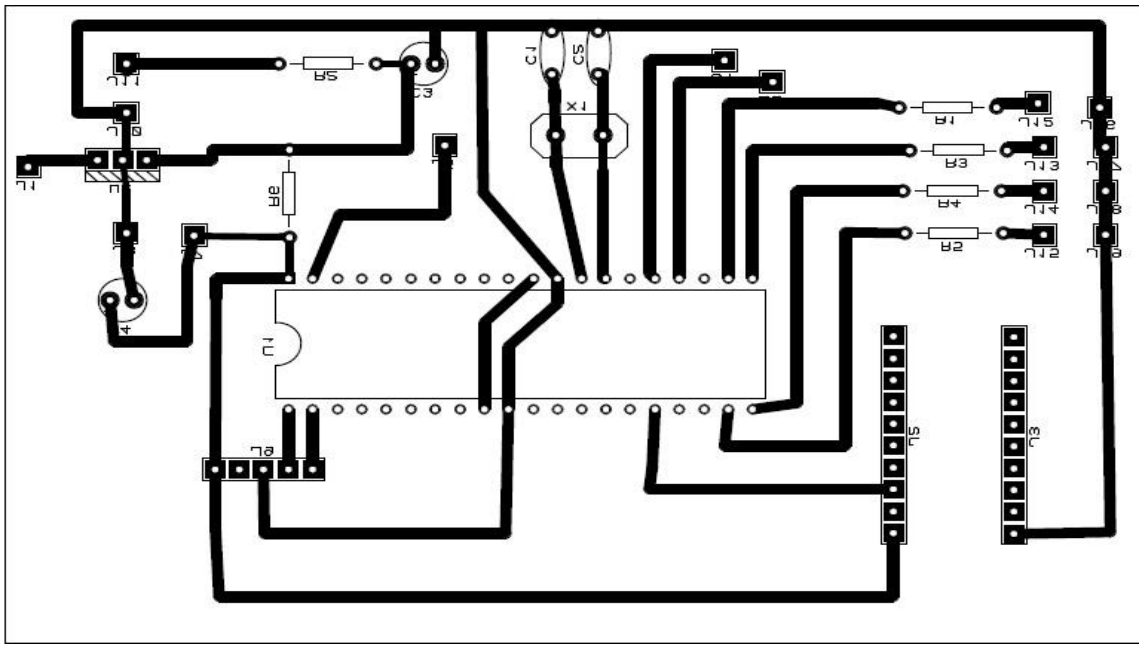
Para o processo de transferência térmica da impressão para o placa de cobre foi utilizado um ferro-de-passar roupas bem quente e foi tomado cuidado para não fazer bolhas e queimar o cobre, então para isto não ocorrer o ferro ao ser pressionado contra a impressão o tempo máximo foi por volta de 20 segundos por toda a placa. Em seguida, após a placa se resfriar para retirar o papel da placa, se for de retroprojetor ele sai com bastante facilidade se for papel fotográfico deve-se então molha-lo com água quente e aos poucos retira-se o papel.

Após este processo de transferência térmica e retirar totalmente o papel da placa, é bom corrigir falhas que possam vir a ocorrer e para isto pode se usar uma caneta de retroprojetor ou canetas específicas para traçar trilhas. Agora a placa está pronta para ser corroída. Para corroê-la é preciso diluir em água percloreto de ferro, a proporção é dada na embalagem do produto, dentro de uma bacia plástica por cerca de 10 minutos.

Após isto deve-se apenas passar levemente uma palha de aço para terminar de retirar excessos, e tomando cuidado para não danificar as trilhas.

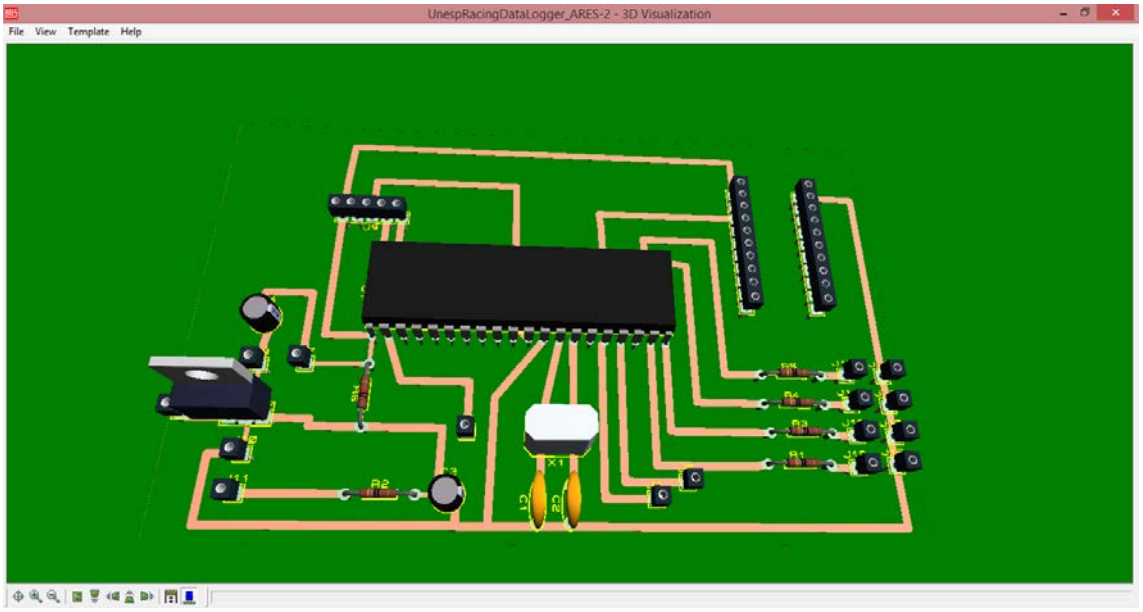
Ainda nesta etapa pode-se ter uma boa ideia de como ficou dispostos os componentes na placa de circuito impresso, pois o ARES possui um recurso de visualização 3D do circuito desenvolvido, apresentado na figura 28.

Figura 27: Molde para confecção do circuito impresso



Fonte: (AUTOR , 2014)

Figura 28: Vista superior 3D da Placa de circuito impresso.

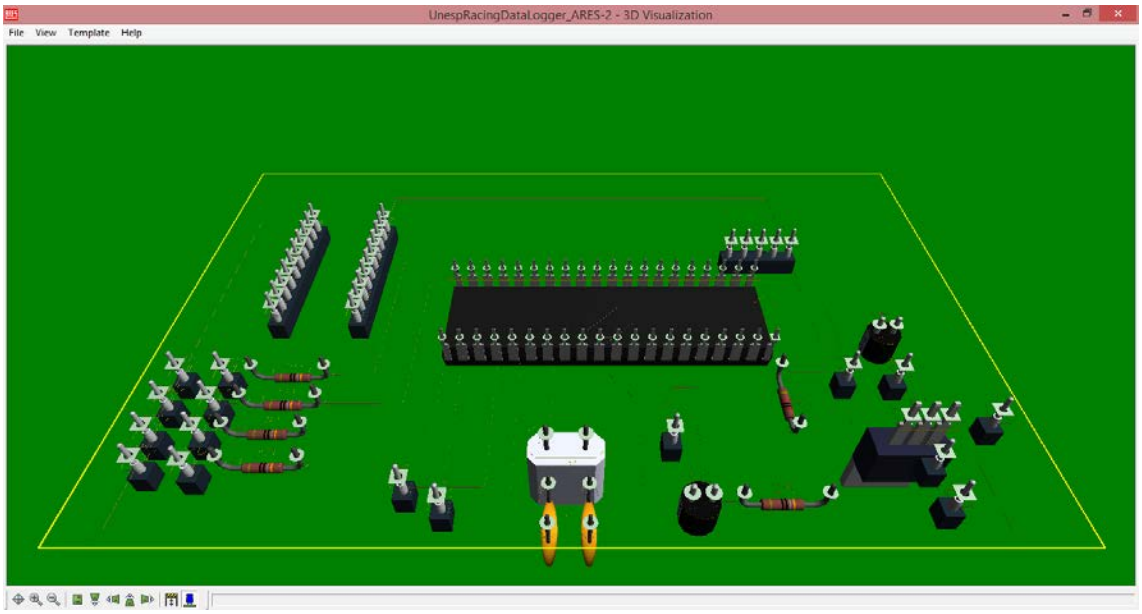


Fonte: (AUTOR , 2014)

Terminada esta etapa é preciso reforçar as marcações das trilhas para garantir que as mesmas estejam bem definidas, para que não sejam corroídas no processo de corrosão do cobre na confecção da placa de circuito impresso.

A figura 29 mostra o design das trilhas da parte inferior da placa de circuito impresso.

Figura 29: Vista inferior 3D da Placa de circuito impresso



Fonte: (AUTOR , 2014)

A partir deste ponto, com o design final, é possível então fabricar a placa de circuito impresso. Para isto é preciso, uma placa com pelo menos um de seus lados com cobre para então transferir a imagem feita no ARES para a placa. O processo é descrito no próximo item.

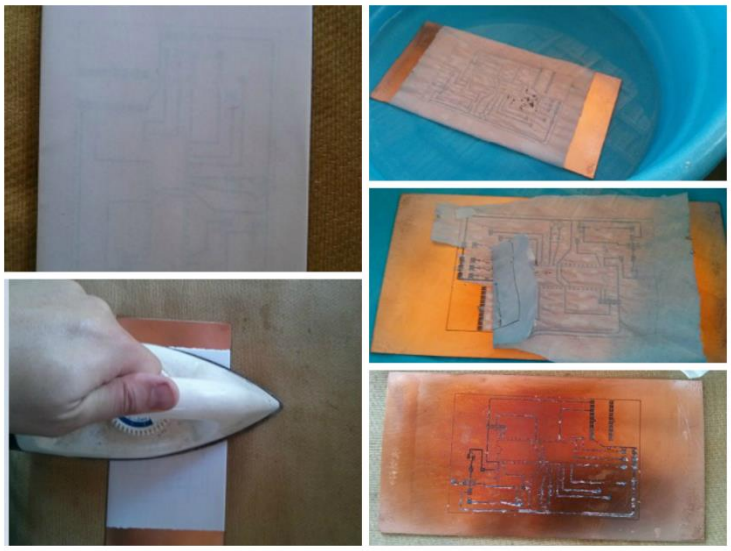
4.8. PROCESSO DE MONTAGEM DA PLACA DE CIRCUITO IMPRESSO

Primeiramente, o circuito final, que foi projetado no ARES, deve ser impresso em papel fotográfico, de transparência para retroprojetores ou mesmo em papel sulfite, porém a qualidade das trilhas de cobre após a corrosão deste último é inferior as duas anteriores, isto ocorre por conta do tipo de papel que o circuito é impresso, pois no processo de transferência térmica, a qualidade do papel e da impressão fazem muita diferença. Neste trabalho, foi utilizado uma impressão de tinta a laser em um papel sulfite, com o intuito de mostrar que mesmo com uma qualidade inferior de impressão o resultado final ainda é satisfatório.

Seguindo o procedimento citado no item 4.7 deste trabalho, deve-se banhar a placa de cobre por aproximadamente 15 (**quinze**) minutos, e verificar se as trilhas do papel aderiram adequadamente à placa de cobre. Em seguida para retirar a folha da placa é preciso uma bacia com água morna, isto para facilitar a remoção, e mergulhar

completamente a placa. Deve-se então muito cuidadosamente, ir retirando a folha da placa para que as trilhas não sejam removidas juntamente com o papel. A sequência deste processo descrito acima pode ser observado na figura 31.

Figura 30: Processo de confecção da Placa de Circuito impresso



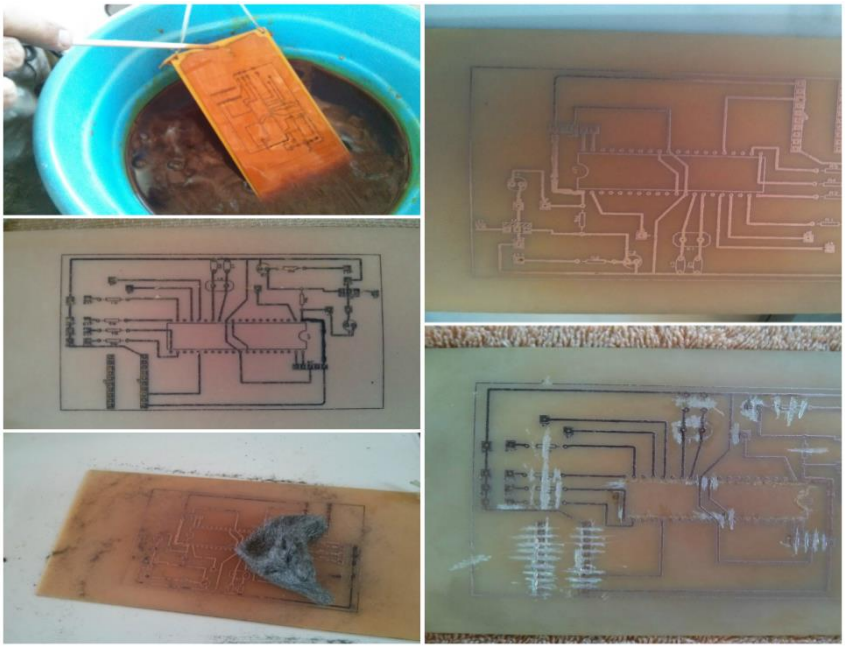
Fonte: (AUTOR , 2014)

Com a transferência térmica realizada, é preciso então remover o cobre da placa que não está protegido pela tinta que foi aplicada sobre a placa na transferência térmica. Para tal, foi utilizado percloreto de ferro que possui a fórmula **$FeCl_3$**. A solução tem a proporção de 0,5 l de água para cada 250 mg do sal adicionado, então a placa de cobre foi submersa nesta solução para que o cobre excedente fosse extraído da placa. Nesta etapa deve-se tomar cuidado com a solução, pois como se trata de uma reação exotérmica, a temperatura pode atingir 70ºC, além de sua acidez em contato com a pele pode causar queimaduras graves. Então para misturar a solução foi utilizada uma bacia de plástico reforçado e uma haste de madeira para misturá-la. Após a mistura ficar homogênea, a placa é colocada na solução para corrosão. Com a intenção de evitar contato direto com a solução foi feito dois furos em uma das extremidades da placa e colocado uma alça de plástico para poder movimentar a placa com mais facilidade. Este processo de corrosão leva cerca de 10 (dez) minutos para ser finalizado.

Concluída a etapa de corrosão da placa, tem-se que verificar se todas as trilhas estão corretas e também averiguar se não há curtos circuitos na placa recém-feita, fazer a furação do posicionamento dos componentes e por fim realizar a montagem dos mesmos na placa, utilizando como referência os circuitos dos softwares para se efetuar

corretamente fixação. Então tem-se como resultado final a Figura 32 já com os componentes devidamente postos em suas respectivas posições.

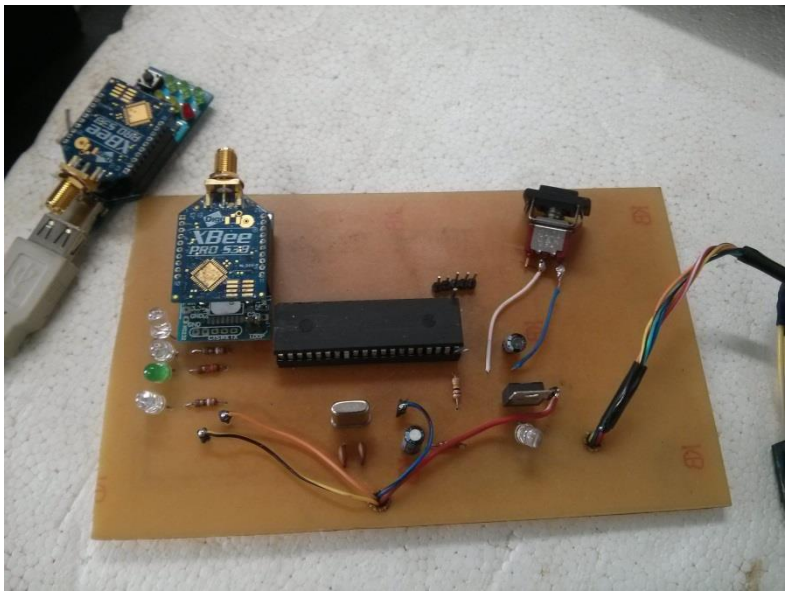
Figura 31: Processo de Corrosão e Acabamento do Circuito Impresso



Fonte: (AUTOR , 2014)

A figura 32 mostra o circuito impresso com seus componentes montados e soldados.

Figura 32: Circuito Impresso Final



Fonte: (AUTOR , 2014)

4.9. AVALIAÇÃO DO NÍVEL DE RECEPÇÃO DO SISTEMA

Antes de montar todo o sistema de aquisição de dados no veículo e inicializar a transmissão dos dados, é necessária uma antena com ganho suficiente para que se tenha um bom alcance, pois do contrário pode haver falhas de comunicação ou mesmo a perda de dados durante a aquisição devido a atenuação entre o carro e o computador.

Com o intuito de se minimizar tais problemas, foi feito um projeto de rádio enlace que levou em consideração, aspectos do local de transmissão como, cidade de grande ou médio porte, movimento de uma das antenas, suas alturas em relação ao solo, distância máxima entre as antenas, ângulo de incidência da onda entre outros mais.

Entre todos os aspectos considerados, há duas modelagens que mais se enquadram são o modelo COST 231 Walfish-Ikagami e Modelo de Lee. As equações e textos dos capítulos 4.9.1 e 4.9.2 são baseados no trabalho de Marco Antonio Betini Pereira, análise de Modelos de propagação na Área Urbana da Região de Curitiba, 2007 e também na Tese de Doutorado de Ronaldo de Andrade Martin, Modelagem e Medições de Ondas de Rádio para Predição de Perda em Ambientes Urbanos.

4.9.1. Modelo COST 231 Walfish-Ikagami

Baseado em trabalhos anteriores como Modelo Hata que por sua vez teve como referência os trabalhos de Okumura. O motivo de ter se escolhido o modelo COST (*Cooperation in the Field of Scientific and Technical*) 231, além de levar em conta aspectos morfológicos detalhados do terreno é a possibilidade da antena da estação base poder ter altura mais baixa do que os telhados dos prédios próximos, pois os modelos de Hata e Okumura também se enquadram em vários aspectos como a frequência de operação, entretanto quanto à altura da antena da estação rádio base, deve ter no mínimo 30 m, característica inviável ao projeto de transmissão de dados do protótipo.

Neste modelo calcula-se os 3 (três) componentes importantes que são: perda no espaço livre (**Lo**), perda por difração e espalhamento no topo de edifícios (**Lrts**), perda devido a múltiplas difrações e reflexões ocorridas ao nível da rua ou multiplanos (**Lms**), onde a perda total (**Lt**) é dada pela soma destes três elementos básicos.

$$L_t = L_O + L_{rst} + L_{ms} \qquad (\boldsymbol{eq.1})$$

$$L_{rts} = -16{,}9 - 10log(w) + 10log(f) + 20log(\Delta hm) + L_{ori} \qquad (\boldsymbol{eq.2})$$

Onde:

$$L_{ori} = -10 + 0{,}354\varphi \qquad para\ 0 \leq \varphi < 35° \qquad (\boldsymbol{eq.2.1})$$
$$L_{ori} = 2{,}5 + 0{,}007(\varphi - 35) \qquad para\ 35 \leq \varphi < 55° \qquad (\boldsymbol{eq.2.2})$$
$$L_{ori} = 4 + 0{,}114(\varphi - 55) \qquad para\ 55 \leq \varphi < 90° \qquad (\boldsymbol{eq.2.3})$$

- **f** – frequência (MHz);
- **d** – distância entre a estação rádio base receptora [km];
- **Δhm** – diferença entre a altura dos telhados dos prédios (ht) e a altura da antena da estação móvel (hm) em metros (Δhm = ht – hm);
- **Lori** – fator de correção devido à orientação da rua em função do ângulo de incidência φ [dB];
- **φ** – ângulo de incidência [graus];
- **w** – largura das ruas [metros].

$$L_{ms} = L_{bsh} + K_a \log(d) + K_f log(f) - 9 \log(b) \qquad (\boldsymbol{eq.3})$$

Onde:

$$L_{bsh} = -18 \log(1 + \Delta hb) \qquad para\ hb > ht \qquad (\boldsymbol{eq.3.1})$$
$$L_{bsh} = 0 \qquad para\ hb \leq ht \qquad (\boldsymbol{eq.3.2})$$
$$K_a = 54 \qquad para\ hb > ht \qquad (\boldsymbol{eq.3.3})$$
$$L_{bsh} = 54 - 0{,}8\Delta hb \qquad para\ d \geq 0{,}5\ km\ e\ hb \leq h \qquad (\boldsymbol{eq.3.4})$$
$$L_{bsh} = 54 - 0{,}8\Delta hb\left(\frac{d}{0{,}5}\right) \qquad para\ d < 0{,}5\ km\ e\ hb \leq h \qquad (\boldsymbol{eq.3.5})$$

- **f** – frequência (MHz);
- **d** – distância entre a estação rádio base receptora [km];
- **b** – distância entre os prédios ao longo do percurso da onda eletromagnética [m];
- **Δhb** – diferença entre a altura da antena da estação transmissora (hb) e a altura dos telhados dos prédios (ht) em metros (Δhb = hb – ht).

Vale lembrar que o modelo COST-231 Walfish-Ikegami é válido para as condições práticas a seguir:

- Altura das antenas de estação rádio base: [4, 50] m;
- Altura das antenas de estação móvel: [1, 3] m;
- Distância do enlace: [0,10 ; 3] km;
- Frequência: [900, 1800] MHz.

4.9.2. Modelo de LEE

Já este segundo, também é válido para frequências próximas de 900 [MHz], é um modelo empírico, que segundo seus experimentos levam em consideração o terreno, taxa de atenuação local (dB/década), nível do sinal recebido no ponto de interseção a 1,6 km da estação rádio base, entretanto não detalha, como o modelo anterior, à respeito das distâncias entre os prédios, suas respectivas posições e alturas, além de que como o modelo de Lee é experimental e adaptado de local a local podem haver algumas divergências mesmo com condições similares em locais diferentes.

A equação deste modelo é dada por:

$$\mathrm{Pr} = Po - \gamma \log\left(\frac{r}{r_0}\right) + Af + Geff_{h(he,h1)} + L + \alpha \qquad (\boldsymbol{eq.4})$$

Onde,

- **Pr** = Potência de recepção [dBm];
- **Po** = Potência de recepção no ponto de interseção [dBm];
- **γ** = Decaimento da atenuação de propagação ou fator de rugosidade do terreno [dB/década];
- **r** = Distância entre a estação-base e o móvel [km];
- $\mathbf{r_0}$ = Distância entre a estação-base e o ponto de interseção [km];
- **Geffh** – Ganho devido à altura efetiva [dB], valor dependente de he (altura efetiva da antena da estação-base [m]) e hl (altura real da antena da estação-base [m]);
- **L** = Perdas por difração no terreno – perdas por sombreamento [dB];
- **Af** = Fator de ajuste da frequência [dB];

- **α** = Fator de ajuste do sinal.

Para esta modelagem deve-se consideram também os seguintes aspectos:

- **Linha de Visada**: Se entre a estação rádio base e a estação móvel não há obstáculos, desta forma a altura efetiva da antena passa a ser um elemento de fundamental importância para o projeto.
- **Sem Linha de Visada (Sombreamento)**: Para este caso não há altura efetiva da antena, sendo assim as perdas são principalmente advindas da difração associada ao local da transmissão.
- **Existência de Água**: Quando o sinal vindo da estação rádio base chega ao móvel que passa por cima de água, deve ser considerado então uma atenuação de 20 [dB/década].

➢ **Elementos do Modelo de LEE**

- **Atenuação de Propagação Área a Área:** Um dos principais elementos deste modelo trata-se de um valor empírico do decaimento da atenuação do nível do sinal ao longo de todos os pontos de uma radial. Todavia podem ser utilizados valores de áreas similares a fim de evitar o retrabalho de efetuar mais medições. Na tabela 4 encontram-se exemplos desta atenuação.

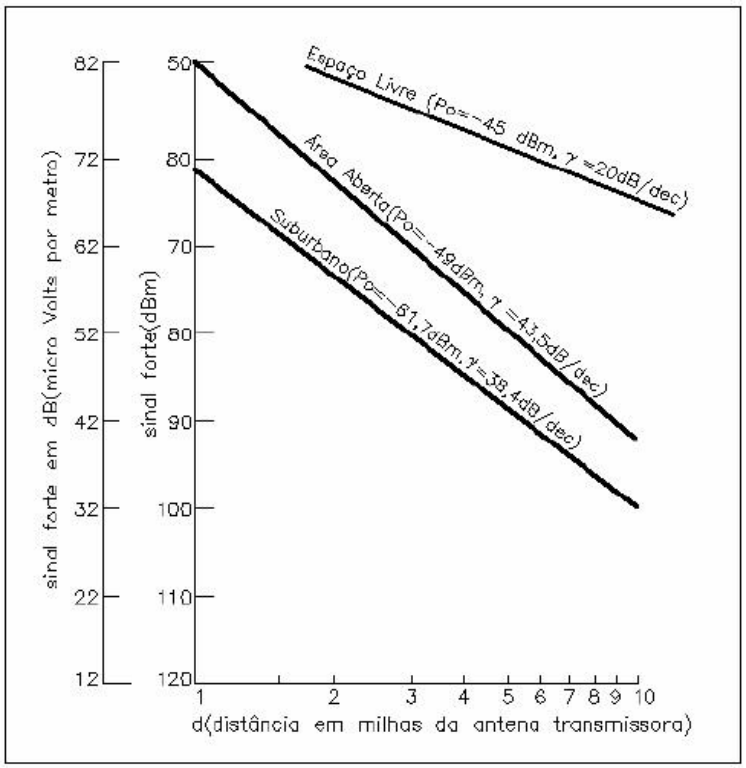
Tabela 4: Decaimento e ponto de interseção derivados de medidas reais

AMBIENTE	*Po* (dBm)	γ(dB/década)
Espaço Livre	-45,0	20,0
Área Aberta	-49,0	43,5
Suburbano	-61,7	38,4

Fonte: (PEREIRA, 2007)

A figura 33 mostra decaimento e o ponto de interseção obtidos de dados reais.

Figura 33: Decaimento e ponto de interseção derivados de medidas reais



Fonte: (PEREIRA, 2007)

- **Fator de Ajuste da Frequência (Af)**: Para este modelo a frequência de referência é a de 850 MHz, válido para três faixas de frequências distintas em dois ambientes diferentes, gerando assim valores de **Af** que podem variar dependendo das condições de projeto. A seguir é mostrada na tabela 5 seus valores.

Tabela 5: Fator de ajuste das frequências (Af)

BANDA	AMBIENTES	
	URBANO	RESTANTES
150 a 450 MHZ	Af = 8,29 - 20log(f / 850)	Af = - 20log(f / 850)
450 a 850 MHZ	Af = - 30log(f / 850)	Af = - 20log(f / 850)
850 a 2400 MHZ	Af = - 30log(f / 850)	Af = - 30log(f / 850)

Fonte: (PEREIRA, 2007)

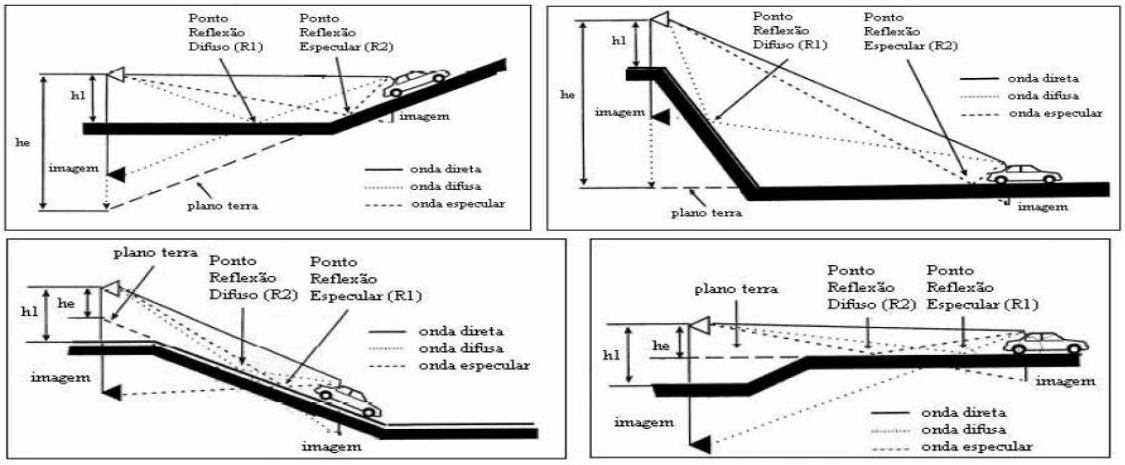
Com uma frequência de operação de 900 [MHz] utilizada neste projeto e com a não dependência do ambiente para esta frequência, pode-se então adotar:

$$A_f = -30log\left(\frac{f}{850}\right) \qquad (\boldsymbol{eq.4.1})$$

- **GANHO ASSOCIADO À ALTURA EFETIVA DA ANTENA (Geffh)**: O nível de potência do sinal recebido não depende apenas da altura que a antena se encontra em relação ao solo, mas também de sua altura efetiva em relação à estação móvel, assim como a distância entre si. Para realizar este cálculo do nível de potência do sinal é utilizado o método das imagens. A imagem da antena da estação de rádio base é ligada à antena da estação móvel, e o ponto que intercepta o terreno é considerado ponto de reflexão (**R1**) o contrário também é feito e o segundo ponto de interceptação com o terreno é o ponto (**R2**). Este modelo se utiliza do ponto de reflexão mais próximo à estação móvel e é exemplificado na Figura 34. O valor deste ganho que é associado à variação da altura efetiva da antena é dado por:

$$G_{effh} = 20log\left(\frac{he}{h1}\right) \qquad (\boldsymbol{eq.4.2})$$

Figura 34: Exemplo de aplicação do Método das Imagens



Fonte: (PEREIRA, 2007)

- **FATOR DE AJUSTE (α)**: Este fator tem por objetivo adequar previsões de cobertura ao longo das radiais do sinal a partir de valores reais da estação móvel e base. São realizados ajustes na potência de transmissão,

alturas dos centros de emissão das antenas da estação-base e móvel e ganhos das antenas da estação-base e móvel. E α é dado por:

$$\alpha = 10log\left(\frac{P_t}{P_{tp}}\right) + 20log\left(\frac{h_1}{h_{1p}}\right) + 10log\left(\frac{h_2}{h_{2p}}\right) + \left(G_r - G_{rp}\right) + \left(G_m - G_{mp}\right) \qquad (\boldsymbol{eq.4.3})$$

Onde,

Pt - Potência de transmissão da estação - base (W);

Ptp - Potência de transmissão padrão da estação - base (W);

h1 - Altura da antena da estação – base (m);

h1p - Altura padrão da antena da estação – base (m);

h2 - Altura da antena do móvel (m);

h2p - Altura da antena padrão do móvel (m);

Gr - Ganho da antena da estação - base (dBd);

Grp – Ganho padrão da antena da estação - base (dBd);

Gm - Ganho da antena do móvel (dBd);

Gmp – Ganho padrão da antena do móvel (dBd).

Os valores padrão para o modelo de Lee são:

Ptp – 10 W;

h1p - 30 m;

h2p – 3 m;

Grp – 6 dBd;

Gmp – 0 dBd.

- **PERDAS POR DIFRAÇÃO NO TERRENO (L)**: Também conhecida como **Perdas por Sombreamento**, deve ser levado em consideração quando há alguma obstrução parcial ou total do raio de visada direta. Esta perda é calculada pela teoria de Fresnel-Kirchoff, **L** é função do parâmetro adimensional **v**. Sua expressão é mostrada na equação 4.4.

$$v = -h_p\sqrt{\left(\frac{2}{\gamma}\right)\left(\frac{1}{r1} - \frac{1}{r2}\right)} \qquad (\boldsymbol{eq.4.4})$$

Onde,

hp - altura da obstrução acima ou abaixo do raio direto (m);

λ - comprimento de onda (m);

r1 - distância da estação - base ao obstáculo em lâmina (m);

r2 - distância do obstáculo em lâmina ao móvel (m).

Assim dependendo da faixa de valores que **v** pode assumir, o sombreamento **L** pode ser descrito por algumas funções:

$$L = 0; v > 1 \qquad (\boldsymbol{eq.4.5})$$

$$L = -20log(0{,}5 + 0{,}62v); 0 < v < 1 \qquad (\boldsymbol{eq.4.6})$$

$$L = -20\log(0{,}5^{0{,}92v}); -1 < v < 0 \qquad (\boldsymbol{eq.4.7})$$

$$L = -20\log\left(0{,}4 - \sqrt{(0{,}1184 - (0{,}1v + 0{,}38)^2}\right); -2{,}4 < v < -1 \qquad (\boldsymbol{eq.\ 4.8})$$

$$L = -20\log\left(-\frac{0{,}225}{v}\right); v < -2{,}4 \qquad (\boldsymbol{eq.4.9})$$

4.9.3. Cálculos da atenuação do sistema

Neste tópico é apresentado alguns cálculos referentes a projetos de atenuação do sistema de comunicação via rádio (ZigBee). Para o sistema em questão que é composto por uma antena base fixa, uma antena móvel acoplada ao protótipo em visada direta, considerando o Efeito Doppler, que nada mais é que a frequência aparente vista pela antena base fixa devido ao movimento do carro em relação a ela dependente da velocidade, comprimento de onda e ângulo de incidência do sinal. As equações **4.9** e **4.10** são baseadas em Ronaldo de Andrade Martins (2006) e Theodore S. Rappaport (1996). O sinal de soma e subtração da **4.11** é dependente se o veículo está aproximando-se ou afastando-se da antena base fixa respectivamente.

$$Fd = (v / \lambda) * cos(\theta) \qquad (\boldsymbol{eq.4.10})$$

$$Fa = f \pm Fd \qquad (\boldsymbol{eq.4.11})$$

Onde,

- **Fd** - Desvio da frequência devido ao Efeito Doppler [Hz];
- **Fa** - Frequência aparente do sistema [Hz];
- **f** - Frequência do sistema [Hz];
- **v** - Velocidade do móvel [m/s];
- **λ** - Comprimento de onda [m];
- **θ** - Ângulo de incidência do sinal [º].

Sabendo-se disto, pode-se então calcular o desvio máximo de frequência para o sistema em questão, que possui as seguintes condições de contorno:

- vmax = 150 km/h → **vmax = 540 m/s** ;
- **f = 900 MHz;**
- **λ** = $\frac{c}{f} = \frac{3.10^8}{900.10^6}$ → **λ = 0,333m**;
- **θmax = 0°.**

Onde,

- **vmax** - Velocidade máxima do protótipo;
- **c** - Velocidade da luz no vácuo;
- **θmax** - Ângulo de incidência para desvio máximo.

Desta forma obteve-se um Fd = 540/0,333 → **Fd = 1621,621 Hz** e como consequência um **Fa = 900,0016MHz** para o móvel se aproximando da antena base fixa e **Fa = 899,9984MHz** para o móvel se afastando da antena base fixa.

Ainda assim, é preciso verificar a Propagação Doppler (***Bd***), que nada mais é que a medida do aumento do espectro causado pela velocidade de mudança do canal de rádio móvel, e é definida como a gama de frequências em que o espectro de doppler recebido é essencialmente não nulo e também o Tempo de Coerência (***Tc***), que diz a taxa de transmissão do sinal em uma determinada distância. E para que a transmissão não seja afetada pela propagação **Bd** a taxa de transmissão deve ser maior que o valor de **Tc**, que é dada pela equação **4.12**.

$$Tc \approx \frac{9}{16\pi fm} \qquad (\mathbf{eq.4.12})$$

Onde f_m é dado por $f_m = v / \lambda$.

Neste caso Tc = 11,041 ms que implica numa transmissão de aproximadamente **9057 bps.** Assim como a taxa de transmissão mínima do ZigBee (10kbps) excede Tc então o sistema não sofre dispersão de frequência.

Então, percebe-se que devido à velocidade e frequência do sistema não serem tão elevadas o Efeito Doppler é mínimo para a frequência aparente do sistema. Logo, é possível então considerar como uma transmissão de dados fixa e em visada direta. A atenuação do espaço livre para antena fixa é dada pela equação **4.13** e foi retirada do livro Projeto de Sistemas Rádio de Edson Mitsugo Miyoshi, editora Érica, segunda edição, 2002.

$$Ao = 32{,}4 + 20log(d * f) \qquad (\boldsymbol{eq.4.13})$$

Onde,

- **Ao** - Atenuação do espaço livre;
- **d** - Distância entre as antenas;
- **f** - Frequência do Sistema (MHz).

Sendo que,

- d = 0,3 km;
- f = 900 MHz.

Assim tem-se **Ao = 81,027 dB**.

Segundo SILA (1983), a atenuação do sistema é dada pelas equações **4.14 e 4.15.**

$$As = Aftx + Agotx - Gtx + Ao - Grx + Aftx + Agotx \qquad (\boldsymbol{eq.4.14})$$

$$Prx = Ptx - As \qquad (\boldsymbol{eq.4.15})$$

Onde,

- **As** - Atenuação do sistema;
- **Aftx** - Atenuação de filtros e conectores de transmissão;
- **Agotx** - Atenuação de guias de onda de transmissão;
- **Gtx** - Ganho da antena de transmissão;
- **Afrx** - Atenuação de filtros e conectores de recepção;

- **Agorx** - Atenuação de guias de onda de recepção;
- **Grx** - Ganho da antena de recepção;
- **Prx** - Potência de Recepção;
- **Ptx** - Potência de Transmissão.

E sabendo-se que as atenuações devido a filtros, conectores, cabos e guias de onda somados, em média, são por volta de 3dB, logo como estes valores não estão disponíveis e nem se dispõe de equipamentos para realizar tal medição, foi atribuída então a atenuação destes elementos como se pode observar na equação **4.16**.

$$Aftx + Agotx + Afrx + Agorx = 3dB \qquad (\boldsymbol{eq.4.16})$$

Sabendo-se também que, as antenas são idênticas e de mesmo fornecedor de ganho igual a 5dBi, conforme o *datasheet* da antena. Então pode-se obter a atenuação do sistema $\mathbf{A_S}$.

$$\mathbf{As} = 3 - 5 + 81{,}027 - 5 \rightarrow \boldsymbol{As = 74{,}027dB}$$

Usando este valor de $\mathbf{A_S}$ na equação **4.15**, e também a potência de transmissão do módulo ZigBee 900 MHz, apresentada anteriormente na tabela 3, de valor de Ptx = 250mW, que para converter seu valor em potência para decibel, basta usar a equação **4.17**.

$$P(dB) = 10log(\,P(W) \,/\, 1mW\,) \qquad (\boldsymbol{eq.4.17})$$

Usando-se a potência apresentada na tabela 3, tem-se:

P(dB) = 10log(250m/1m) = 10log(250) = 10* 2,3979 = 23,979 dBm ≈ **24dBm**

Logo, pode-se calcular a potência de recepção para este projeto dada pela equação **4.15**.

$$Prx = Ptx - As = 24 - 74{,}027 \rightarrow \boldsymbol{Prx = -50{,}027dBm}.$$

Esta potência de recepção é suficiente para a transmissão, mesmo com eventuais atenuações no sistema, pois o limiar de recepção do ZigBee Digi Xbee-PRO DigiMesh 900HP - Conector RPSMA - Extended-Range é de -101dB, conforme especificação no site do fornecedor.

Entretanto a análise anterior não leva em consideração a movimentação do carro. Já o estudo de Walfish-Ikagami, apresentado no tópico 4.9.1, leva a velocidade do móvel em consideração em seus cálculos. Então um estudo deste projeto, tomando-se por base que as condições de contorno reais do sistema de telemetria estão de acordo com as deste tipo de projeto, então a seguir é mostrado seus respectivos cálculos.

Primeiramente é considerado o cálculo pela potência média deste modelo, pois como a transmissão irá ocorrer num local onde não há prédios ou casas nos arredores do sistema e que pode ser considerado a transmissão em visada direta, então cálculos como o **Lms** dado pela equação **3** não fazem muito sentido em serem calculados. A equação **4.18** representa a perda média de transmissão.

$$Lt = 42{,}6 + 26log(d) + 20log(f) \qquad (\boldsymbol{eq}, \mathbf{4.18})$$

Assim, como d = 0,3 km e f = 900 MHz, tem-se Lt igual a:

$$Lt = 42{,}6 + (-13{,}59) + 59{,}08 \rightarrow \boldsymbol{Lt = 99{,}16\ dB}.$$

Com esta informação pode-se então calcular a potência de recepção do sistema:

$Prx = Ptx - Lt = 24 - 99{,}16 \rightarrow \boldsymbol{Prx = -75{,}16\ dB}$, que também está de acordo com o limiar de recepção do módulo ZigBee apresentado.

E para se calcular a potência de transmissão real requerida por este estudo, é necessário calcular 3 (**três**) parâmetros que são: **Lo**, **Lrts** e **Lms**.

Sabendo-se disto pode-se então calcular **Lt** que é a soma destes três parâmetros, que são dados pelas equações **4.13**, **2** e **3** respectivamente.

Calculando-se então Lrts pela equação **2**, observa-se que há dependência de algumas condições de contorno como, w, f, Δ**h**m, que são respectivamente iguais a 3m, 900MHz, 3m (hb = 4m e hm = 1m) e considerando φ = 0° tem-se **Lori = -10**. Então Ltrs é igual a:

$$Lrts = -16{,}9 + 10log(3) + 10log(900) + 20log(4-1) + (-10)$$

$$\boldsymbol{Lrts = 16,95dB}$$

Agora calculando-se Lms pela equação **3** com as seguintes condições de contorno:

- d = 0,3 km;
- hb = 4m;
- ht = 15m;
- hm = 1m;
- f = 900MHz.

Desta forma como hb < ht tem-se Lbsh = 0.

Ka que é dada pela equação **3.5** tem seu valor igual à:

$$Ka = 54 - 0{,}8 * (4 - 1) * (0{,}3/0{,}5) \rightarrow \boldsymbol{Ka = 52,56\ dB}$$

Kd que é dada pela equação **4.19**, segundo Martins (2006). Modelagem e Medições de Ondas de Rádio para Predição de Perda em Ambientes Urbanos:

$$Kd = 18 - 15\left(\frac{dhb}{ht}\right) \qquad (\boldsymbol{eq.4.19})$$

$$Kd = 18 - 15(-9/15) \rightarrow \boldsymbol{Kd = 27.}$$

Kf que é dada pela equação **4.20**, segundo Martins (2006). Modelagem e Medições de Ondas de Rádio para Predição de Perda em Ambientes Urbanos:

$$Kf = -4 + 0{,}7\left(\frac{f}{925} - 1\right) \qquad (\boldsymbol{eq.4.20})$$

$$\boldsymbol{Kf = -4,02}$$

Agora é possível fazer o cálculo de **Lms** dada pela equação **3**

$$Lms = 0 + 52{,}56 + 27 * log(0{,}3) + (-4{,}02) * log(900) + 9log(b)$$

Como "b" é a distância entre os prédios ao longo do percurso do móvel, e não existem prédios ao longo deste percurso, o valor de: 9log(b) foi desconsiderado dos cálculos. Então:

$$Lms = 52{,}56 + (-14{,}12) + (-11{,}88) + 0 \rightarrow \boldsymbol{Lms = 26{,}26\ dB}.$$

E finalmente basta apenas o cálculo de **Lo** dada pela equação **4.13**

$$Lo = 92{,}4 + 20\log(d * f) = 92{,}4 + 20\log(0{,}3 * 0{,}9) \rightarrow \boldsymbol{Lo = 81{,}027\ dB}.$$

Finalmente com Lo, Lrts e Lms devidamente calculados, pode-se obter o valor da atenuação total do sistema Lt, que é dado por: $\boldsymbol{Lt = Lo + Lrts + Lms}$, então tem-se:

$$\boldsymbol{Lt = 124{,}23\ dB}$$

Então,

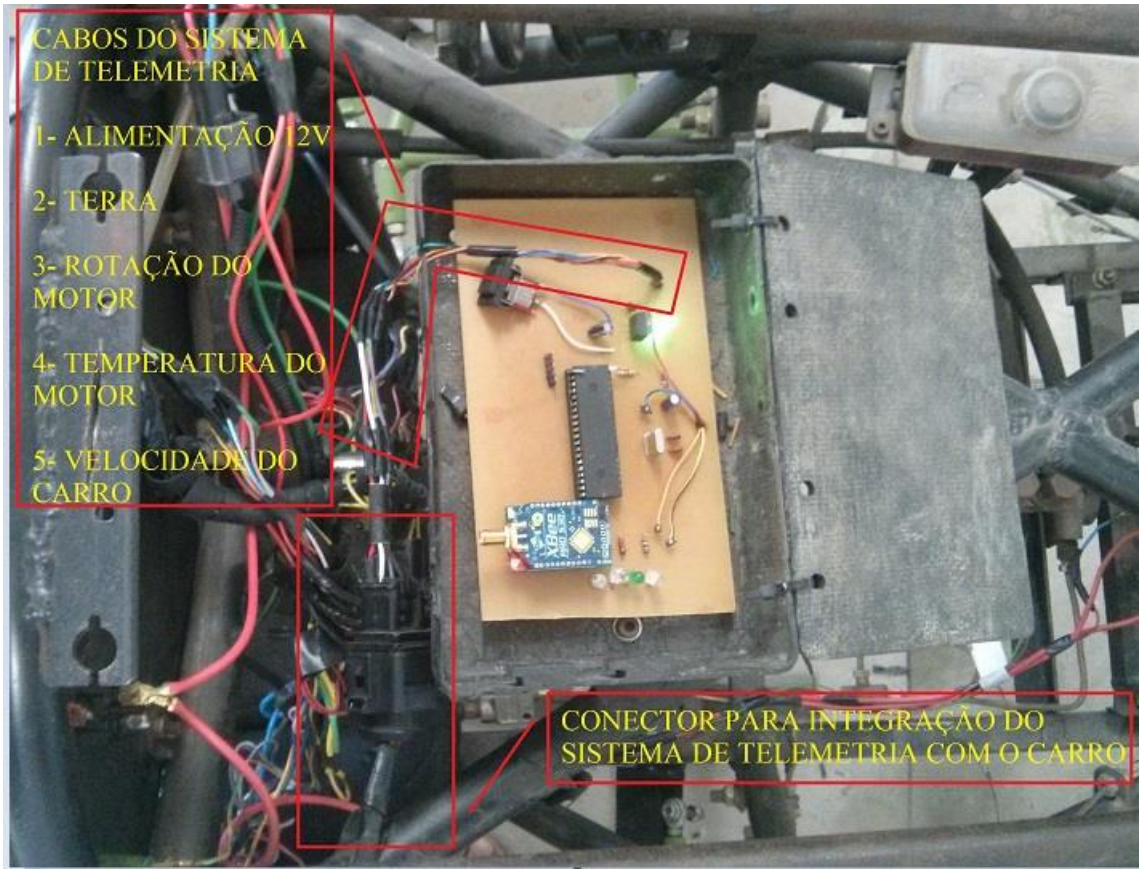
$$Prx = Ptx - Lt = 24 - 124{,}23 \rightarrow \boldsymbol{Prx = -100{,}23dB}.$$

E este valor também está de acordo com o limiar de recepção do módulo ZigBee em questão. Lembrando que neste cálculo foi considerado o valor de **Lrts** que é um fator devido à existência de prédios ao longo do percurso do móvel, então é um valor que pode ser desprezado, já que não existem prédios ao longo da transmissão, o que aumenta a margem para eventualidades. Assim chegamos a um novo valor de Prx igual a **- 83,29 dB**, que também é suficiente para que a transmissão ocorra de forma normal.

5. TESTES E RESULTADOS DO SISTEMA

Após os cálculos e montagem do circuito final, resta apenas integrar o sistema de telemetria ao carro. Para tal foi preciso conectar os 5 (cinco) cabos: alimentação, terra, velocidade, rotação e temperatura do motor no conector do sistema de telemetria nos cabos: vermelho, marrom, amarelo, laranja e azul respectivamente. Tais conexões podem ser vista na figura 35.

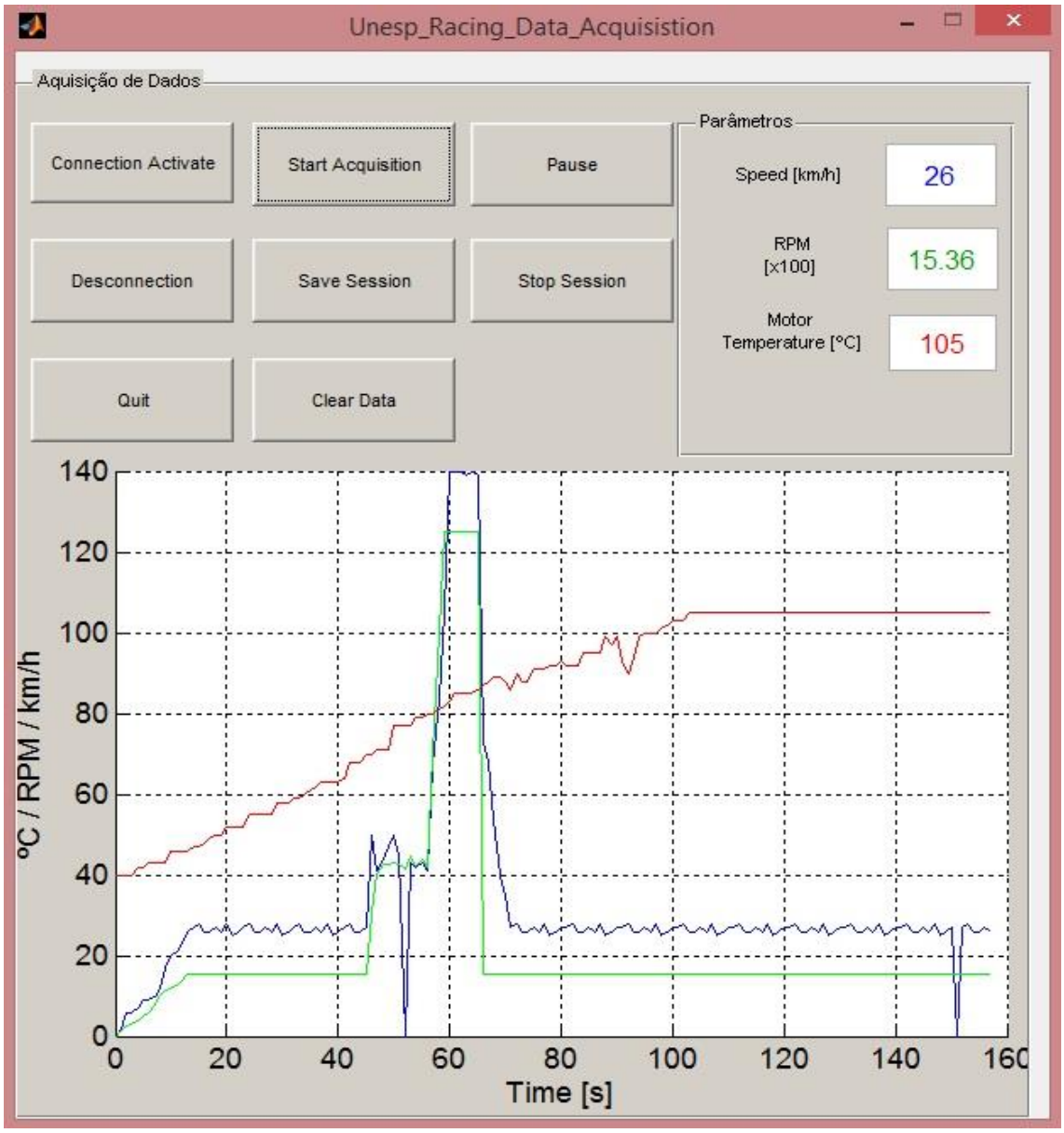
Figura 35: Conexão para integração do sistema de telemetria ao veículo.



Fonte: (AUTOR , 2014)

Com o sistema de telemetria integrado ao veículo, pode-se então capturar os sinais e visualizá-los na HID, como segue na figura 36, onde a curva da temperatura representada pelo numero [7], a curva da Rotação do motor pela numero [8] e por fim a curva da velocidade pelo numero [9].

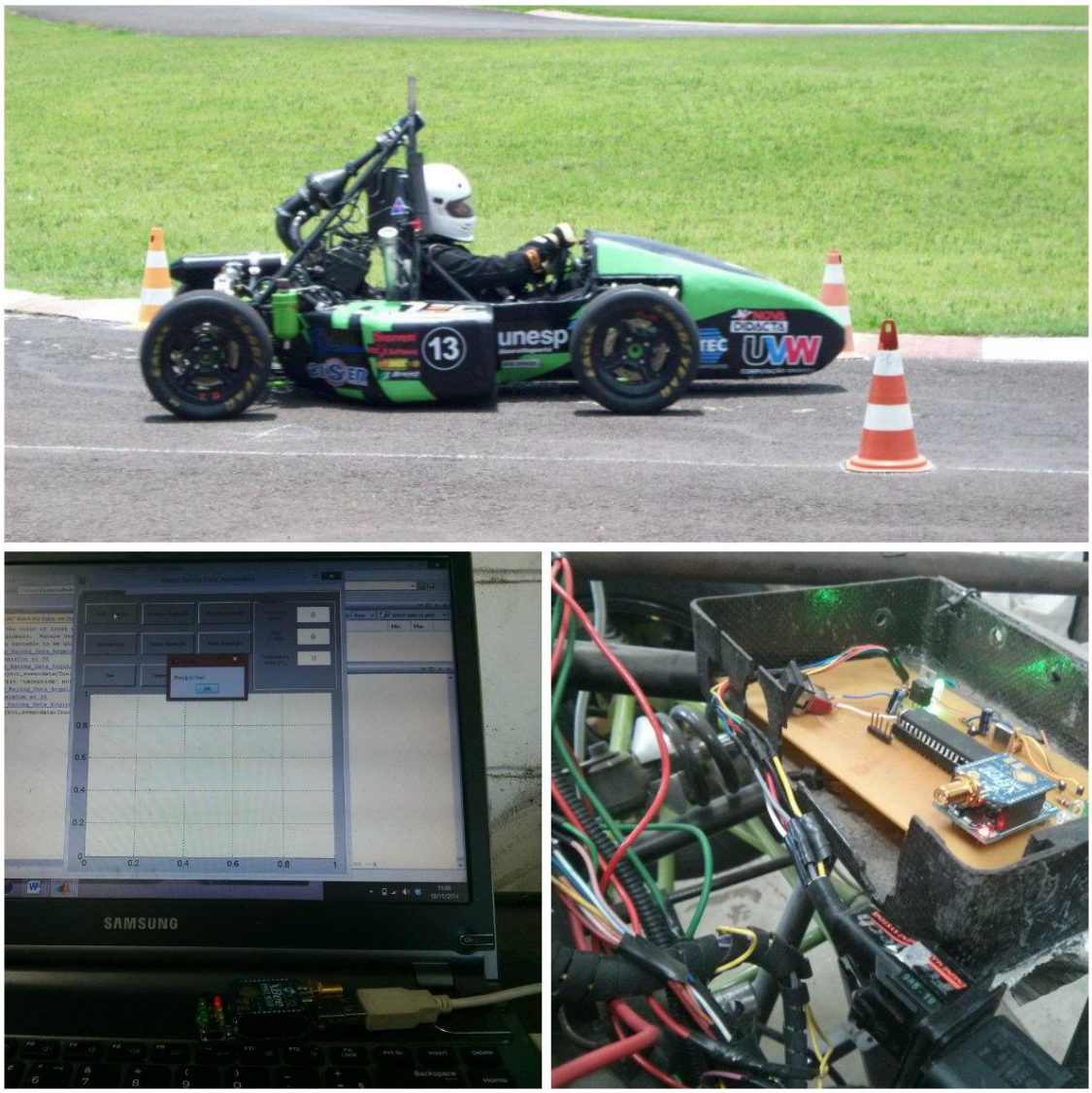
Figura 36: Visualização dos dados aquisitados na HID.



Fonte: (AUTOR , 2014)

Percebe-se então que o sistema de telemetria funciona conforme o esperado, capturando os dados de maneira simples, porém eficaz, e de fácil interpretação e manuseio da HID, tendo-se ainda a possibilidade de interpretar, posteriormente a sua aquisição. Os sinais são então exportados para o Excel quando estes são salvos e isto se mostrou ser de grande valia para os ajustes finos do protótipo, aumentando assim a eficiência e desempenho do carro e como consequência diminuindo o lapso que se tinha em ter de esperar parar o carro para se ter uma análise destas, hoje isto é feito em tempo real que permite ainda atuar de forma preventiva para que danos sejam minimizados.

Figura 37: Sistema de Telemetria Completo



Fonte: (AUTOR , 2014)

6. CONCLUSÕES

Neste trabalho foi apresentada uma forma simples, eficiente, sólida e de baixo custo de como elaborar um sistema de telemetria. Pode-se ter uma boa ideia sobre as necessidades mínimas para o correto funcionamento do microcontrolador, também teve-se uma boa compreensão de como os sensores utilizados na transmissão de dados se comportam e como cada um deve ser tratado antes de serem transmitidos e a descrição detalhada do que foi feito para que este projeto pudesse ser elaborado, implementado e testado.

Obteve-se uma boa noção de como utilizar, de forma clara e objetiva, os softwares que foram escolhidos para garantir que este trabalho fosse realizado com sucesso. Pode-se ver como configurar o MPLAB para que o código do microcontrolador fosse desenvolvido, compilado, testado e embarcado no microcontrolador; o ISIS e ARES utilizados em conjunto para projetar e confeccionar a placa de circuito impresso do sistema de telemetria, assim como todo seu processo de fabricação de baixo custo; o MATLAB para elaborar o código do sistema de aquisição de dados *"wireless"* para que o computador pudesse receber e mostrar os sinais em tempo real em forma de gráfico e displays, também com recursos de escolher quando salvar a aquisição para uma análise posterior, pausar e parar uma aquisição, bem como limpar os dados para evitar ou eliminar erros na transmissão, tendo também como opção habilitar ou desabilitar a conexão do sistema de telemetria com o computador; o software X-CTU para configurar os módulos ZigBee de transmissão e recepção de maneira simples, rápida e eficaz; Driver da FTDI para fazer com que o dispositivo CON-USBEE seja reconhecido pela porta USB do computador como uma porta virtual COM de entrada de dados para se realizar uma transmissão serial virtual para que o MATLAB possa reconhecer os dados que chegam advindos do sistema de telemetria.

Alguns projetos de antenas foram apresentados, mostrando as diferenças existentes entre cada um dos modelos apresentados, destacando-se o modelo COST 231 que viabiliza a transmissão de maior alcance com maior confiabilidade com uma antena de 5 dBi multidirecional, tanto na transmissão quanto na recepção, em um local de visada direta, sem obstáculos ou objetos que interfiram na transmissão.

Após a montagem do sistema no veículo, foram realizados ensaios no carro e com os primeiros resultados foram feitas melhorias no sistema, como uma melhor apresentação dos dados na HID, velocidade de transmissão, modo como a palavra a ser

transmitida é elaborada, facilitando tanto sua transmissão quanto sua recepção. Outro avanço obtido com os primeiros resultados foi também melhorar a disposição dos dados aquisitados no Excel para estudos posteriores.

Por fim, os resultados obtidos se mostraram satisfatórios e coerentes com os cálculos realizados neste estudo, possibilitando uma análise em tempo real da velocidade, da temperatura e rotação do motor, mostrando-se muito útil no auxílio da manutenção e correção dos ajustes do protótipo em tempo real a fim de melhorar o seu desempenho instantaneamente. Já o armazenamento dos dados para análise posterior permite uma comparação entre todos os tipos de ajustes do carro, permitindo assim a existência de um ajuste ótimo para cada situação de corrida que o veículo possa se encontrar.

Em virtude do que foi mencionado anteriormente, pode-se concluir que se obteve um sistema de telemetria eficiente, de baixo custo, fácil manutenção, implementação e utilização do sistema, atendendo todos os pontos de operação requisitados pelo sistema, permitindo assim a evolução esperada com a implementação deste sistema. E ainda possibilitando que o sistema seja expandido permitindo que o usuário possa aumentar o número de sinais aquisitados, como por exemplo a tensão da bateria, pressão do óleo, temperatura de entrada e saída do radiador, curso de suspensão, temperatura e pressão dos pneus, posição angular do volante, entre muitos outros dados que podem ser de grande utilidade para melhorar ainda mais a manutenção e regulagem do carro ou ainda melhorando a qualidade dos componentes do veículo.

REFERÊNCIAS

ANTONIO, Marco. Apostila de Programação de Microcontroladores PIC Usando Liguagem C. Vitória, 2006. 97 p.

DANTAS, Rafeal. **Apostila**. Disponível em: <http://dc378.4shared.com/img/elwoVzex/preview.html)>. Acesso em: 24. Set. 2014

DESMONTA & CIA. **Zigbee ou IEE 802.15.4 - Conheça a tecnologia a fundo**. Disponível em: <http://desmontacia.wordpress.com/2011/02/22/zigbee-ou-ieee-802-15-4-conhea-a-tecnologia-a-fundo/>. Acesso em: 13 mai. 2013

DIGI INTERNATIONAL. **X-CTU Configuration & Test Utility Software: User´s Guide.** 2008. 16 P.

FIGUEREDO, Luis Felipe C. Tutorial XBee. Brasília:LAVSI - UnB, 2008. 74 p.

FUTURE TECHNOLOGY DEVICES INTERNATION Ltd. - FTDI CHIP. **Application Note AN_234: FTDI Drivers Installation guide for Windows 8.** Version 1.0. United Kindom: Glasgow Park, 2013 27 p.

GOMES, Alcides Tadeu, TELECOMUNICAÇOES – TRANSMISSÃO RECEPÇÃO AM-FM: SISTEMAS PULSADOS. 17ª ed. São Paulo: Érica, 2001. 415p.

KEC. **SEMICONDUCTOR TECHNICAL DATA: KIA7805AP~KIA7824AP.** Revision 1. 2010. 20 p.

MARTINS, Ronaldo de Andrade. Modelagem e Medições de Ondas de Rádio para Predição de Perda de Propagação em Ambientes Urbanos. 2006. 135 p. Tese (Doutorado em Engenharia Elétrica) - Universidade Federal do Rio Grande do Norte, Universidade Federal do Rio Grande do Norte, Natal, 2006.

MICROCHIP TECHNOLOGY. **PIC18F2420/2520/4420/4520 Data Sheet.** DS39631E. Arizona: Chandler, 2008. 410 p.

PEREIRA, Marco Antonio Betini. Análise de Modelos de Propagação na Área Urbana da Região de Curitiba - PR na Faixa de Frequência de 1800 MHz. 2007. 142 p. Dissertação (Mestrado em Engenharia Elétrica, com Ênfase em Telecomunicações) - Universidade Federal do Paraná - UFPR, Universidade Federal do Paraná, Curitiba, 2007.

RAMOS, Jadeilson de Santana Bezerra. Instrumentação Eletrônica sem Fio. 1ª ed. São Paulo: Érica, 2012. 240 p.

RAPPAPORT, Theodore S. WIRELESS communications: Principles & Practice. 1ªed. New York: IEEE PRESS, 1996. 641 p.

ROGERCOM. **Manual do Adaptador CON-USBBEE Xplus.** Versão Julho/2012. Alagoas, 2012. 5 p.

ROGERCOM. **Manual do Adaptador PROTO-BEE.** Versão 1.1. Alagoas, 2012. 9 p.

SILA, Gilberto; BARRADAS, O. . Sistemas Radiovisibilidade, 2ª ed. São Paulo: LTC, 1983. 294 p.

VIKA CONTROLS. **Módulos XBee: Família XBee.** Disponível em: <http://www.vikacontrols.com.br/wp-content/uploads/downloads/folder-wireless.pdf>. Acesso em: 24 set.2014.

XBEE STORE. **Produtos**. Disponível em: <http://xbeestore.lojavirtualfc.com.br/listaprodutos.asp?IDLoja=16187&IDProduto=3947074&q=modulo-digi-xbee-pro-digimesh-900hp---conector-wire---extended-range>. Acesso em: 19/04/2014.

ZANCO, Wagner da Silva. Microcontroladores PIC18 com Linguagem C. 1ª ed. São Paulo: Érica, 2010. 448 p.

ZIGBEE ALIANCE INC. **Telecom Applications Profile Specification.** Version 1.00. San Ramon: CA, 2009-2010. 210 p.

APÊNDICES

APÊNDICE A – Código Microcontrolador (C18)

```
//Bibliotecas utilizadas no projeto
#include <p18f4520.h>
#include <delays.h>
#include <stdio.h>
#include <string.h>
#include <usart.h>
#include <stdlib.h>
#include <capture.h>
#include <timers.h>

//Configuração dos fusíveis do PIC
#pragma config OSC = HS //Oscilador externo > 4MHz
#pragma config FCMEN = OFF
#pragma config IESO = OFF
#pragma config BOREN = OFF
#pragma config MCLRE = ON
#pragma config WDT = OFF
#pragma config LVP = OFF

//Declaração dos protótipos de funções
void Inic_Reg(void);
void high_isr(void);
void low_isr(void);
void putsUART( char * buffer );
void Configura_AD(void);
void interrupt_at_high_vector(void);
void interrupt_at_low_vector(void);
unsigned int ReadCapture1( void );
```

```
void OpenCapture1( unsigned char config );
void RPM(void);

//Declaracao das Variaveis Globais e Definicoes
        volatile unsigned float vPulsosVel = 0;
        volatile unsigned float vPulsosRPM = 0;
        volatile unsigned float vOverflow = 0;

        volatile unsigned int doSendData = 0;

        float vTemp = 0;//Varivel para guardar o Valor da Temperatura do Motor
        volatile unsigned float vVel = 0;
        volatile unsigned float vRPM = 0;
        volatile unsigned char contTMR0 = 0;

        volatile unsigned int UpdateVel = 0;
        volatile unsigned int UpdateRPM = 0;
        volatile unsigned int UpdateTemp = 0;
        volatile unsigned float vVoltas = 0;

        unsigned int pulsos = 0;
        int LenTx, Fill, Preen;

        unsigned char Tx[15];
        unsigned char FinalTx[15];
        unsigned char AtualChar;

        unsigned char TxTemp[4];
        unsigned char TxVel[4];
        unsigned char TxRPM[6];
        float TrasmitDados[];

        int c=0,contagem, contPV=0;
```

```
	unsigned int long result, result1;

	char str[17];
	char str1[18];

	int i;
	int contUV = 0;
	long lWhole=0; // Stores digits left of decimal
	unsigned long ulPart=0; // Stores digits right of decimal

	//Elementos Fisicos
	#define NDentesRPM 11 //numero de dentes da roda fônica rotação, 11 dentes = 1 volta
	#define NDentesVel 6 //numero de dentes da roda fônica velocidade, 6 dentes = 1 volta

	#define Raio_Roda 0.3556 //Raio 14' => 14*25,4mm = 355.6mm --> 0.3556m
	#define Comprimento_Roda (2*3.141598*Raio_Roda)//Perimetro da roda

	#define Ch_Temp 0

//Função principal do projeto
void main(void)
{
	Inic_Reg();
	Configura_AD();

//Inicialização do Timer 0
	TMR0H = 0xED;
	TMR0L = 0xB2;

//Laço infinito para aquisição dos dados
	while(1)
```

```
        {

//ATUALIZANDO VALORES PARA ENVIAR

//*****************************************************************************
*****************************************************************************
***
//Lendo Valor Velocidade(após 2ciclos de TMR0)
                if(UpdateVel == 1)
                {
                        TrasmitDados[3] = vVel;
                        UpdateVel = 0;
                }
//*****************************************************************************
*****************************************************************************
***
//Lendo Valor Rotação do Motor
                if(UpdateRPM == 1)
                {
                        TrasmitDados[2] = vRPM;
                        UpdateRPM = 0;
                }
//*****************************************************************************
*****************************************************************************
***

                //Habilita transmissão dos dados quando doSendData = 1
                if(doSendData)
                {
                        if (TXSTAbits.TRMT)
                        {

//Desabilita Funções para transmissão
```

```
                INTCONbits.GIE = 0;//Interrupções Globais

                PIE1bits.TMR1IE = 0;//Timer 1

                PIE1bits.CCP1IE = 0;//CCP1
                PIE2bits.CCP2IE = 0;//CCP2

                PORTDbits.RD4 = 1;//Ascende LED para mostrar transmissão

        //Converte valores numéricos em Strings
                itoa(vTemp,TxTemp);
                itoa(TrasmitDados[2],TxRPM);
                itoa(TrasmitDados[3],TxVel);

        //Concatena strings dos dados para transmissão
                sprintf(Tx,"%s;%s;%s;",TxTemp,TxVel,TxRPM);

        //Verifica se a palavra para transmissão contém 15 caracteres
                for (i = 0 ; i <=14; i++)
                {
                //Le números separados por ";" e conta os mesmos
                        AtualChar = Tx[i];
                        if(';' == AtualChar)
                        {
                                contPV = contPV + 1;
                        }

                //Preenche com "X" caso a palavra tenha menos de 15 caracteres
                        if (contPV>=3 & i<13)
                        {
```

```
                Tx[i+1] = 'X';
            }

        //Coloca-se "." para garantir o final da palavra a ser transmitida
            if(i==14)
            {
                Tx[i] = '.';
                contPV = 0;
            }
        }

    //Envia a palavra de 15 caracteres pela porta UART
        putsUSART(Tx);

    //Apaga LED para mostrar que a transmissão acabou
        PORTDbits.RD4 = 0;


    //Apaga variáveis para próxima aquisição
        for (i=0;i<=5;i++)
        {
            if (i<=3)
            {
                TxVel[i] = 0;
                TxTemp[i] = 0;
            }

            TxRPM[i] = 0;
        }

        for (i=0;i<=14;i++)
        {
```

```
                    Tx[i] = 0;
                    FinalTx[i] = 0;
                }

                //Habilita Funções para transmissão
                INTCONbits.GIE = 1;

                PIE1bits.TMR1IE = 1;

                PIE1bits.CCP1IE = 1;
                PIE2bits.CCP2IE = 1;

                doSendData = 0;//Desbilita transmissão até a aquisição dos próximos 3 dados
            }
        }

//#########################################################################
#########################################################################
########################################
//Pisca MainLed
        PORTDbits.RD0 = 1;

        PORTDbits.RD0 = 0;
//#########################################################################
#########################################################################
########################################
//Leitura do Sensor de Temperatura

//*****Checagem da Temperatura
        //Considerando:
        // 5V -> 40ºC
        // 0V -> 120ºC          -> Temp = 120 - ((120-40)/5)*Tensao
```

```
                vTemp = ADCRead(Ch_Temp);
                vTemp = 120 - 16*(vTemp/1000);//Equacao de Conversao de
Temperatura

//Verifica temperatura e acende LED
                if (vTemp > 100)
                {
                        PORTDbits.RD1 = 1;
                }
//Verifica temperatura e apaga LED
                if (vTemp <= 100)
                {
                        PORTDbits.RD1 = 0;
                }
//#####FIM Checagem da Temperatura
//#################################################################
//*****************************************************
        }
}
////////////////////////////////////////////////////////
//Função de Inicialização das variáveis
void Inic_Reg(void)
{
  // Reset das Variaveis Globais
                vPulsosVel = 0, vVel = 0;
                vPulsosRPM = 0, vRPM = 0;
                vOverflow = 0,          contagem = 0;

// 0 - Saida || || 1 - Entrada

//Pino AN0 como entrada analógica(Temperatura do Motor)
        TRISA = 0x01;
```

```
//Pino RB2 como entrada(Interrupção Velocidade) || Pino RB1 como entrada(Interrupção Rotação do Motor)
        TRISB = 0x00000000;
        TRISC = 0x11000110;//Fazendo Pinos de de CCP1(17) e CCP2(16) como Inputs e Habilitando TX(26) e RX(27)
        TRISD = 0x00;
        TRISE = 0x00;

        ADCON1 = 0x00; //0000 1111 pinos AN0-AN12 digitais

//APAGA LEDS
        PORTA = 0;
        PORTB = 0;
        PORTC = 0;
        PORTD = 0;
        PORTE = 0;

// 0 - DESABILITA NIVEL DE PRIORIDADE || 1 - HABILITA NIVEL DE PRIORIDADE
        RCONbits.IPEN = 1;

//HABILITA INTERRUPÇÃO GLOBAL E DE ALTA PRIORIDADE
        INTCONbits.GIEH = 1;
        INTCONbits.GIE = 1; // 1 -->Interrupções globais Habilitadas || 0 -->Interrupções globais Desabilitadas

//Setup dos Timers para os CCPs
//Tempo da captura registrado no CCPRxH CCPRxL
        OpenCapture1( C1_EVERY_RISE_EDGE & CAPTURE_INT_ON );
        OpenCapture2( C2_EVERY_RISE_EDGE & CAPTURE_INT_ON );
        OpenTimer1( TIMER_INT_ON & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_8 & T1_SYNC_EXT_OFF & T1_SOURCE_CCP & T1_OSC1EN_OFF );
```

```
//Pré setup do TMR1 para estouro a cada 0,125s
/*
        Fosc = 16000000 Hz - Prescaler = 8

        (65535 - x)*(1/(16000000/4) * 8 = 0,125
        (65535 - x) * 2^(-6) = 0,125
        65535 - x = 62500
        x = 3035 -->> 0BDB

        Se o estouro acontece a cada 0,125 segundos basta o TMR1 estourar 1/0,125 = 8
vezes
        então teremos o estouro a cada 1s
*/

//Pre-set do Timer 1 para 1s para contar os pulsos de velocidade e rotação
                TMR1H = 0b00001011;
                TMR1L = 0b11011011;
//Limpa flag para próxima contagem
                PIR1bits.TMR1IF = 0;

PIE1 = 0b00000101;//Habilita CCP1(2) e TMR1(0)
//Limpa flag para próxima contagem
PIR1bits.CCP1IF = 0;

PIE2 = 0b00000001;//Habilita CCP2(0)
//Limpa flag para próxima contagem
PIR2bits.CCP2IF = 0;

//TIMER0 DESABILITADO
        INTCONbits.TMR0IE = 0;

//SET PARAMETROS TMR0 EM CONJUNTO
        T0CON = 0b00000111;
```

```
//Contas para Reset TMR0:
/*
        FreqOsc = 20MHz --> 20M/4 = 5M|| 8
        (5M)^(-1)*prescaler(256) = 51,2us

        FreqOsc = 16MHz --> 16M/4 = 4M
        (4M)^(-1)*prescaler(256) = 64us -->> ()300ms - EDB2)

        Ciclo de Estouro = n*51,2u = TempEst
        onde n = incrementos do TMR0 e TempEst = Tempo de Estouro desejado(100ms ou 300ms ou qualquer outro valor)

        n = 300m/51,2u = 5859

        256*256- n -->> 65540 - 5859 = 0d59680 ==> 0xE920
*/

// Reset 100ms => 0xF85C || 300ms => E920 || 80ms => F9EA
                TMR0H = 0xED;
                TMR0L = 0xB2;
                INTCONbits.TMR0IF = 0;

//Configurando TX/RX e BAUDRATE do PIC
        TXSTA = 0b00100100;
        RCSTA = 0b10010000;
        BAUDCON = 0b00000000;

/* SYNC = 0 ; BRGH = 1 ; BTG16 = 1 --- 011
//Configurando Baud Rate

Fosc/(M*BaudRate) - 1
M = 16 --> BGRH = 0 --- M = 4 --> BGRH = 1 ----- 16bits
```

```
20M/(4*38400) - 1 = 103,2 --> 103

103/16 = 6,438 --> 0,438*16 = *7*  /\
6/16 = 0,375 --> *6*               ||

0d103 = 0h67 = 0b 00000000 01100111

0d130 = 0h62
*/

//0h822 = 0d2082 --> 0b 00001000 00100010 = 0x8
//  exemplo: 259 ==> 1:3 00000001 00000011 => 1*2^8 + 2 + 1
//Congigurando o valor 9600 de Baud Rate 0d520 --> 00000010 00001000 --> 0h28 de
16bits
        //SPBRGH = 0x11;

        //valor 51 para oscilador de 8MHz interno(pag 387)
//para 16MHz e BRGH = 0 usar 103 para 9600
//para 16MHz e BRGH = 0 usar 25 para 38400
        SPBRG = 103;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////

//Chamada da Interrupção Baixa Prioridade
#pragma code low_vector=0x18 //0x18 --> baixa prioridade de interrupção || 0x08 -->
alta prioridade de interrupção
void interrupt_at_low_vector(void)
{
```

```
        _asm
                GOTO low_isr
        _endasm
}
#pragma code
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////

//Chamada da Interrupção Alta Prioridade
#pragma code high_vector=0x08 //0x18 --> baixa prioridade de interrupção || 0x08 -->
alta prioridade de interrupção
void interrupt_at_high_vector(void)
{
        _asm
                GOTO high_isr
        _endasm
}
#pragma code

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//Interrupção de Baixa Prioridade
#pragma interruptlow low_isr
void low_isr(void)
{

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////

#pragma interrupt high_isr
void high_isr(void)
{
```

```
//ARRUMAR A CONTAGEM PARA CAPUTRA DOS PULSOS!!! --- RPM

//CCP1
        if(PIR1bits.CCP1IF == 1)
        {
        //Desabilita periféricos para contagem
                PIE1bits.TMR1IE = 0;
                PIE1bits.CCP1IE = 0;

        // Limpa a flag para nova captura e conta em seguida
                PIR1bits.CCP1IF = 0;
                vPulsosRPM++;

        //Habilita perifericos para contar
                PIE1bits.TMR1IE = 1;
                PIE1bits.CCP1IE = 1;
        }

//Em caso de estouro do Timer 1
        else if(PIR1bits.TMR1IF == 1)
        {
                PIE1bits.TMR1IE = 0;          // Desabilita interrpções de TMR1
                PIE1bits.CCP1IE = 0;          // Desabilita interrupção - periféricos CCP1

                TMR1H = 0b00001011;       // Inicializa os registradores do TMR1
                TMR1L = 0b11011011;
                contagem ++;

                PIR1bits.TMR1IF = 0;          // Limpa a flag do TIMER1
                PIE1bits.CCP1IE = 1;          // Habilita interrupção - periféricos CCP1
                PIE1bits.TMR1IE = 1;          // Habilita interrpções de TMR1
        }
```

```
//Ocorrendo 8x o estouro, significa que 1s se passou
        if (contagem == 8)
        {
//Zera contador para próxima aquisição
                contagem = 0;

//Chama função de contagem
                RPM();
        }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////

        //CCP2
        if(PIR2bits.CCP2IF == 1)
        {
                //Desabilita periféricos para contagem
                PIE1bits.TMR1IE = 0;
                PIE2bits.CCP2IE = 0;

        // Limpa a flag para nova captura e conta em seguida
                PIR2bits.CCP2IF = 0;
                vPulsosVel++;

        //Habilita perifericos para contar
                PIE1bits.TMR1IE = 1;
                PIE2bits.CCP2IE = 1;
        }

//Em caso de estouro do Timer 1
                else if(PIR1bits.TMR1IF == 1)
                {
                        PIE1bits.TMR1IE = 0;          // Desabilita interrpções de TMR1
```

```
                    PIE2bits.CCP2IE = 0;          // Desabilita interrupção - periféricos
CCP1

                    TMR1H = 0b00001011;           // Inicializa os registradores do
TMR1
                    TMR1L = 0b11011011;
                    contagem ++;

                    PIR1bits.TMR1IF = 0;          // Limpa a flag do TIMER1
                    PIE2bits.CCP2IE = 1;           // Habilita interrupção - periféricos
CCP1
                    PIE1bits.TMR1IE = 1;          // Habilita interrpções de TMR1

                }
}

//Set Leitura de Temperatura do Motor
void Configura_AD(void)
{
        ADCON0=0b00000001;
        ADCON1=0b00001110;//AN0 input Analogico -- (VREF+ = VDD = 4,97V) --
(VREF- = VSS = 0V) --> passo por ºC = 19,41mV = 4,97/256
        ADCON2=0b10101001;
}

//Função de leitura da Temperatura do Motor
int ADCRead(unsigned char ch)
{
  int ValADL,ValADH;
  if(ch>13) return 0;  //Invalid Channel

        ADCON0bits.GO = 1;
        while(ADCON0bits.GO); //wait for the conversion to finish
```

```
        ADCON0bits.GO = 0;
   //The DATA register of the ADC has 10bits 0bHH LLLL LLLL; (HH)->ADRESH;(LLLL LLLL)->ADRESL
   ValADL = ADRESL;
   ValADL = ValADL*5/1.024; //each bit = 4.8828125mV =~ 5mV ; CorrectionFactor => 1.024
   ValADH = ADRESH;
   ValADH = ValADH*5*256/1.024;//This register may be 0x1;0x2;0x3;CorrectionFactor => 256

   return (ValADH+ValADL);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////
void RPM(void)
        {
//.....
...
        //Calculo RPM
                vPulsosRPM = (vPulsosRPM/11);
                vRPM = vPulsosRPM * 60;

                vPulsosRPM = 0;
                UpdateRPM = 1;

//.....
...
        //Cáculo Velocidae
                vPulsosVel = vPulsosVel/6;
                vVel = vPulsosVel * Comprimento_Roda;// Obtendo velocidade em [m/s]
```

```
                vVel = vVel * 3.6; //Obtendo Velocidade em [km/h]

                vPulsosVel = 0;
                UpdateVel = 1;
//.....
...
        //Habilita envio dos dados
                doSendData = 1;
        }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
```

6.1.APÊNDICE B – Código Aquisição de Sinais (MATLAB)

```
function varargout = Unesp_Racing_Data_Acquisistion(varargin)
% UNESP_RACING_DATA_ACQUISISTION MATLAB code for
Unesp_Racing_Data_Acquisistion.fig
%      UNESP_RACING_DATA_ACQUISISTION, by itself, creates a new
UNESP_RACING_DATA_ACQUISISTION or raises the existing
%      singleton*.
%
%      H = UNESP_RACING_DATA_ACQUISISTION returns the handle to a new
UNESP_RACING_DATA_ACQUISISTION or the handle to
%      the existing singleton*.
%
%
UNESP_RACING_DATA_ACQUISISTION('CALLBACK',hObject,eventData,handles
,...) calls the local
%      function named CALLBACK in UNESP_RACING_DATA_ACQUISISTION.M
with the given input arguments.
%
%      UNESP_RACING_DATA_ACQUISISTION('Property','Value',...) creates a new
UNESP_RACING_DATA_ACQUISISTION or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Unesp_Racing_Data_Acquisistion_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Unesp_Racing_Data_Acquisistion_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help Unesp_Racing_Data_Acquisistion

% Last Modified by GUIDE v2.5 10-Jun-2014 10:03:14

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

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

% --- Executes just before Unesp_Racing_Data_Acquisistion is made visible.
function Unesp_Racing_Data_Acquisistion_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Unesp_Racing_Data_Acquisistion (see
VARARGIN)
```

```
% Choose default command line output for Unesp_Racing_Data_Acquisistion
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

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

end

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

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

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

%Vetor tempo
   j = 0;
```

```
graphIt = '';
%x = 0:1:180;
%y = x+2;

%Habilita Start/Re-Start da aquisição
global StopButton
%StopButton = 0;

global Has_Open Dados_UsbSerial
if Has_Open == 1
   %Fazer Laço com o Botão de Stop
   while(StopButton == 0)

      j = j + 1;
      global j

    %  if FirstTime == 0%TALVEZ NÃO SEJA MAIS PRECISO!

      %Para pular as 2 primeiras aquisições para adequar a recepção
     %     FirstTime = FirstTime + 1;
     %     graphIt = fgets(Dados_UsbSerial);
     %else

    %Guarda da variavel o que esta na PortCOM(6) já Adequada para leitura
    graphIt = fscanf(Dados_UsbSerial,'string',16);

      %   for p = 1:1:4

            %graphIt = fgets(Dados_UsbSerial);

            %Le Txt para TESTES
            %graphIt = '95;0;2000;0000';
```

```
 %  end
%end

t = 0:1:(j-1);
global t

%[a b c] = strread(teste, '%s %s %s', 'delimiter', ';')
[tempm vel rpm resto] = strread(graphIt, '%s %s %s %s', 'delimiter', ';');

%Convertendo de Valores de celulas para Strings
if isempty(tempm)
   tempm = '0';
else
   tempm = tempm{1};
end

if isempty(vel)
   vel = '0';
else
   vel = vel{1};
end

if isempty(rpm)
   rpm = '0';
else
   rpm = rpm{1};
end

%Somente aceita Char
set(handles.Velocidade,'String',vel);
set(handles.RPM,'String',rpm);
set(handles.TempM,'String',tempm);
```

```
        pause(0.001);

        %Converte para Número para plotar no Gráfico
        vel = str2num(vel);
        rpm = str2num(rpm)%/100;
        tempm = str2num(tempm);

        V(j) = vel;
        global V

        R(j) = rpm;
        global R

        T(j) = tempm;
        global T

        %pause(0.15);
        hold on;
        grid on;

    %Grágico Velocidade
        plot(t,V,'b');
    %Grafico RPM
        plot(t,R,'g');
    %Gráfico Temperatura
        plot(t,T,'r');

        hold off;
    end
else
    W = warndlg('PortCom connection NOT ESTABLISHED!');
    waitfor(W);
```

```
    end
end

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



% --- Executes on button press in sair.
function sair_Callback(hObject, eventdata, handles)
% hObject    handle to sair (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    %MGuide = 'C:\Users\Gustavo\Desktop\TG - Telemetria FSAE -
Mamute\Softwares\TRANSMISSÃO DE DADOS\TG_matlab_data_aquisition';
    %fclose(MGuide);

    choice = questdlg('Are you sure?','Quit','Yes','No','No');
    waitfor(choice);
    switch choice
        case 'Yes'

            global Dados_UsbSerial;

            try
                fclose(Dados_UsbSerial);
                clear global Dados_UsbSerial;
            catch exceptionClose

            end
```

```
        MsgBox('Closing...','Quit');
        pause(2);

        close all;

    case 'No'
        M = MsgBox('Click on the button to continue...','Quit');
        waitfor(M);
  end
end

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

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

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

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

```
end


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

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

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

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

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

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

```
end


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

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


% --- Executes on button press in LimparDados.
function LimparDados_Callback(hObject, eventdata, handles)
% hObject    handle to LimparDados (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    exceptionClaen = '';
try
    set(handles.Velocidade,'String','0');
    set(handles.RPM,'String','0');
    set(handles.TempM,'String','0');

    clear global t
    clear global j
    clear global T
    clear global V
```

```
    clear global R

    cla;
    clear global Has_Open

    global Dados_UsbSerial
    fclose(Dados_UsbSerial);
    clear global Dados_UsbSerial

catch exceptionClaen

end

    if exist('exceptionClaen','var') == 1

        MC = msgbox({'All Cleared!','','The Connection has Stoped','Try Another
one'},'Cleaning...');
        pause(3);
        close(MC);

        clear all
        clc

    elseif ~isempty( regexp(exceptionClaen.message,'Dados_UsbSerial'))
        EM = errordlg({'Failure when trying to Clean Dados_UsbSerial!' 'Already closed
or not Exist anymore!'},'Cleaning Error');
        waitfor(EM);
    end
end


% --- Executes on button press in PararSessao.
function PararSessao_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to ParарSessao (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global StopButton
    StopButton = 1;
    global StopButton

    ComecarSessao_Callback()
    M = MsgBox('Stoped Session','Stop Acquisition');
end


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

    StopButton = 1;
    PB = MsgBox('Session paused, click continue...','Pause');
    uiwait(PB);
    %StopButton = 0;
    %global StopButton

    %ComecarSessao_Callback();
end


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

```
%HABILITANDO CONEXÃO COM PORTCOM PARA TRANSMISSÃO DE
DADOS
global StopButton
StopButton = 0;
global StopButton
%Teste para Abrir PortCom sem que Interropa a Execução por conta de Erros
   try
     fopen(Dados_UsbSerial);
   catch exception
      Ult_Err = exception.message;

      %Tipos dos Erros
      Not_Create = regexp(Ult_Err,'Undefined function or variable "Dados_UsbSerial"');
      Has_Open = regexp(Ult_Err,'OBJ has already been opened');
   end

   if Not_Create == 1

      M = MsgBox('Creating PortCom connection...','Opening PortCom');

      global Dados_UsbSerial
        %Captura Diretamente os dados enviados ao ZigBee pela porta USB
         if isempty(Dados_UsbSerial)
            Dados_UsbSerial = serial('COM6','BaudRate',9600);

            %Setup da Captura dos Dados Transmitidos ao PC
            set(Dados_UsbSerial,'Parity','none');
            set(Dados_UsbSerial,'DataBits',8);
            set(Dados_UsbSerial,'StopBits',1);
            set(Dados_UsbSerial,'Terminator','CR');

            %Determina a quantidade de bit a ser lido na COM
```

```
            set(Dados_UsbSerial,'inputbuffersize',16);

            %Abre a porta COM
            fopen(Dados_UsbSerial);
            close(M);

            %Flag para PortCom Criada -->> Not_Create = 0(PortCom Criada) ||
Not_Create = 1(PortCom Não Criada)
            Not_Create = 0;

            %Flag Para Porta aberta
            Has_Open = 1;
            global  Has_Open
            %CONFERIR SE PORTCOM É ABERTA PELA PRIMEIRA VEZ!!!!
            FirstTime = 0;

            delete exception;

            M = MsgBox('PortCom Created Successful!','Opening PortCom');
            pause(3);
            Close(M);

            M = MsgBox('Ready to Start','Opening PortCom');
            waitfor(M);

        elseif strcmp('open',Dados_UsbSerial.Status) == 1
            Close(M);

             Has_Open = 1;
             global Has_Open

             W = warndlg('PortCom Already Opened!');
             pause(2);
```

```
            Close(W);
        else
            Close(M);
            EM = errordlg('Failure when trying to Connect. Please try again','Connection
Error');
            waitfor(EM);
        end

    end
end


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

    choice = questdlg('Are you sure?','Closing PortCom','Yes','No','No');
    waitfor(choice);

    switch choice
        case 'Yes'

        try
            fclose(Dados_UsbSerial);
             clear global Dados_UsbSerial;
        catch exceptionDesc

        end

        if exist('exceptionDesc','var') == 1
```

```
        if regexp(exceptionDesc.message,'Undefined function or variable')
            erro = errordlg(exceptionDesc.message,'Closing PortCom ERROR');
            pause(2)
            close(erro);

        else
            W = warndlg('PortCom Already Disconnected!');
            pause(2);
            Close(W);

        end
    else
        MsgBox('PortCom Closed and Erased','Closing PortCom');
        Has_Open = 0;
        global Has_Open

        pause(2);
    end

    case 'No'
  end
end


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

NewCreate = 1;
choice = questdlg('Are you sure?','Save and Export to Excel file','Yes','No','No');
```

```
waitfor(choice);

 switch choice
   case 'Yes'

      CaminhoWS = 'C:\FSAE Telemetry\Sessions\';
      mkdir(CaminhoWS);

      FileName = 'Unesp_Racing_Session_Day_';
      SheetName = 'Session_';

      Data = date;
      CompleteFileName = strcat(FileName,Data);

      contFile = 1;


      %Verificando Existencia de Arquivo e criando se necessário
      allFiles = dir(CaminhoWS);
      allLibNames = {allFiles.name};
      [aLf,aCf] = size(allLibNames);

      for contAllLib = 1:aCf
         LibName = allLibNames{1,contAllLib};

         %Compara as Datas dos Arquivos, se a data for a mesma do PC
         %cria somente mais uma planilha na Pasta de Trabalho Excel
         rmdl = regexp(LibName, Data);

         if ~isempty(rmdl)
            M = MsgBox('File Found!','Save as');
            pause(3);
            close(M);
```

```
        NewCreate = 0;
    else
        if contAllLib == aCf & NewCreate ~= 0;
            M = MsgBox('Creating a new File...','Save as');
            waitfor(M);

            NewCreate = 1;

            Cabecalho = [{'Time[s]'} {'Temperature[ºC]'} {'Speed[km/h]'} {'RPM'}];
            SearchFile = strcat(CaminhoWS,CompleteFileName);
            SheetName = strcat(SheetName,'1');

            XLSWRITE(SearchFile,Cabecalho,SheetName,'A1:D1');
        end
    end
end

SearchFile = strcat(CaminhoWS,CompleteFileName);

%Criando Nova Planilha no respectivo Arquivo para cada Session
[status,sheets] = xlsfinfo(SearchFile);
NumSheets = size(sheets,2);

if NewCreate == 0
    for contSheet = 1:NumSheets
        SName = sheets{1,contSheet};
        rmdl = regexp(SName, SheetName);

        if ~isempty(rmdl)
            %Contador para quantidade de Planilhas "Session_"
            contFile = contFile + 1;
        end
```

```
    end

    SNum = num2str(contFile);
    SheetName = strcat(SheetName,SNum);

    global t
    global V
    global T
    global R

    if length(V)+1 == length(t)
        p = length(V);
        V(p+1) = V(p);

        p = length(T);
        T(p+1) = T(p);

        p = length(R);
        R(p+1) = R(p);
    end


    Matriz = [t' T' V' R'];
    SM = size(Matriz,1) + 1;

%CABEÇALHO
    SearchFile = strcat(CaminhoWS,CompleteFileName);
    Cabecalho = [{'Time[s]'} {'Temperature[ºC]'} {'Speed[km/h]'} {'RPM'}];
    XLSWRITE(SearchFile,Cabecalho,SheetName,'A1:D1');

%DADOS
    SMn = num2str(SM);
    Range = strcat('A2:','D',SMn);
```

```
    XLSWRITE(SearchFile,Matriz,SheetName,Range);

    M = MsgBox('Session Created!','Save as');
    pause(3);
    close(M);

else

    global t
    global V
    global T
    global R

    if length(V)+1 == length(t)
        p = length(V);
        V(p+1) = V(p);

        p = length(T);
        T(p+1) = T(p);

        p = length(R);
        R(p+1) = R(p);
    end

    Matriz = [t' T' V' R'];
    SM = size(Matriz,1) + 1;

%CABEÇALHO
    SearchFile = strcat(CaminhoWS,CompleteFileName);
    Cabecalho = [{'Time[s]'} {'Temperature[ºC]'} {'Speed[km/h]'} {'RPM'}];
    XLSWRITE(SearchFile,Cabecalho,SheetName,'A1:D1');
```

```
        %DADOS
            SMn = num2str(SM);
            Range = strcat('A2:','D',SMn);

            XLSWRITE(SearchFile,Matriz,SheetName,Range);

            M = MsgBox('Session Created!','Save as');
            pause(3);
            close(M);

        end

    case 'No'
 end
end
```