MYBIKE: UM COMPUTADOR DE BORDO PARA BICICLETAS DE BAIXO CUSTO

Osmar Silva
osmar.rubert@gmail.com

Victor Oliveira
victor.oliveira023@gmail.com

Mateus Vanzin
mateus.vanzin@gmail.com

Departamento de Engenharia da Computação
Universidade Federal do Amazonas

Resumo

Este artigo apresenta uma proposta de um computador embutido em uma bicicleta usando microprocessador Arduino Uno. O computador de bordo da bicicleta é responsável por medir a velocidade atual, a quilometragem total, o tempo total de viagem, bem como realizar um auto-diagnóstico do sistema em caso de falha, informando ao usuário quais itens da bicicleta devem ser trocados. Todas as informações estarão disponíveis ao usuário através de um display LCD e um botão de modo para visualização das informações. Assim, um computador de bordo para bicicletas dotado de funcionalidades é proposto.

Palavras-chaves: Mybike, Arduino, Sensor Hall, Computador de bordo, bicicleta.

1 Introdução

MyBike é um computador de bordo dotado de funcionalidades, como medição de velocidade, medição de quilometragem e auto-diagnóstico do sistema como um todo. Esse sistema ainda provê uma autonomia de energia através do uso de baterias, um display de cristal liquido (lcd) para mostrar informações aos usuários, um botão de navegação denominado de MODO para trocar as informações mostradas no display e um botão de RESET que ajustará o contador da quilometragem atual de volta para zero.

O computador de bordo proposto ainda contará com um sensor de efeito hall instalado no garfo da roda dianteira e um imã que ficará preso ao raio da roda dianteira. Esse sensor coletará as informações de distância e mandará de tempos em tempos para o microcontrolador arduino que por sua vez mandará a informações para o display.

O artigo foi organizado como segue: a seção dois contém a descrição do problema encontrado e a solução encontrada; a seção três traz os objetivos propostos a serem alcançados com o projeto MyBike; na seção quatro temos uma visão geral do funcionamento de todo o projeto enquanto que na seção cinco mostramos a metodologia utilizada; a seção seis mostra o desenvolvimento do projeto e explicamos detalhadamente o seu funcionamento;

na seção sete temos os resultados obtidos e, na seção oito, encerramos com a conclusão do projeto e os trabalhos futuros.

2 Descrição do Problema

Com o passar dos anos, constata-se que a sociedade tem ficado cada vez mais obesa, e tal fato tem chamado atenção de muitas pessoas. Em um artigo do G1 [6], constata-se que, em 2011, 52,6% dos homens estão acima do peso. Tantos os profissionais da saúde, como a própria população está tentando reverter esse quadro e buscando soluções para emagrecimento. Uma dessas soluções é a prática de esportes e atividades físicas que vem ganhando muitos adeptos nos últimos anos, entre os esportes os mais procurados são o ciclismo.

Segundo o professor de Educação Física, Roberto Toscano, pedalar melhora o condicionamento físico, aumenta a capacidade cardio-respiratória e a prática está entre as mais apropriadas na prevenção e tratamento de doenças como: hipertensão, colesterol alto, infarto do miocárdio, entre outras. Muitos especialistas também apontam que uma hora de pedaladas é possível perder aproximadamente entre 300 e 500 calorias, mas isso pode variar de acordo com a carga, velocidade e esforço empregados durante o treino.

Como essa prática de pedalar vem se tornando hábito da população, um computador de bordo poderá trazer informações relevantes para os ciclistas, tais como quanto tempo leva um circuito, quantos quilômetros foram percorridos, qual a velocidade atingida, e entre outras informações bem úteis como quantidade de calorias perdidas. Essas informações poderiam surgir para servir de interesse para os frequentadores do esporte e é, nesta linha, que o projeto MyBike está sendo proposto.

3 Objetivos

- Criar um dispositivo capaz de fornecer informações ao ciclista.
- Criar um protótipo de computador de bordo para uma bicicleta para prova de conceito.
- Fazer um produto eficiente a baixo custo comparado aos dispositivos existentes no mercado.
- Fazer um software de tempo real, obedecendo às restrições temporais e gerando um autodiagnostico de todo o sistema.
- Relatar os resultados obtidos com o método proposto.

4 Visão Geral

O projeto MyBike apresenta uma metodologia baseada em um arquitetura simples, no qual propõe que seja construído um sistema modular, capaz de realizar manutenções de módulos sem que seja comprometido todo o sistema.

A solução constará de um módulo controlador e disponibilizador de informações que será instalado no guidão da bicicleta, e um módulo de sensoriamento magnético que será instalado na roda da bicicleta.

O módulo controlador é dotado de um microprocessador atmega 328 com bootloader do arduino, programado para coletar informações do módulo sensor e disponibilizar para o usuário. Tal módulo consiste de rotinas de interrupção que serão acionadas a cada vez que o módulo magnético mandar uma informação. Ainda o módulo Controlador terá disponível dois botões, um denominado de MODO, no qual consiste de realizar a troca dos estados disponíveis no display LCD, como por exemplo: quilometragem atual e velocidade atual, como mostra a Figura 1, e o outro botão chamado RESET, o qual volta os contadores de viagem e tempo para zero.

Figura 1. Módulo Controlador.

O módulo sensor consiste de um imã e um sensor magnético. O imã será fixado no raio da roda, e o sensor será fixado no garfo de sustentação da roda. Dessa forma, a cada vez que a roda der uma volta completa, o imã passará pelo sensor ocasionando a troca de estado do sensor e, com isso a informação chegará até o microcontrolador para posteriormente ser transmitida ao ciclista. Como mostra a Figura 2, abaixo.

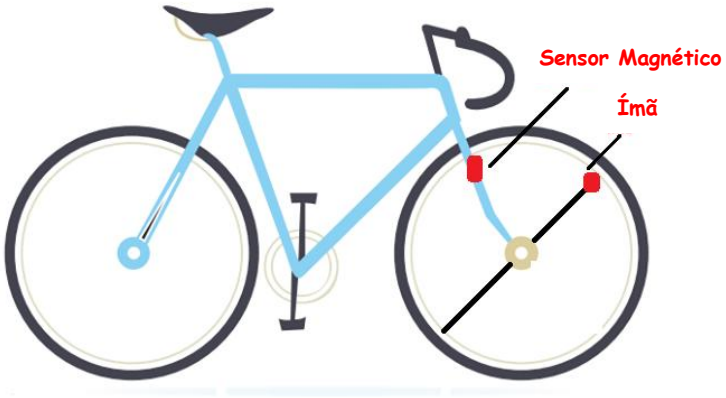


Figura 2. Módulo sensor.

5 Metodologia

Para atingir os objetivos propostos o projeto seguirá um cronograma macro especificado na tabela 1. Dessa forma faz-se necessário em primeiro momento realizar um levantamento bibliográfico, para que se possa aprender mais sobre a plataforma de trabalho arduino, além de encontrar metodologias já usadas em outros projetos, para usar como base de conhecimento para o desenvolvimento do projeto. Realizadas as pesquisas, o próximo passo é adquirir os componentes necessários para o desenvolvimento do hardware e juntamente com isso trabalhar no desenvolvimento do software do sistema.

A terceira etapa consiste em continuar o desenvolvimento do software e iniciar o processo de desenvolvimento do hardware, com isso espera-se que no inicio de fevereiro o dispositivo já está pronto em sua concepção de hardware e melhorar o software adicionando atividades de auto-diagnóstico, entre outras atividades cabíveis ainda nessa etapa, e para concluir as atividades dessa terceira etapa, orçar o gabinete do protótipo. A quarta e última etapa consiste em fechar as atividades que por ventura ficarem abertas, preparar apresentação do produto e apresentá-lo aos interessados.

Todas as atividades propostas para o desenvolvimento do projeto passará por um acompanhamento, para verificação e validação do cronograma, ao final do projeto. Os fontes do projeto estarão sobre controle de versão, para evitar percas de dados no decorrer do projeto, e possibilitar ainda desfazer alterações mal sucedidas que possam vir a ocorrer durante a fase de desenvolvimento do mesmo.

Ao todo serão 3,5 meses de trabalho, o mês inicial do projeto será destinado ao levantamento bibliográfico, bem como as definições dos componentes a serem utilizados, os dois meses seguintes serão destinados para o desenvolvimento do hardware e do software e

para finalizar as ultimas semanas do projeto serão destinadas a elaboração da apresentação do produto.

Atividades	Nov	Dez	Jan	Fev	Mar
Reuniões do grupo semanais	X	X	X	X	X
Escolha e aquisição dos componentes	X				
Apresentação parcial do projeto		X	X	X	X
Aquisição da bicicleta		X			
Construção do Hardware: ligações do sensor com o microcontrolador		X	X		
Comunicações do microcontrolador com sensor e interface (display)		X	X		
Construção do software: Arquitetura de interrupções			X		
Ligações dos botões de interrupção com o microcontrolador			X		
Desenvolvimento do artigo			X	X	
Construção do Software: Testes e ajustes				X	
Construção do Hardware: Testes e ajustes				X	
Desenvolvimento da Apresentação Final				X	
Testes Finais					X
Finalização do artigo					X
Finalização da Apresentação final					X

Tabela 1. Cronograma de Atividades.

6 Desenvolvimento

O desenvolvimento em si do projeto consistiu em duas frentes principais: hardware e software. Optamos por trabalhar essas duas frentes separadamente e, em toda reunião semanal, relatar o que foi avançado na semana para sincronizar o projeto.

A seguir trataremos com detalhes de cada parte do projeto, de modo claro e específico. Relatamos tanto a parte de software como a de hardware para um melhor entendimento.

a) Processador

O processador de tempo real funciona da seguinte forma: Toda vez que uma nova informação chega ao processador, ele manda para o LCD a fim de atualizar a tela, caso seja necessário. Neste tempo, pode sofrer interrupções do sensor de efeito hall, botão *mode,* botão *reset*, interrupção do tempo ou diagnósticos.

O processador é responsável pela integração de todo o sistema, ou seja, é o núcleo do projeto. Utilizamos o processador do Arduino Uno para realizar esta tarefa. O Arduino é um processador muito potente para o que estamos trabalhando e ele

conseguiu, como veremos na próxima seção, atingir todos os requisitos temporais previstos.

A figura a seguir descreve bem a tarefa do processador. Ele possui várias entradas - sensor de efeito hall, botões mode e reset, interrupções de tempo e diagnóstico – e apenas uma saída: o LCD. Como podemos observar, ele capta todas as informações e mostra ao usuário aquela que é do seu interesse.

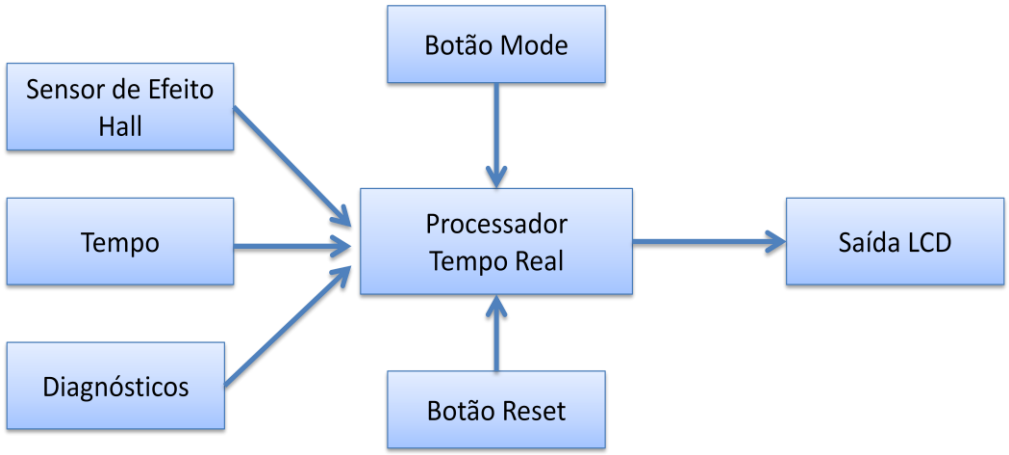


Figura 3: Visão geral do Processador do MyBike.

Na parte de configurações do software, iniciamos as variáveis utilizadas e as interrupções, tais como tempo, velocidades, distâncias, raio da bicicleta e etc, enquanto que no *loop* principal, o processador executa apenas três tarefas que serão mostradas a seguir.

```
float RAIO = 0.32;

//Variaveis Globais
volatile int modo = 0; //Modo atual
/*
  MODOS DISPONIVEIS:
*/
int MODO_VIAGEM = 0; // Distância da Viagem atual
int MODO_VELOCIDADE = 1; // Velocidade da Viagem
int MODO_DISTANCIA_TOTAL = 2; // Distância Total
int MODO_TEMPO = 3; // Tempo de Viagem
int MODO_VELOCIDADE_INSTANTANEA = 4;

// Pinagem
```

```
// BOTAO MODE NO PINO 2
// SENSOR DE EFEITO HALL NO PINO 3
// Reset no pino 13
int resetPin = 13;

void setup()
{
  lcd.begin(16, 2); //Inicia o LCD com dimensões 16x2(Colunas x Linhas)
  attachInterrupt(0, mode_int, RISING);
//Inicia Interrupção de mode com o método RISING
  attachInterrupt(1, wheel_int, RISING);
//Inicia Interrupção do sensor de efeito hall com o método RISING
  Timer1.initialize(1000000); //A cada um segundo
  Timer1.attachInterrupt(soma_tempo); //Chamar a função soma_tempo()
  pinMode(resetPin,INPUT); //Iniciando pino do reset
  Serial.begin(9600);
}
```

```
void loop()
{
  verificaModo(modo);
  verificaPinoReset();
  verificaDiagnosticos();
}
```

O código completo está disponível em anexo ao artigo.

b) Sensor de Efeito Hall

O *sensor de efeito hall* funciona como um botão normalmente aberto. Quando o ímã, que está preso ao aro, passa perto do sensor que está preso ao garfo da roda, o circuito interno fecha e manda o sinal *HIGH* para a porta de interrupção do *Arduino.* O *Arduino* está pronto para atender a esta interrupção com a maior prioridade, e acrescenta, à variável de distância e distância total, o valor do comprimento do pneu, calculado pela fórmula:

$$D = 2 * \pi * r$$

Onde r é a distância entre o ponto central do pneu e o sensor.

A execução da rotina de interrupção do sensor leva, em média 1 milissegundo. Entretanto, ocorre um fenômeno em que o sensor lê várias atenuações de voltagem em

apenas uma interrupção, fazendo várias contagens em uma única interrupção. A figura 3 mostra este fenômeno.

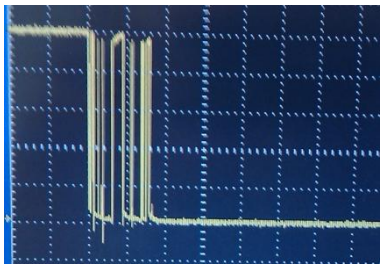


Figura 4. Sinal de entrada de uma interrupção no osciloscópio.

Para contornar este fenômeno, indicamos, via *software*, que o sistema só deve entender uma nova interrupção após 10 milissegundos da primeira. Assim, o sensor é capaz de atender a 100 interrupções por segundo, isso faz com que a velocidade máxima que o sistema consegue atingir é de:

$$V_{máx} = 2 * \pi * r * 100$$

$$V_{máx} \cong 628 * r \ [m/s]$$

Para um raio de 32 centímetros, o sistema chega a uma velocidade máxima de mais de 200 km/h, mais do que suficiente para uma bicicleta.

Na parte de software, a função executada quando uma interrupção é acionada é a que segue:

```
//Interrupcao sensor de efeito hall
void wheel_int()
{
  wheel_time = millis();
  if (wheel_time - last_wheel_time > 20)
  {
    distancia += distanciaPneu;
    distanciaTotal += distanciaPneu;
    gap_wheel_time = wheel_time - last_wheel_time;
    last_wheel_time = wheel_time;
  }
}
```

Podemos verificar que gravamos o tempo em que a interrupção foi executada pela última vez para não acontecer o fenômeno descrito acima. Além disso, adicionamos o valor da distância do pneu ao contadores.

c) Botão *Mode*

O botão *Mode* também está conectado a uma interrupção do *Arduino*, e funciona de uma maneira semelhante ao sensor de efeito *hall*. Ao ser pressionado, o sistema muda a exibição do LCD, começando pela viagem, seguindo a sequência da figura 4, abaixo:

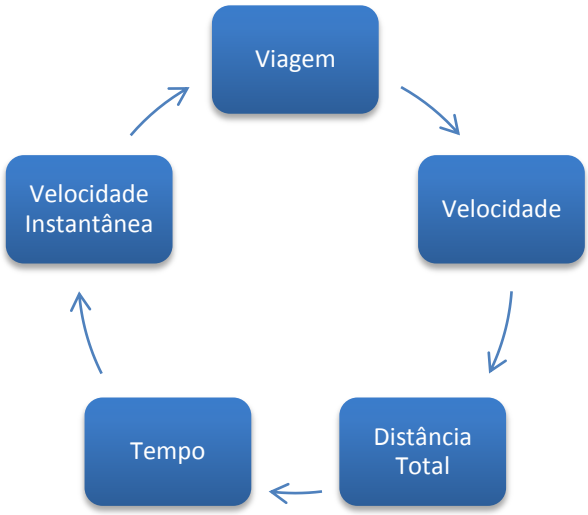


Figura 4. Sequência mostrada no LCD ao pressionar o botão *mode*.

O botão *mode* também sofre o mesmo fenômeno de receber várias interrupções em um curto intervalo de tempo quando o botão é pressionado e damos a mesma solução descrita na sessão do sensor de efeito hall.

A função executada pela interrupção do botão mode é bem simples. Verificamos o tempo de execução da última interrupção e adicionamos uma unidade à variável modo até atingir a quantidade máxima de modos, quando então a variável é zerada, fechando o ciclo. A seguir mostramos o código da interrupção.

```
//Interrupcao do botao mode
void mode_int()
{
  button_time = millis();
  if (button_time - last_button_time > 250)
  {
    if (modo == QTD_MODOS)
      modo = 0;
    else
      modo++;
    last_button_time = button_time;
  }
}
```

d) Botão Reset

O botão *Reset* é feito com interrupção de software e, por isso, tem prioridade menor que os componentes citados acima. Sua funcionalidade é *resetar* ou zerar o tempo e a distância percorrida quando pressionado. Ao receber um diagnóstico, é este botão que deve também ser pressionado para sair da tela de exibição do diagnóstico.

```
//Funcao do pino reset
void reset_int()
{
  //Resetar tudo quando pressionado:
  distancia = 0.0;
  segundo = 0;
  minuto = 0;
  hora = 0;
}
```

e) LCD

O LCD 16x2 é a saída do sistema. Uma rotina foi criada para verificar o modo atual e exibi-lo na tela. Tem a menor prioridade do sistema, ou seja, só é executado quando o processador está livre (na prática, o processador está a maioria do tempo livre, como mostraremos nos resultados). A tela do diagnóstico é exibida na tela até que o ciclista aperte o reset, mas enquanto isso não ocorre, o sistema continua funcionando.

Para executar no sistema, deve-se, primeiramente, declarar a biblioteca:

```
#include <LiquidCrystal.h> //Inclui a biblioteca do LCD
```

Após a inclusão da biblioteca, declaramos um objeto LCD:

```
LiquidCrystal lcd(9, 8, 5, 4, 7, 6); //Configura os pinos do
Arduino para se comunicar com o LCD
```

E ainda, chamamos a função dentro de setup():

```
lcd.begin(16, 2); //Inicia o LCD com dimensões 16x2(Colunas
x Linhas)
```

f) Interrupção de Tempo

A interrupção de tempo tem o objetivo de calcular o tempo decorrido de uma viagem. Esse tempo pode ser visto na opção 04 (ver Botão Mode) e também é útil para calcular a velocidade média da viagem. Para que possamos calcular o tempo sem influenciar os

outros serviços do sistema, utilizamos uma biblioteca especializada em calcular interrupções de tempo para o Arduino. A bliblioteca é a “TimerOne.h”. Nela, podemos chamar uma função toda vez que completar um determinado tempo.

Assim, fizemos uma função que soma o contador de segundos e escolhemos para executá-la a cada um segundo. No código do Arduino, dentro da função setup(), temos:

```
Timer1.initialize(1000000); //A cada um segundo
Timer1.attachInterrupt(soma_tempo); //Chamar a função soma_tempo
```

```
//Soma o tempo um segundo
void soma_tempo()
{
  segundo++;
  if (segundo == 60)
  {
    segundo = 0;
    minuto++;
  }
  if (minuto == 60)
  {
    minuto = 0;
    hora++;
  }
  if (hora == 24)
  {
    hora = 0;
  }
}
```

g) Diagnósticos

Os diagnósticos oferecidos pelo sistema corresponde à distância total percorrida. Avisamos, portanto, da troca de pneus e da sua calibração. Quando a distância total percorrida atinge o limite para a calibração dos pneus ou a sua troca, o sistema avisa pelo LCD e espera até que o usuário aperte o botão reset para sair da tela. Mesmo assim, o sistema continua funcionando normalmente em background.

A figura a seguir é um exemplo de aviso para calibrar os pneus da bicicleta. O sistema, mesmo mostrando essa informação na tela, continua executando suas tarefas em background.

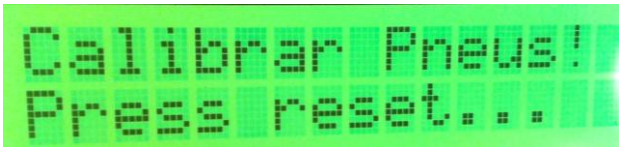


Figura 5: Exemplo de Diagnóstico por distância.

7. Resultados

Nesta seção mostraremos os resultados obtidos no projeto MyBike. O quadro a seguir mostra as prioridades do computador de bordo. O número 01 indica a maior prioridade e o 03 a menor. O sensor de efeito hall e o botão *mode* possuem as maiores prioridades no sistema, seguido pela interrupção de tempo e, por último, *reset*, diagnósticos e escrita no LCD.

Componente	Prioridade
Sensor de Efeito Hall	01
Botão Mode	01
Interrupção de Tempo	02
Botão Reset	03
Diagnósticos	03
Escrita no LCD	03

Tabela 2. Prioridade dos componentes no processador.

Para analisar o tempo gasto de cada função do sistema, utilizamos a função millis() disponível pelo Arduino. Chamamos esta função no inicio e no fim de um teste e depois subtraímos os seus valores. O resultado corresponde ao tempo gasto, em microssegundos, de cada função. A tabela a seguir mostra esses valores.

Função	Tempo Gasto (µs)
Sensor de Efeito Hall	04~08
Botão Mode	04~08
Interrupção de Tempo	04
Botão Reset	04~08
Diagnósticos	03~04
Escrita no LCD	04~10

Tabela 3. Tempo gasto em cada função.

Analisamos o projeto MyBike também quanto ao custo de produção. A tabela a seguir contém todos os componentes utilizados e o seu valor no mercado.

Componentes	Unidades	Preço Unitário	Preço Total
Tela LCD	1	R$ 15,00	R$ 15,00
Botões	3	R$ 1,00	R$ 1,00
Bateria	1	R$ 3,00	R$ 3,00
Arduino	1	R$ 20,00	R$ 20,00
Conectores	4	R$ 0,07	R$ 0,28
Sensor	1	R$ 6,00	R$ 6,00
Barra de Pinos	4	R$ 0,12	R$ 0,48
Placa	1	R$ 2,00	R$ 2,00
Papel Fotográfico	1	R$ 1,00	R$ 1,00
Total	-	-	R$ 48,76

Tabela 4: Custo do projeto MyBike

O conjunto do Arduino foi adquirido no site www.webtronico.com

Enquanto que os outros componentes foram adquiridos em www.soldafria.com.br

8. Conclusão

O projeto MyBike permite que o usuário tenha total controle sobre sua atividade na bicicleta, pois o sistema mostra ao usuário informações sobre velocidade média e instantânea, tempo da viagem e distância percorrida. Tudo isso com um baixo custo e em tempo real.

O processador em tempo real utilizado no sistema é capaz de atualizar as informações de maneira muito rápida e abaixo do *deadline* proposto, fazendo com que a percepção do usuário seja a de estar lendo as informações na hora em que pedala.

Além disso, o sistema de diagnóstico previne o usuário ao avisar sobre calibragem e troca de pneus da sua bicicleta. O MyBike foi desenvolvido para poder ser acoplado em qualquer bicicleta.

O projeto MyBike ainda tem muito espaço para desenvolvimento e melhorias. Adicionar sensores ao sistema é uma maneira de deixar o sistema mais robusto e diminui a probabilidade de erros. Além disso, podemos melhorar na área do consumo de bateria, ou seja, algumas tarefas como diminuir a luminosidade do LCD quando o sistema não está em uso ocasiona

uma grande economia.

Outras frentes também tem margem para deixar o sistema mais eficiente, como, por exemplo, o diagnóstico em caso de falha do sensor.

Referências Bibliográficas

[1] Laboratório de Garagem. Utilizando Interrupção e função Random() do Arduino. Em: http://labdegaragem.com/profiles/blogs/tutorial-utilizando-interrup-o-e-fun-o-random-do-arduino . Acesso em: 17 de Dezembro de 2013.

[2] Minicurso de Arduino. Em: http://www.inf.ufes.br/ ~erus/arquivos/ERUS_minicurso%20arduino.pdf . Acesso em: 22 de Dezembro de 2013.

[3] Laboratório de Garagem Como utilizar o sensor de efeito hall com arduino. Em http://labdegaragem.com/profiles/ blogs/tutorial-como-utilizar-o-sensor-de-efeito-hall-com-arduino. Acesso em 5 de Dezembro de 2013.

[4] LaboratóriodeGaragemhttp://labdegaragem.com/profiles/blogs/tutorial-lcd-com-arduino.Acessoem 5 de Dezembro de 2013.

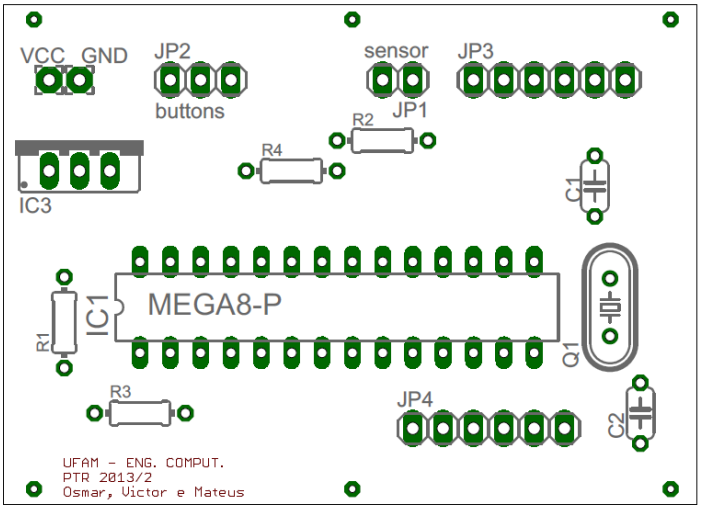
[5] Arduino. Em: http://arduino.cc/en/Reference/AttachInterrupt.Acessoem10deDezembrode2013.

[6]http://g1.globo.com/ciencia-e-saude/noticia/2012/04/quase-metade-da-populacao-esta-acima-do-peso-diz-saude.html

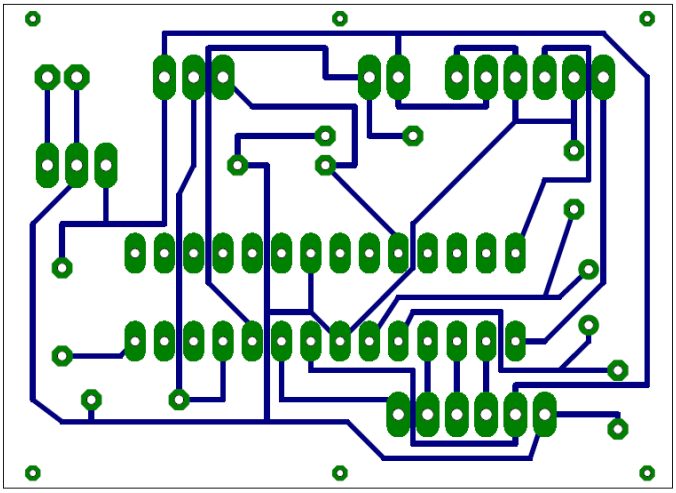
[7] http://boaforma.abril.com.br/fitness/todos-os-treinos-bicicleta/va-bike-emgarecer-tonificar-musculos-693679.shtml

Anexo

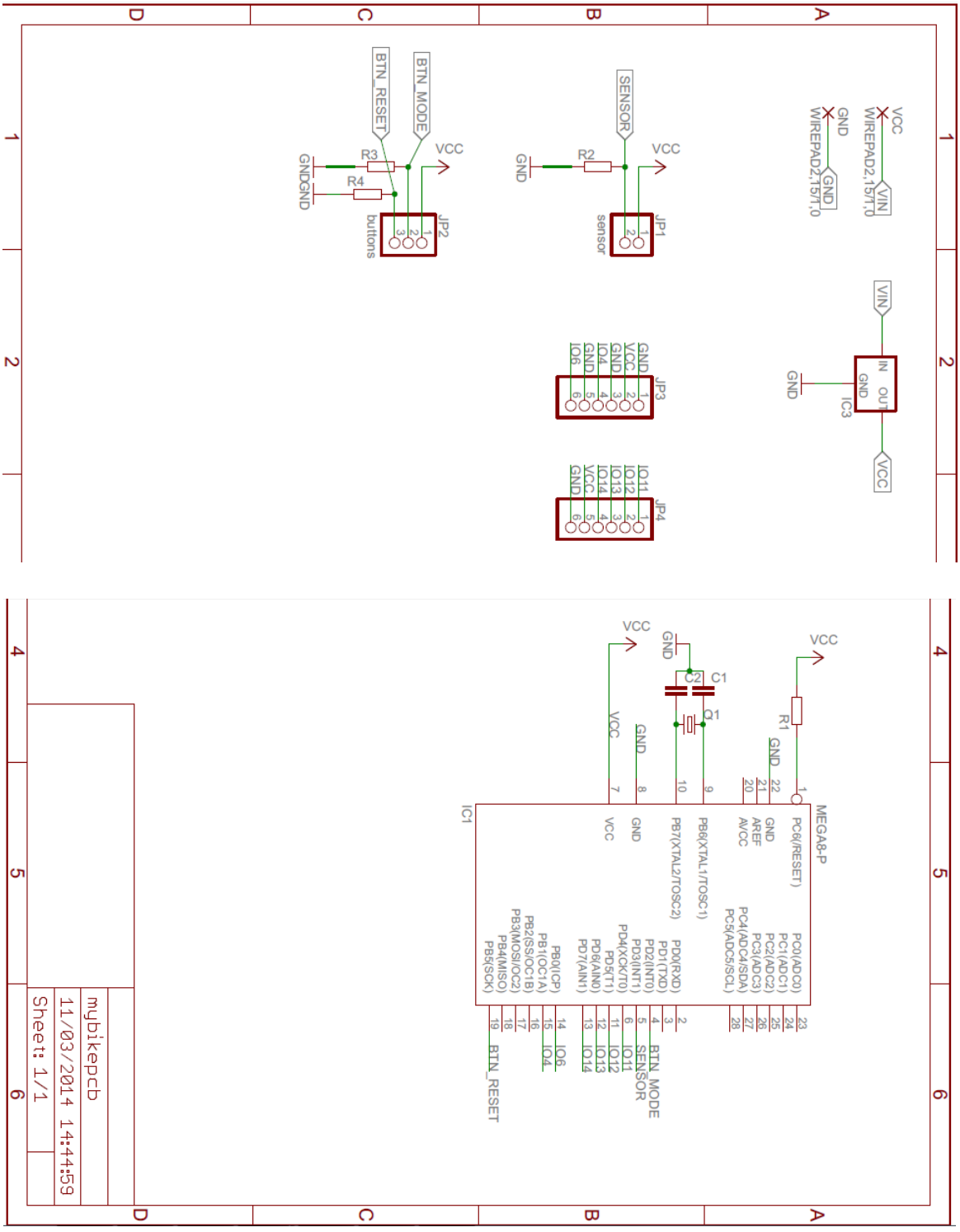
Esquemáticos



Anexo 1: Esquemático TopLayer



Anexo 2: Esquemático BottomLayer



Anexo 3: Esquemático

Anexo 4: Programa Arduino

```
#include <LiquidCrystal.h> //Inclui a biblioteca do LCD
#include "TimerOne.h" //Inclui a biblioteca de Interrupção de Tempo
//LCD
LiquidCrystal lcd(9, 8, 5, 4, 7, 6); //Configura os pinos do Arduino para se comunicar com o LCD
//Deixar configurável
float RAIO = 0.32;
float DIST_CALIBRAR = 50.0;
int kc = 1;
float DIST_TROCA = 200.0;
int kt = 1;

//Variaveis Globais
volatile int modo = 0; //Modo atual
/*
  MODOS DISPONIVEIS:
*/
int MODO_VIAGEM = 0; // Distância da Viagem atual
int MODO_VELOCIDADE = 1; // Velocidade da Viagem
int MODO_DISTANCIA_TOTAL = 2; // Distância Total
int MODO_TEMPO = 3; // Tempo de Viagem
int MODO_VELOCIDADE_INSTANTANEA = 4;
int QTD_MODOS = 4; // Quantidade de modos
int TIMEOUT_VELOCIDADE = 1000;

volatile float distanciaTotal = 0.0; //Em metros
volatile float distancia = 0.0; //Em metros
float velocidade = 0.0; //Em km/h
float distanciaPneu = 2*PI*RAIO;
int segundo = 0;
int minuto = 0;
int hora = 0;

String tempo;
String tela;
char dist[100];
char veloc[100];

//Tempo entre interrupções do botão mode
unsigned long button_time;
unsigned long last_button_time;
//Tempo entre interrupções do botão reset
unsigned long reset_button_time;
unsigned long last_reset_button_time;
//Tempo entre interrupções do sensor de efeito hall
unsigned long wheel_time;
unsigned long last_wheel_time;
unsigned long gap_wheel_time;

// Pinagem
// BOTAO MODE NO PINO 2
// SENSOR DE EFEITO HALL NO PINO 3
// Reset no pino 13
int resetPin = 13;

void setup()
{
  lcd.begin(16, 2); //Inicia o LCD com dimensões 16x2(Colunas x Linhas)
  attachInterrupt(0, mode_int, RISING); //Inicia Interrupção de mode com o método RISING
  attachInterrupt(1, wheel_int, RISING); //Inicia Interrupção do sensor de efeito hall com o método RISING
  Timer1.initialize(1000000); //A cada um segundo
  Timer1.attachInterrupt(soma_tempo); //Chamar a função soma_tempo()
  pinMode(resetPin,INPUT); //Iniciando pino do reset
```

```
  Serial.begin(9600);
}

//Interrupcao do botao mode
void mode_int()
{
  button_time = millis();
  if (button_time - last_button_time > 250)
  {
    if (modo == QTD_MODOS)
      modo = 0;
    else
      modo++;
    last_button_time = button_time;
  }
}


//Interrupcao do botao reset.

//Verifica se foi pressioando
void verificaPinoReset()
{
  reset_button_time = millis();
  if (reset_button_time - last_reset_button_time > 250)
  {
    if ( digitalRead(resetPin) )
    {
       reset_int();
    }
    last_reset_button_time = reset_button_time;
  }
}

//Funcao do pino reset
void reset_int()
{
  //Resetar tudo quando pressionado:
  distancia = 0.0;
  segundo = 0;
  minuto = 0;
  hora = 0;
}

//Interrupcao sensor de efeito hall
void wheel_int()
{
  wheel_time = millis();
  if (wheel_time - last_wheel_time > 20)
  {
    distancia += distanciaPneu;
    distanciaTotal += distanciaPneu;
    gap_wheel_time = wheel_time - last_wheel_time;
    last_wheel_time = wheel_time;
  }
}

//Soma o tempo um segundo
void soma_tempo()
{
  segundo++;
  if (segundo == 60)
  {
    segundo = 0;
    minuto++;
  }
  if (minuto == 60)
  {
```

```
    minuto = 0;
    hora++;
  }
  if (hora == 24)
  {
    hora = 0;
  }
}

//Escreve dados no LCD
void escreverLCD(char* modo, String valor, char* unidade)
{
  String nova_tela = modo + ';' + valor + ';' + unidade; //Verifica se a tela a ser
escrita é a mesma que já está escrita.
  if (nova_tela != tela) //Só escreve se a tela não for a mesma
  {
    //ZERAR A TELA ANTES DE ESCREVER
    lcd.clear();
    lcd.setCursor(0, 0);
    //Modo é uma string: Viagem, Velocidade, etc.
    lcd.print(modo);
    //Setando cursor para a linha inferior
    lcd.setCursor(0, 1);
    //Escrever o valor
    lcd.print(valor);
    //Logo depois escrever a unidade
    lcd.print(unidade);
    //Salva a tela para comparações futuras
    tela = modo + ';' + valor + ';' + unidade;
  }
}

int pass2sec(int seg, int minut, int hora)
{
  return (seg + (60 * minut) + (3600 * hora));
}

float metro2km(float metro)
{
  return metro/1000.0;
}

float seg2hora(int seg)
{
  return seg/3600.0;
}

void verificaModo(int modo)
{
  if (modo == MODO_VIAGEM)
  {
    if (distancia > 1000.0)
    {
      float km = metro2km(distancia);
      dtostrf(km,5,2,dist);
      escreverLCD("Viagem", dist, "km");
    } else {
      dtostrf(distancia,5,2,dist);
      escreverLCD("Viagem", dist, "m");
    }
  }
  else if (modo == MODO_VELOCIDADE)
  {
    //Metro por segundo
    //int t = pass2sec(segundo, minuto, hora);
    //velocidade = distancia / ((float) t);
    //dtostrf(velocidade,5,2,veloc);
    //escreverLCD("Velocidade", veloc, "m/s");
```

```
    //Km por Hora
    float h = seg2hora(pass2sec(segundo,minuto,hora));
    float km = metro2km(distancia);
    velocidade = km/h;
    dtostrf(velocidade,5,2,veloc);
    escreverLCD("Velocidade", veloc, "km/h");
  }
  else if (modo == MODO_DISTANCIA_TOTAL)
  {
    if (distanciaTotal > 1000.0)
    {
      float km = metro2km(distanciaTotal);
      dtostrf(km,5,2,dist);
      escreverLCD("Distancia Total", dist, "km");
    } else {
      dtostrf(distanciaTotal,5,2,dist);
      escreverLCD("Distancia Total", dist, "m");
    }
  }
  else if (modo == MODO_TEMPO)
  {
    tempo = String(hora) + 'h' + String(minuto) + 'm' + String(segundo) + 's';
    escreverLCD("Tempo", tempo, "");
  }
  else if (modo == MODO_VELOCIDADE_INSTANTANEA)
  {
    //GAP do TEMPO
    unsigned long time_tmp = millis();
    if (time_tmp - last_wheel_time <= TIMEOUT_VELOCIDADE)
    {
      velocidade = distanciaPneu / (float(gap_wheel_time)/3600.0);
      dtostrf(velocidade,5,2,veloc);
      escreverLCD("Velocidade Inst.", veloc, "km/h");
    } else {
      escreverLCD("Velocidade Inst.", "0.0", "km/h");
    }
  }
}

void verificaDiagnosticos()
{
  if (kc*DIST_CALIBRAR < distanciaTotal)
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Calibrar Pneus!");
    lcd.setCursor(0,1);
    lcd.print("Press reset...");
    while(true)
    {
      reset_button_time = millis();
      if     (reset_button_time     -     last_reset_button_time     >     250     &&
digitalRead(resetPin))
      {
        last_reset_button_time = reset_button_time;
        break;
      }
      delay(100);
    }
    lcd.clear();
    kc++;
  }
  if (kt*DIST_TROCA < distanciaTotal)
  {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Trocar Pneus!");
    lcd.setCursor(0,1);
```

```
    lcd.print("Press reset...");
    while(true)
    {
      reset_button_time = millis();
      if     (reset_button_time     -     last_reset_button_time     >     250     &&
digitalRead(resetPin))
      {
        last_reset_button_time = reset_button_time;
        break;
      }
      delay(100);
    }
    lcd.clear();
    kt++;
  }
}

void loop()
{
  verificaModo(modo);
  verificaPinoReset();
  verificaDiagnosticos();
}
```