Telemetria veicular e análise de dados aplicados a veículos Baja

Proposta do Projeto

Júlio Ohfugi Yamaguti
FGA
UNB – Universidade de Brasília
Gama,Brasil
julio.ohfugi21@gmail.com

Guilherme Felix De Andrade
FGA
UNB – Universidade de Brasília
Gama,Brasil
guilhermefelixandrade@gmail.com

Resumo- **Este projeto tem como finalidade a criação de uma plataforma embarcada para o monitoramento em tempo real do carro baja da equipe de competição Unbaja da universidade de Brasília, fazendo o mapeamento da pista e aquisição de dados para o melhor dimensionamento do veículo.**

I. INTRODUÇÃO

Os veículos Baja são protótipos de um veículo off road com a função de uso fora de estrada, com quatro ou mais rodas e motor padrão, estabelecido pelo regulamento da competição, de 10 HP, que devem ser capazes de transportar pilotos com até 1,90 m de altura, e com peso de até 113,4 kg. Os sistemas de eletrônica, suspensão, transmissão e freios, assim como o próprio chassi, são projetados e construídos pelas equipes [1].

O projeto Baja SAE foi criado na Universidade da Carolina do Sul, Estados Unidos, sob a direção do Dr. John F. Sthevens, e a primeira competição ocorreu em 1976. O ano de 1991 marcou o início das atividades da SAE BRASIL, e em 1994 lançou o Projeto Baja SAE BRASIL [2].

Em 1995, realizou-se a primeira competição nacional, na pista Guido Caloi, bairro do Ibirapuera, cidade de São Paulo. No ano seguinte a competição foi transferida para o Autódromo de Interlagos, onde permaneceu até o ano de 2002. A partir de 2003 a competição passou a ser realizada em Piracicaba, interior de São Paulo, no ECPA – Esporte Clube Piracicabano de Automobilismo [2].

Desde 1997 a SAE BRASIL também apoia a realização de eventos regionais do Baja SAE BRASIL, por meio de suas Competições Regionais. Desde então dezenas de eventos foram realizados em vários estados do país como Rio Grande do Sul, São Paulo, Minas Gerais e Bahia [2].

O programa Baja SAE BRASIL é um desafio proposto aos estudantes de Engenharia que oferece a aplicação na prática dos conhecimentos adquiridos em sala de aula, com o objetivo de incrementar sua preparação para o mercado de trabalho. Ao participar do programa Baja SAE, o aluno se envolve com um caso real de desenvolvimento de um veículo off road, desde sua concepção, projeto detalhado, construção e testes. Além disso, as equipes vencedoras da etapa nacional são convidadas a participar da competição internacional, nos Estados Unidos [2].

Figura 1 - Prova de conforto, Regional Nordeste, 2017.

Figura 2 - Prova Enduro de resistência, Regional Nordeste, 2017.

Além da aplicação prática das matérias passadas em sala de aula, a competição valoriza muito a parte teórica realizada e as simulações, que são necessárias para justificar todo o dimensionamento do projeto [3].

As avaliações e notas são dadas da seguinte maneira:

Avaliação de Projeto - 320 pontos	
Relatório de Projeto	120
Avaliação de Projeto Dinâmico	20
Apresentação de Projeto e Finais de Apresentação de Projeto	180
Eventos dinâmicos - 280 pontos	
Aceleração	45
Velocidade Máxima	45
Tração	45
Lama	45
Suspensão	70
Manobrabilidade	30
Enduro de Resistência - 400 pontos	
Pontuação total	**1000**

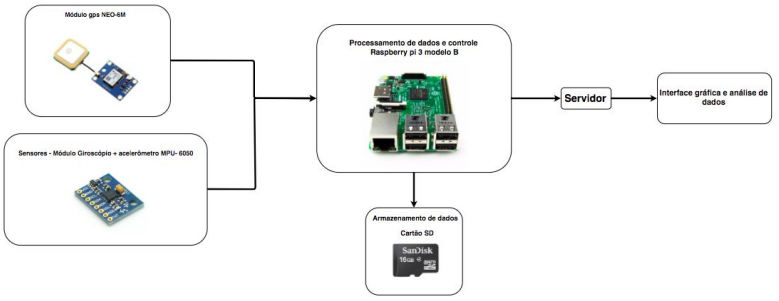
Figura 3 - Pontuações da competição[3].

II. Objetivos

Este projeto almeja a modelagem de um sistema embarcado que possa ser aplicado em veículos de competição, tendo como foco o veículo baja da equipe de competição Unbaja da Universidade de Brasília campus Gama, com o intuito de informar para os integrantes os dados qualitativos e quantitativos sobre o veículo e seu posicionamento em tempo real durante o período que antecede a corrida, para que seja possível a realização de uma análise e dimensionamento adequado e resistente de cada subsistema do projeto.

Será realizado o mapeamento do carro na pista a partir dos dados adquiridos com o módulo gps e módulo giroscópio e acelerômetro (latitude, longitude, altitude, velocidade, inclinação do chassi), ocorrerá o estabelecimento de uma comunicação com um computador externo para a exibição e avaliação dos dados em tempo real, a partir da criação de zonas de calor em um mapa. Com isso será possível a obtenção dos perfis de terreno que possibilitará ter um conhecimento em relação ao tipo de comportamento que o veículo realizará sobre diferentes trechos de pistas. Além disso, os dados serão essenciais para o cálculo de esforços que serão exercidos nos componentes do carro.

Figura 4 – Diagrama de funcionamento do



sistema

III. Justificativa e benefícios

Para realização de um projeto e construção de um veículo é necessário a análise e estudo de inúmeros fatores, como por exemplo, as características de terreno em que ele será utilizado

e o tipo e a quantidade de esforços que cada parte do veículo sofrerá. Essas informações são essenciais para a realização de um dimensionamento de cada subsistema do carro, de forma coerente e apta para um devido funcionamento do veículo [5].

Temos como exemplo um sistema de suspensão veicular, que possui como objetivo absorver as irregularidades do solo e dissipá-las sem afetar o desempenho do veículo e o conforto dos ocupantes. Hoje em dia diversas bibliografias apresentam como efetuar analiticamente o projeto de uma suspensão tendo como meta as características citadas. Mas é necessário, com o projeto construído, validar tais características. Para isso os projetistas precisam de ferramentas que durante o deslocamento do veículo forneçam informações precisas sobre o trajeto percorrido pelo veículo, e as respostas da carroceria a tais deslocamentos. Informações como velocidade em determinado trecho de pista, e deslocamento angular lateral da carroceria, são dados de entrada para modelos matemáticos e softwares de análise em multi corpos que são capazes, em ambiente virtual, de simular e otimizar o comportamento de um veículo antes de seu desenvolvimento final. Com isso é possível economizar recursos e gastos com prototipagem [6].

Empregou-se o uso do Raspberry Pi para realizar o projeto que se diferencia de microcontroladores na questão de permitir múltiplas tarefas do software nele embarcado e o monitoramento em tempo real. Com isso é possível gerenciar os sistemas que serão empregados para cumprir as tarefas a serem resolvidas no projeto.

Para a captação dos dados de latitude, longitude e altitude, necessitou-se a utilização do módulo gps NEO-6M, pois este atende as necessidades do projeto com um custo baixo no mercado, suas dimensões são compactas, podendo ser colocada no painel do Baja e comunica-se com a raspberry pi 3 através da comunicação UART, fazendo a interpretação dos dados no padrão NMEA, conexão de internet e envio das informações para o servidor que com o banco de dados de posicionamento, é capaz de disponibilizar remotamente as informações coletadas, gerando um ambiente de fácil entendimento para o usuário.

V. Materiais utilizados

Processamento de dados e controle Raspberry pi 3 modelo B

Armazenamento de dados

Cartão SD

Módulo GPS NEO-6M

Sistema

Linux e Raspbian

VI. Descrição de Hardware

A. Hardware para teste do sistema

A raspberry é conectada diretamente ao módulo gps NEO-6M, com os grounds ligados, o pino vcc do gps ligado à saída de 3.3v, as comunicações Rx ligado ao Tx e Tx ligado ao Rx da raspberry, os pinos do MPU Vcc ligado à saída de 3.3v, scl e sda ligados aos GPIOS 2 e 3 respectivamente. O circuito para teste foi montado em uma protoboard de acordo com a figura 5, esquematizado utilizando software Fritzing.

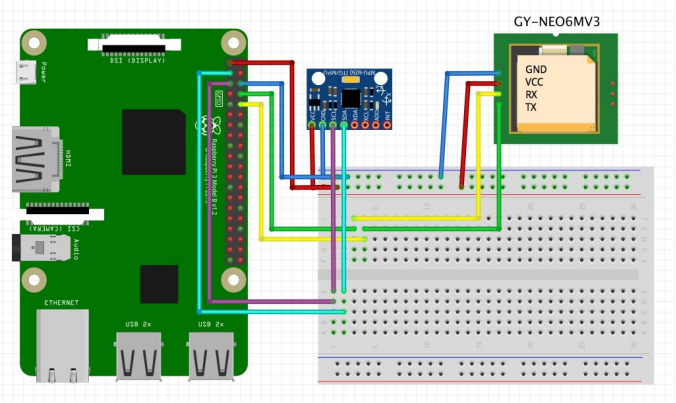


Figura 5 - Circuito para teste.

B. Hardware do protótipo final

Para o hardware do protótipo final, projetou-se uma placa de circuito impresso, utilizando o software Proteus, com o intuito de facilitar a montagem, não necessitando do uso de jumpers, gerar exclusividade e tornar o projeto mais profissional. O o circuito e o layout da pcb se estabeleceram de acordo com as figuras 6 e 7.

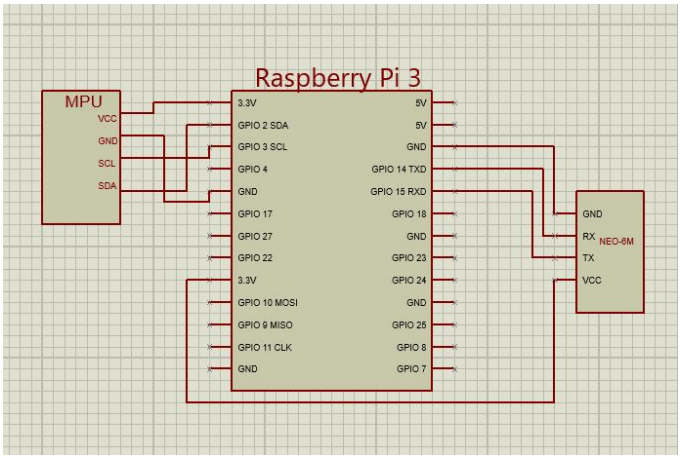


Figura 6 - Circuito do hardware do sistema

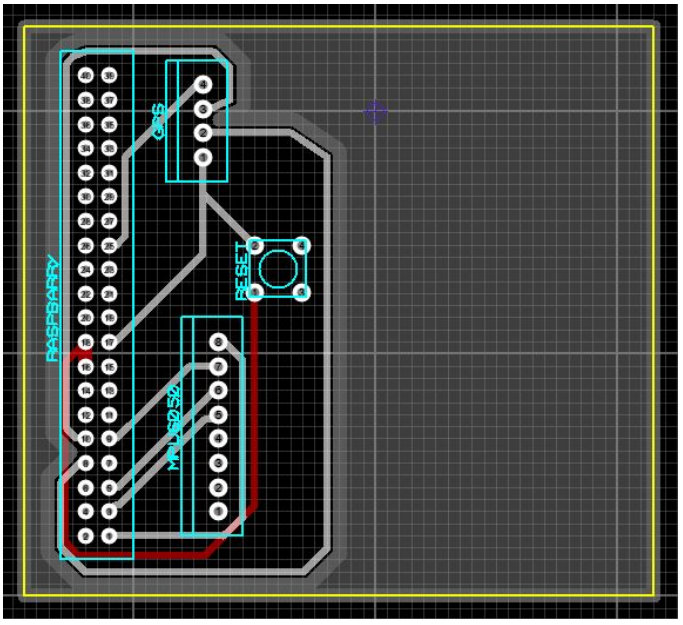


Figura 7 - Layout da placa de circuito impresso

Fabricou-se a placa de circuito impresso pelo método fotosensível de acordo com a figura 8 e 9.

Figura 8 - PCB final (top)

Figura 9 - PCB final (bottom)

.

VII. Descrição de Software

A. Padrão NMEA

O módulo gps NEO-6M comunica-se através de um protocolo definido pela National Marine Eletronics Association(NMEA) [4].

Os dados provenientes deste dispositivo são fornecidos em pacotes chamados de sentenças NMEA. Existem vários tipos diferentes de sentenças com finalidades distintas, tais como latitude, longitude, altitude, velocidade, dentre outras [4].

Todas as sentenças de NMEA consistem em até 82 caracteres em ASCII. Cada sentença começa com o símbolo $ e termina com um carriage return e o consequente começo de nova linha. Todos os campos de dados são delimitados por vírgulas (,) e têm comprimento variável. Os campos nulos também são delimitados por vírgulas (,) mas não contêm nenhuma informação. O primeiro campo de dados é um campo de endereço, ao passo que o último campo de dados é um controle de soma (checksum) [4].

B. Configuração UART

O módulo gps e a raspberry pi 3 comunicam-se a partir da comunicação UART. O Raspian aplica a porta serial assíncrona (UART) para acesso remoto ao terminal e para utilizá-la é necessário desabilitar este acesso remotos.

C. Servidor

Para a criação do servidor utilizou-se a solução Apache, que é um servidor HTTP open-source [11]. Na modificação da interface do servidor, é necessário modificar o arquivo index.html, que é responsável por gerenciar tudo que aparecerá ao acessar o servidor por um navegador web, onde digitamos nosso código html.

Para acessar o servidor é necessário obter o endereço do mesmo, realizando o comando $ifconfig e obtendo o endereço do inet addr, no caso do nosso servidor, é 192.168.0.51. Para visualizar o conteúdo do servidor basta digitar o endereço em um navegador web, como por exemplo, o google chrome.

Instalou-se o PHP7 para que fosse possível gerar o mapa de calor provenientes dos dados coletados pelo gps.

D. GPS

Para a obtenção de dados e codificação do sistema de gps, utilizou-se do recurso de threads, com o intuito de realizar 2 processos em paralelo, de acordo com a figura 11.

```c
int main(int argc, char** argv){

    pthread_t t0;
    pthread_t t1;

    signal(SIGINT, fecharPrograma);

    puts("Iniciando gps");
    pthread_create(&t0,NULL,&gps,NULL);
    puts("Botão de reset pronto");
    pthread_create(&t1,NULL,&rst,NULL);
    while(1){}

return 0;
}
```

Figura 11 - Função main do código do gps.

O primeiro processo realiza a obtenção de dados do gps e os salva em um arquivo csv. Utiliza-se da biblioteca gps.h que é responsável por realizar as configurações da comunicação uart, utilizada pelo gps, e por interpretar a sentença NMEA gerada pelo pelo mesmo, para a obtenção dos dados de latitude, longitude, altitude e velocidade, de acordo com a figura 12 [13].

```c
void* gps(void *arg){

    gps_init( );
    loc_t data;


    while(1){

        //fp = fopen("/home/pi/Desktop/data.csv", "a+");
        fp = fopen("/var/www/html/data.csv", "a+");

        if(inicioArquivo ==0){
        fprintf( fp,"\"lat\",\"lon\",\"alt\",\"vel\"\n");
        inicioArquivo=1;
        puts("Iniciando arquivo");
        }

        gps_location(&data);
        fprintf(fp, "%lf,%lf,%f,%f\n", data.latitude, data.longitude, data.altitude, data.speed);
        fclose(fp);
        usleep(1000000);

    }
    return NULL;
}
```

Figura 12 - Função do gps.

Já o segundo processo, identifica o acionamento de um botão com o intuito de apagar o conteúdo do arquivo csv onde armazena-se os dados do gps, para a realização de uma nova obtenção de dados.

Para a identificação do acionamento do botão, realizou-se o polling do mesmo em um pino GPIO.

Primeiro configurou-se a GPIO 24 para identificar a entrada de uma borda de descida, e após a identificação apaga-se o arquivo csv, de acordo com a figura 13, que será criado novamente ao realizar o fopen na linha 58.

```c
void* rst(void *arg){

    int btn_press;
    system("echo 24     > /sys/class/gpio/export");
    system("echo falling > /sys/class/gpio/gpio24/edge");
    system("echo in     > /sys/class/gpio/gpio24/direction");
    pfd.fd = open("/sys/class/gpio/gpio24/value", O_RDONLY);
    if(pfd.fd < 0)
    {
        puts("Erro abrindo /sys/class/gpio/gpio24/value");
        puts("Execute este programa como root.");
        return 0;
    }
    pfd.events = POLLPRI | POLLERR;
    pfd.revents = 0;

    while(1){
    poll_btn(&pfd);
    remove("data.csv");
    puts("Apagando arquivo");
    inicioArquivo = 0;
    }

    return NULL;

}
```

Figura 13 - Função de reset do gps.

Além disso, para o melhor funcionamento do botão e minimizar trepidações realiza-se o dbounce do mesmo, pela função criada, poll_btn. Essa função obtêm 100 valores e a mesma é finalizada quando pelo menos 50% dos valores são diferentes do valor inicial, de acordo com a figura 14.

```c
void poll_btn(struct pollfd *pfd)
{
    char buffer, vals[N] = {0};
    int pos = 0, test=1, i, s;
    lseek(pfd->fd, 0, SEEK_SET);
    read(pfd->fd, &buffer, 1);
    for(i=0;i<N;i++) vals[i] = buffer-'0';
    poll(pfd, 1, -1);
    while(test)
    {
        lseek(pfd->fd, 0, SEEK_SET);
        read(pfd->fd, vals+pos, 1);
        vals[pos] -= '0';
        pos = (pos+1)%N;
        for(s=i=0; i<N; i++)
            s += vals[i];
        if(buffer=='0')
            test = s<Nlim;
        else
            test = s>Nlim;
    }
}
```

Figura 14 - Função de dbounce.

Por final, para finalizar o programa, utilizou-se do recurso de sinais, de acordo com a linha 119 na figura 11. Então quando pressiona-se cntrl c, a função fecharPrograma é chamada e realiza o fechamento do arquivo e o unexport do pino GPIO, de acordo com a figura 15.

```
49  void fecharPrograma( ){
50
51      printf("Fechando programa...\n");
52      fclose(fp);
53      system("echo 24 > /sys/class/gpio/unexport");
54      close(pfd.fd);
55      sleep(1);
56      exit(0);
57  }
```

Figura 15 - Função para finalização do programa.

E. MPU6050

Para codificação do sistema de obtenção de dados do MPU6050, utilizou-se dos mesmos recursos utilizados na obtenção de dados do GPS, ou seja, threads e pinos GPIO, de acordo com a figura 16.

```
161 int main(int argc, char** argv)
162 {
163
164     pthread_t t2;
165     pthread_t t3;
166
167     signal(SIGINT, fecharPrograma);
168
169     puts("Iniciando mpu");
170     pthread_create(&t2,NULL,&mpu,NULL);
171     puts("Botão de reset pronto");
172     pthread_create(&t3,NULL,&rst,NULL);
173     while(1){}
174
175 return 0;
176 }
```

Figura 16 - Função main do código do MPU6050.

O primeiro processo realiza a obtenção de dados do giroscópio e acelerômetro, presentes no módulo MPU6050, e os salva em um arquivo csv, de acordo com a figura 17.

Utiliza-se da biblioteca wiringPiI2C.h, para a facilitar utilização da comunicação I2C no Raspbarry. De acordo com a figura 17, inicia-se pela função wirinfPiI2CSetup, que inicializa o sistema I2C com o dado dispositivo. Logo após, entra-se em um loop que realiza a gravação dos dados no arquivo “mpu.csv”. Os dados são obtidos a partir das funções criadas, de acordo com a figura 18. A função read_word_2c , linha 55, realizam a leitura de 8 bits do registrador indicado, de acordo com as linhas 103, 104 e 105. Já as funções get, são responsáveis por calcular a angulação em graus, dos eixos x e y do acelerômetro [14].

```
87  void* mpu(void *arg){
88
89      fd = wiringPiI2CSetup (0x68);
90      wiringPiI2CWriteReg8 (fd,0x6B,0x00);//disable sleep mode
91      printf("set 0x6B=%X\n",wiringPiI2CReadReg8 (fd,0x6B));
92
93      while(1) {
94
95      fp = fopen("/var/www/html/mpu.csv", "a+");
96
97      if(inicioArquivo == 0){
98      fprintf( fp,"\"Giroscópio\",\"Acelerômetro angulo\"\n");
99      inicioArquivo = 1;
100     puts("Iniciando arquivo");
101     }
102
103     acclX = read_word_2c(0x3B);
104     acclY = read_word_2c(0x3D);
105     acclZ = read_word_2c(0x3F);
106
107     acclX_scaled = acclX / 16384.0;
108     acclY_scaled = acclY / 16384.0;
109     acclZ_scaled = acclZ / 16384.0;
110
111
112     fprintf(fp, "X=%f,X=%f\n", acclX_scaled,get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
113     fprintf(fp, "Y=%f,Y=%f\n", acclY_scaled,get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
114     fprintf(fp, "Z=%f, " "\n", acclZ_scaled);
115     usleep(1000000);
116     fclose(fp);
117     }
```

Figura 17 - Função do módulo mpu.

```
55  int read_word_2c(int addr)
56  {
57  int val;
58  val = wiringPiI2CReadReg8(fd, addr);
59  val = val << 8;
60  val += wiringPiI2CReadReg8(fd, addr+1);
61  if (val >= 0x8000)
62  val = -(65536 - val);
63
64  return val;
65  }
66
67  double dist(double a, double b)
68  {
69  return sqrt((a*a) + (b*b));
70  }
71
72  double get_y_rotation(double x, double y, double z)
73  {
74  double radians;
75  radians = atan2(x, dist(y, z));
76  return -(radians * (180.0 / M_PI));
77  }
78
79  double get_x_rotation(double x, double y, double z)
80  {
81  double radians;
82  radians = atan2(y, dist(x, z));
83  return (radians * (180.0 / M_PI));
84  }
```

Figura 18 - Funções definidas, e utilizadas na função mpu.

Já o segundo processo, identifica o acionamento de um botão com o intuito de resetar os dados gravados, para iniciar-se novas leituras. Para isso, realizou-se os mesmos processos realizados no código do gps, realizando apenas a mudança da porta GPIO utilizada, para a GPIO 23, de acordo com a figura 19. Além disso, mantêm-se a função para dbounce do botão, de acordo com a figura 14.

```
121  void* rst(void *arg){
122
123      int btn_press;
124      system("echo 23     > /sys/class/gpio/export");
125      system("echo falling > /sys/class/gpio/gpio23/edge");
126      system("echo in     > /sys/class/gpio/gpio23/direction");
127      pfd.fd = open("/sys/class/gpio/gpio23/value", O_RDONLY);
128      if(pfd.fd < 0)
129      {
130          puts("Erro abrindo /sys/class/gpio/gpio23/value");
131          puts("Execute este programa como root.");
132          return 0;
133      }
134      pfd.events = POLLPRI | POLLERR;
135      pfd.revents = 0;
136
137      while(1){
138      poll_btn(&pfd);
139      remove("mpu.csv");
140      puts("Apagando arquivo");
141      inicioArquivo = 0;
142      }
143
144      return NULL;
145
146  }
```

Figura 19 - Função de reset do mpu.

F. Interface gráfica para teste

Para a construção da interface gráfica e realização dos primeiros testes, utilizou-se das linguagens de programação HTML, em conjunto com CSS e JavaScrip. Além disso, utilizou-se do Google Maps JavaScript AP e da biblioteca google.maps.visualization, para a geração do mapa e de uma camada de mapa de calor,respectivamente, utilizando-se de suas classes, construtores e métodos [10].

```
1   <!DOCTYPE html>
2
3   <html>
4
5     <head>
6
7       <title>Mapa UNBAJA</title>
8       <meta name="viewport" content="initial-scale=1.0">
9       <meta charset="utf-8">
10
11      <style>
12
13        #map {
14          height: 100%;
15        }
16
17        html, body {
18          height: 100%;
19          margin: 0;
20          padding: 0;
21        }
22      </style>
23
24    </head>
```

Figura 20 - Código HTML, primeira parte.

A primeira parte do código se dispõe de acordo com a figura 20. Na primeira linha do código é declarado que o documento a ser escrito é do tipo HTML5. Logo após, na linha 3, é declarado o inicio da tag html, e tudo que estiver descrito abaixo, até o fechamento da tag, será considerado um documento html.

Já a descrição da linha 5 a linha 22, está representado a parte do cabeçalho, ou seja, é responsável pelas configurações comportamentais do site. As primeiras informações do programa são o título da página(linha 7), as instruções de controle das dimensões e escala da página(linha 8) e a definição do uso dos conjuntos de caracteres latinos(linha 9). Além disso, da linha 11 a 22, é realizado a definição do estilo de informação para a página HTML, em que é configurado para que a página ocupe todo o navegador [8].

```
27    <body>
28
29      <div id="map"></div>
30
31      <script>
32        var map, heatmap;
33        function initMap() {
34          map = new google.maps.Map(document.getElementById('map'), {
35            center: {lat: 37.775, lng: -122.434},
36            zoom: 13
37          });
38
39
40          heatmap = new google.maps.visualization.HeatmapLayer({
41            data: getPoints(),
42            map: map
43          });
44        }
```

Figura 21 - Código HTML, segunda parte.

A segunda parte do código é apresentado de acordo com a figura 21. Na linha 27 do código, é declarado o inicio da tag body, em que tudo que vai aparecer, visualmente, tem q estar na área body. Logo depois, na linha 29, é criado um elemento div, que é nomeado e obtém uma referência para o elemento map, no modelo de objetos do documento (DOM) do navegador. Isso é feito para reservar um lugar para mapa, para que o mesmo seja exibido em uma página. Além disso, utiliza-se do CSS para definir a altura do div do mapa para “100%”[7].

A partir da linha 31 até a linha 550, encontra-se a tag script, em que tudo que tiver entre <script> e </script> é considerado um código em linguagem JavaScript, responsável por manipular objetos que estejam dentro do navegador [8].

Logo em seguida, é declarado uma variável “map” e “heatmap”, linha 32, e declara-se uma função “initMap”, linha 33. Dentro dessa função é criado um objeto “map”, pertencente a classe “google.maps.Map”, e logo em seguida, é chamado oconstrutor “Map(document.getElementById('map'), {center: {lat: 37.775, lng: -122.434}, zoom: 13 });”. Esse construtor é responsável por criar um novo mapa dentro do dado HMTL, com o zoom escolhido, e a latitude e longitude, utilizadas, para determinar o local, que para a fase de teste, utilizou-se um local em São Francisco[7].

Ainda dentro da função, é criado o objeto “heatmap”, linha 39, e chama-se o construtor “heatmap = new google.maps.visualization.HeatmapLayer({
data: getPoints(),
map: map });”

Esse construtor é responsável por criar o mapa de calor, a partir dos dados de latitude e longitude retornados pela função “getPoints()”, que para teste, foram determinados valores de exemplo, fornecidos pelo Google API [9].

```
552    <script async defer
553        src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDSbHv2gnqaJT8xgBjaGviCNytVz8A99VA&libraries=visualization&callback=initMap">
554    </script>
555
556    <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDSbHv2gnqaJT8xgBjaGviCNytVz8A99VA&callback=initMap"
557    async defer></script>
558
559  </body>
560
561
562  </html>
```

Figura 22 - Código HTML, terceira parte.

A terceira parte do código é apresentado de acordo com a figura 22. Da linha 552 a linha 557 são carregados a Google Maps JavaScript API e a biblioteca de Visualization, para que possamos utilizar de seus conteúdos.

G. Interface gráfica final

Para a realização da interface final, seguiu-se como inspiração o template fornecido pela html5up.net. Realizou-se diversas modificações nos códigos html e nos códigos css, para a obtenção do formato e interface desejado para a página do servidor. Após a definição da interface, realizou-se, mais uma vez, modificações nos códigos hmtl, realizando a implementação das codificações em java script, php, alguns já utilizados no código da interface de teste, com o intuito de gerar o mapa e a tabela de dados referentes ao gps e ao módulo MPU, respectivamente.

As figuras 24, 25, 26 e 27 representam a interface da página inicial, página do mapa, página dos dados do GPS e páginas dos dados do MPU, respectivamente.

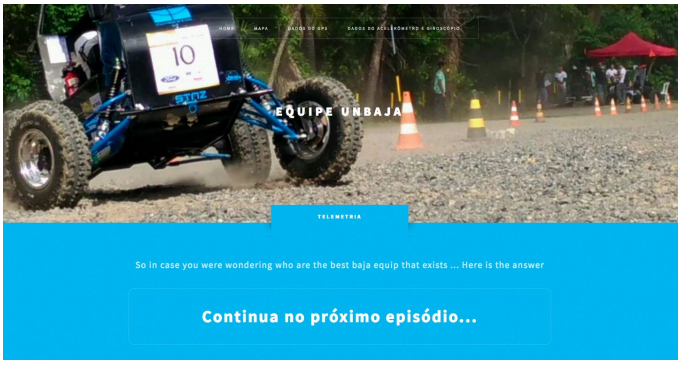


Figura 24 - Home.

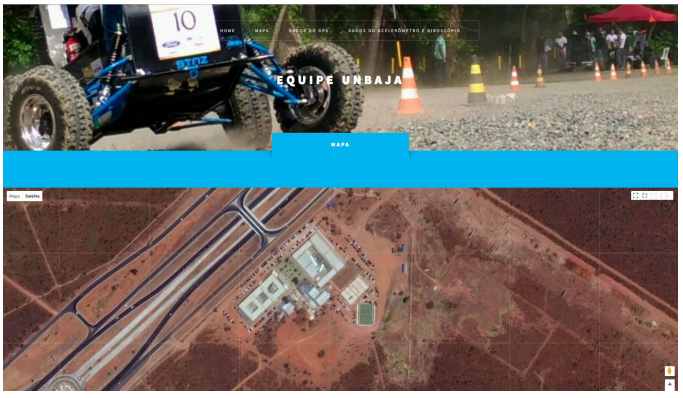
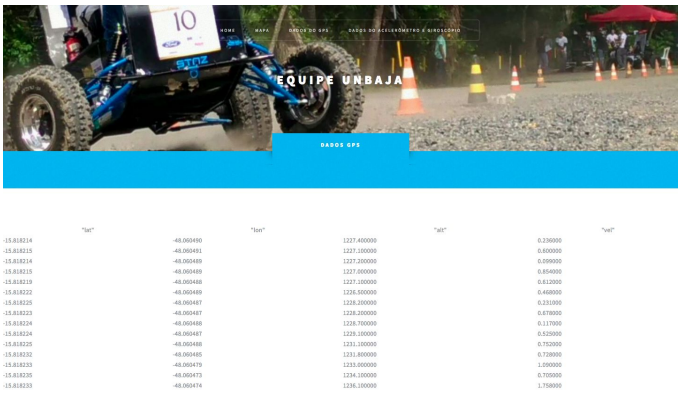


Figura 25 - Mapa.



"lat"	"lon"	"alt"	"vel"
-15.818214	-48.060490	1227.400000	0.236000
-15.818215	-48.060491	1227.100000	0.600000
-15.818214	-48.060489	1227.200000	0.099000
-15.818215	-48.060489	1227.000000	0.854000
-15.818219	-48.060488	1227.100000	0.612000
-15.818222	-48.060489	1226.500000	0.468000
-15.818225	-48.060487	1228.200000	0.231000
-15.818223	-48.060487	1228.200000	0.678000
-15.818224	-48.060488	1228.700000	0.117000
-15.818224	-48.060487	1228.100000	0.525000
-15.818225	-48.060488	1231.100000	0.752000
-15.818232	-48.060485	1231.800000	0.728000
-15.818233	-48.060479	1233.000000	1.090000
-15.818235	-48.060473	1234.100000	0.705000
-15.818233	-48.060474	1236.100000	1.758000

Figura 26 - Gps.

"Giroscópio"	"Acelerômetro angulo"
X=-0.859131	X=-7.700978
Y=-0.135986	Y=57.843910
Z=-0.522705	
X=-0.829834	X=-4.887730
Y=-0.085449	Y=55.837497
Z=-0.556641	
X=-0.809326	X=-5.680245
Y=-0.098633	Y=54.306290
Z=-0.572998	
X=-0.797363	X=-4.898782
Y=-0.085938	Y=52.404420
Z=-0.607910	
X=-0.759766	X=-4.281243
Y=-0.097412	Y=55.608976
Z=1.056396	

Figura 27 - Mpu.

VIII. Descrição de estrutura

Para o armazenamento de todos os componentes do projeto, projetou-se e construiu-se uma caixa. Para o projeto do cad, utilizou-se do software Sketchup, de acordo com a figura 28 e 29, que foi projetada para o guardar a raspberry e uma bateria para sua alimentação.

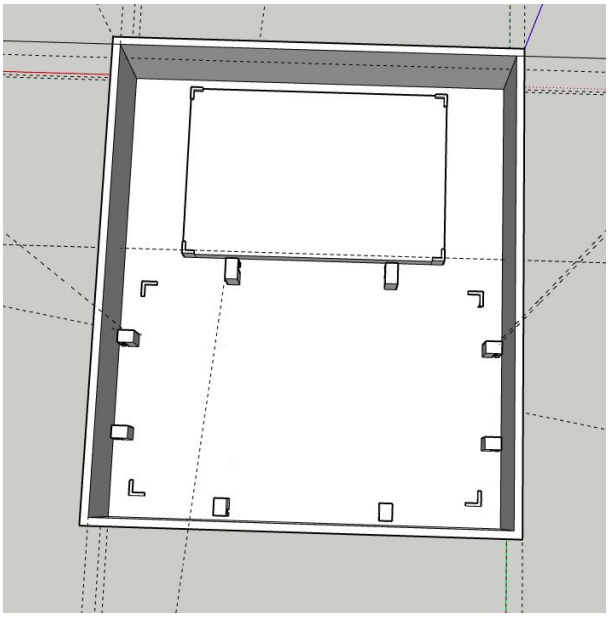


Figura 28 - Caixa, software Sketchup .

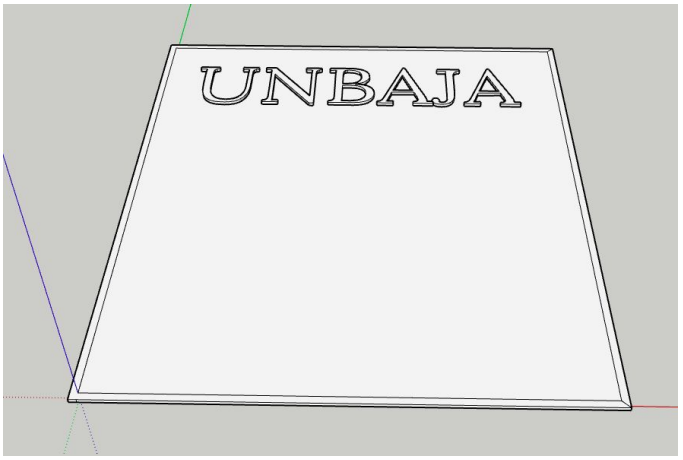


Figura 29 - Tampa, software Sketchup.

Após a finalização do projeto de cad, utilizou-se do software Cura, para, a partir do arquivo .stl, obter-se o código g, para que possa ser interpretado pela impressora 3D, de acordo com as figuras 30 e 31.

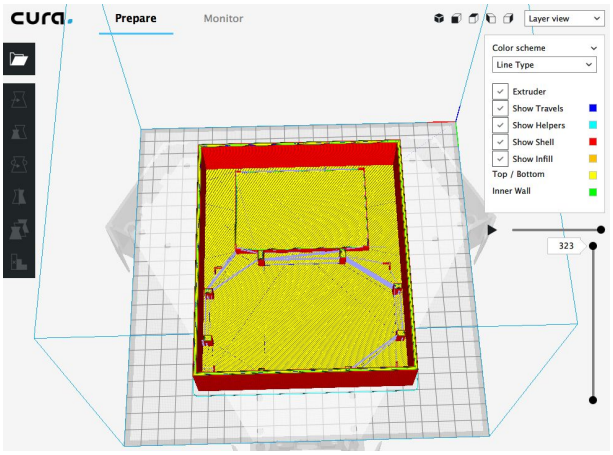


Figura 30 - Caixa, software Cura.

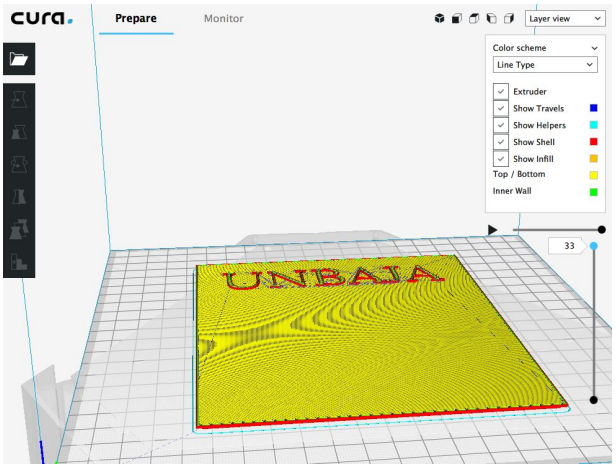


Figura 31 - Tampa, software Cura.

Por fim, utilizou-se dos códigos .g gerados e realizou-se a impressão das peças na impressora 3D (ANET A8), e obteve-se os objetos de acordo com a figura 32 e 33. Além disso, figura 34 mostra a alocação dos componentes na caixa.

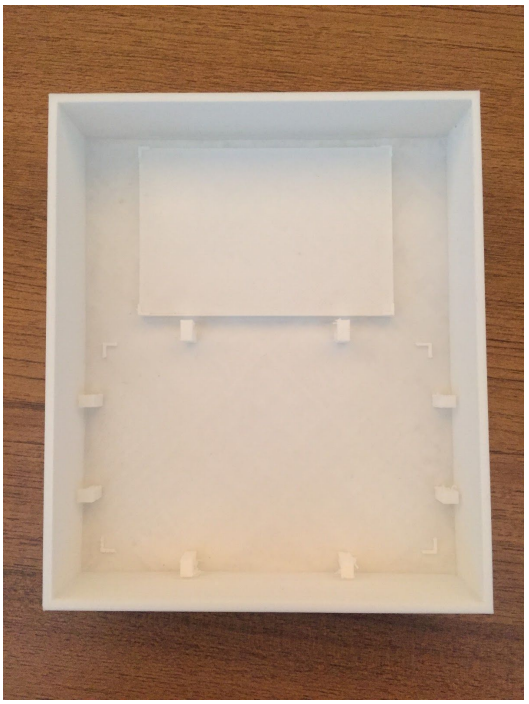


Figura 32 - Caixa impressa na impressora 3D.

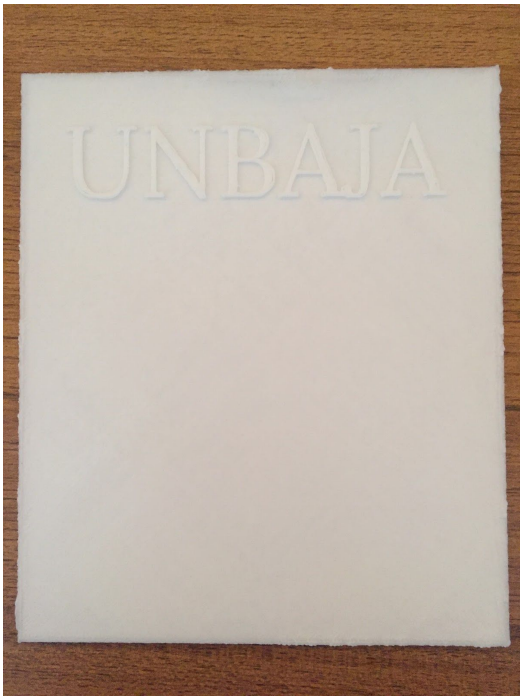


Figura 33 - Tampa impressa na impressora 3D.

Figura 34 - Disposição dos componentes na caixa finalizada.

IX. Resultados e Conclusões

Para a validação do projeto de modo a verificar a experimentalmente a capacidade do módulo GPS de obter dados suficientes precisos para a aplicação desejada, fez-se um teste de deslocamento a bordo de um veículo próprio nas ruas da Qna em Taguatinga-Sul, próximo a casa do integrante do projeto.

Observa-se o trajeto onde as cores vermelhas representam a coleta de vários dados muito próximos, ou seja, constatando que o carro ficou um tempo parado naquele local, na garagem há uma maior concentração e nos cruzamentos das ruas.

A API do *Google* utilizada como interface de apresentação gráfica de informações se mostrou adequada para o projeto, exibindo os dados de uma maneira visual intuitiva e simplificada, mostrando o trajeto percorrido a partir de manchas verdes e vermelhas no mapa.

O servidor também mostrou-se adequado para o projeto, funcionando como o esperado. Após um requerimento é gerado um *log* pelo Raspberry que pode retornar o IP da máquina que está requisitando acesso ao monitoramento.

Figura 35 - Mapa de Calor.

X. Referências

[1] Disponível em : <http://www.petrobras.com.br/fatos-e-dados/equipes-e-dados/equipes-brasileiras-de-sae-baja-se-destacam-em-competicao-internacional.htm>. Acesso em: 02 março.2018.

[2] Disponível em :< http://portal.saebrasil.org.br/>

[3] Disponível em : <http://portal.saebrasil.org.br/Portals/0/PE/Baja%20Nacional%202018/RATBSB.pdf>. Acesso em: 02 março.2018.

[4] Evandro Rui Fernandes. Sistema On-Line de Localização e Roteamento Urbano. Curitiba, Novembro de 2006.

[5] Brito, Matheus Hoffmann. *Desenvolvimento e Dimensionamento Do Projeto Do Subsistema De Direção De Veículo Fora-de-estrada.* Departamento de Engenharia Mecânica, 2002.

[6] William, F; Douglas, L. Race Car Vehicle Dynamics. SAE International, 1994.

[7] Disponível em : <https://developers.google.com/maps/documentation/javascript/tutorial>. Acesso em: 15 abril.2018.

[8] Disponível em : <https://developers.google.com/maps/documentation/javascript/heatmaplayer>. Acesso em: 16 abril.2018.

[9] Disponível em : <https://developers.google.com/maps/documentation/javascript/reference#HeatmapLayerOptions>. Acesso em: 02 junho.2018.
[10] Disponível em : <https://projects.apache.org/project.html?httpd-http_server>. Acesso em: 02 junho.2018.

Apêndice

Códigos do sistema encontra-se nos repositórios do github dos projetistas.