Danilo Jun Shibata
Leandro Donizetti Soares

Análise de Direção de Motoristas Utilizando o Protocolo OBD2 e Sensores Embarcados

São Paulo

2015

Danilo Jun Shibata
Leandro Donizetti Soares

Análise de Direção de Motoristas Utilizando o Protocolo OBD2 e Sensores Embarcados

Trabalho de Formatura apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia

São Paulo
2015

Danilo Jun Shibata
Leandro Donizetti Soares

Análise de Direção de Motoristas Utilizando o Protocolo OBD2 e Sensores Embarcados

Trabalho de Formatura apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia

Orientador: Prof. Dr. Reginaldo Arakaki

São Paulo
2015

Este trabalho é dedicado às nossas famílias que
nos apoiaram durante todo o decorrer da graduação.

Agradecimentos

Agradecemos primeiramente ao nosso amigo Felipe Massao Kitanaka Matsuoka, que foi membro do grupo por grande parte do projeto e ajudou a formular parte da especificação.

Agradecemos ao nosso orientador, Prof. Dr. Reginaldo Arakaki, e ao Marcelo Angelo Pita e Leandro Rodrigues de Souza, que atuaram como co-orientadores, por toda orientação e o auxílio prestado na especificação e implementação do projeto.

Agradecemos à nossa amiga Maria de Fátima Salgueiro, do Departamento de Engenharia de Computação, pelos materiais fornecidos que utilizamos na conexão do Raspberry Pi e outros sensores embarcados.

Agradecemos à todos nossos colegas de turma, com quem tivemos o enorme prazer de conviver e aprender durante estes anos de graduação.

Agradecemos, por fim, à Escola Politécnica da USP e todo o seu corpo docente, pelas lições e ensinamentos que, mesmo por vezes duros e difíceis, moldaram-nos em pessoas melhores.

"O ótimo é inimigo do bom".
(Voltaire)

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CI	*Circuito integrado*
CNRI	*C*orporation for National Research Initiatives
CWI	*S*tichting Mathematisch Centrum
DETRAN	*Departamento Estudual de Trânsito*
ECU	*Engine Control Unit*
ELM327	CI que traduz informação do OBD2 para serial
GPIO	*General Purpose Input/Output*
GPL	*G*eneral Public License
GPS	*Global Positioning System*
IBM	*International Business Machines*
L2CAP	*Logical Link Control and Adaptation*
MAC	*M*edia Access Control
OBEX	Abreviação de Object Exchange
OBD2	*On Board Diagnostics* padrão de segunda geração
RFCOMM	*Radio Frequency Communication(s)*
RPM	*Rotações por minuto*
RS232	Padrão de troca de dados serial
SAE	*Society of automotive Engineers*
UART	*Universal asynchronous receiver/transmitter*
USB	*Universal Serial Bus*
Wi Fi	Denominação para classe de dispositivos de rede local sem fios baseados no padrão IEEE 802.11.

Sumário

1 Introdução

1.1 Objetivo

O objetivo deste trabalho é a concepção e implementação de um sistema de aquisição de dados de automóveis, que serão utilizados para atribuir notas aos trajetos dos motoristas com o intuito de possibilitar a melhoria do seu comportamento no volante.

A avaliação dos motoristas utilizará a velocidade, aceleração lateral e longitudinal, localização, hora atual e rotação do motor para calcular uma pontuação para cada trajeto percorrido, baseando-se na detecção de comportamentos considerados imprudentes e que trazem riscos de acidentes. Por sua vez, esta pontuação auxiliará o motorista a melhorar o modo como dirige, por meio da visualização e monitoramento de sua evolução.

1.2 Motivação

O trabalho tem como foco perfis de pessoas que tem probabilidade maior de causar acidentes. Sua motivação vem de que, apesar das montadoras de carros terem evoluído positivamente na questão da segurança das pessoas em acidentes de automóveis (com a invenção do AirBag, por exemplo), o comportamento dos motoristas no volante tem evoluído negativamente: por exemplo, com a utilização de vários aplicativos no celular enquanto dirige.

Segundo dados do DETRAN do estado de São Paulo, houve um aumento de 100% no número de mortes por acidentes entre 2006 e 2011 no estado. Supondo que este valor tenha aumentado proporcionalmente à taxa de aumento do número de pessoas que dirigem, então obtém-se em 2015 um aumento de 80% em relação a 2011, ao extrapolar estes dados para o ano atual.

Outras pesquisas indicam que as duas maiores causas de acidentes no trânsito são distrações na direção e comportamentos imprudentes no volante, fatores que são ambos mensuráveis de maneira satisfatória por equipamentos eletrônicos.

Por fim, o número de usuários que utilizam celular smartphone no Brasil tem aumentado substancialmente, bem como o uso da internet móvel. Uma pesquisa do CETIC.br aponta que, em 2013, 85% das pessoas com 10 anos de idade ou mais usavam telefone celular, totalizando 143 milhões de brasileiros. Na classe D e E esta proporção é de 69% e na área rural é de 73%. O uso da Internet no celular destacou-se em 2013: 31% dos brasileiros com 10 anos ou mais acessaram a rede pelo aparelho, o que representa 52,5 milhões de pessoas em números absolutos.

Assim, optou-se pela utilização de um smartphone associado a um hardware embarcado como solução para a o problema proposto. Esta solução possibilita a utilização do equipamento do smartphone para captar alguns dados, como posição, bem como o uso de API's para desenvolvimento em plataformas móveis. O uso do hardware embarcado foi adotado para que seja possível comunicar-se com o carro por meio do protocolo OBD-II, bem como utilizar um acelerômetro embarcado, o que facilita na captura das acelerações.

1.3 Organização do Documento

Este documento está organizado em:

2 - Abordagem do Projeto e Ideias: Discute como foram estabelecidas as especificações do projeto, detalhando as e ideias iniciais e as alternativas de implementação da ideia proposta.

3 - Revisão Bibliográfica: - Apresenta os estudos que serviram de base para o projeto, bem como produtos similares ou com ideias semelhantes.

4 - Especificação: - Especifica o projeto em relação aos requisitos, classes, diagrama de sequência e em relação as tecnologias utilizadas.

5 - Métodos: Discute as ferramentas e métodos utilizados para o gerenciamento e execução do projeto.

6 - Resultados: Exibe os resultados obtidos em cada etapa, bem como testes realizados no prótotipo.

1.4 Repositório online

O código escrito durante o desenvolvimento deste projeto pode ser acessado no repositório online:
https://bitbucket.org/lds1804/tcc-obd2/

2 Abordagem do Projeto e Ideias

Neste capítulo, expõe-se as principais abordagens e ideias levantadas durante a fase inicial do projeto, bem como o processo de decisão adotado para a escolha da ideia final. Analogamente à abordagem iterativa de desenvolvimento adotada durante o projeto, cada subcapítulo será uma adição ao anterior.

2.1 Ideias iniciais e alternativas do projeto

As ideias iniciais continham premissas que envolviam o uso do padrão OBD2 para aquisição de dados e alguns algoritmos de Machine Learning (ML), utilizando o OBD-II e celular para aquisição de dados e em penalidades para atribuir notas para os motoristas. Posteriormente, em virtude de sua complexidade, os algoritmos de machine learning foram substituídos por uma simples pontuação ponderada. Assim, a pesquisa inicial tratou os seguintes temas:

- Comportamento de direção de motoristas;

- Padrão OBD2 e ELM327;

- Arduíno;

- Internet of Things;

- Soluções parecidas que utilizam as tecnologias citadas;

O objetivo desta pesquisa foi a consolidação dos conhecimentos do grupo em relação ao estado atual destas tecnologias aplicadas em automóveis.

O OBD2 é um padrão de interface para comunicação com o ECU (Engine Control Unit) de automóveis, sendo obrigatório por lei em todos os carros no Brasil desde 2008. Seu propósito é fornecer um diagnóstico rápido para a manutenção do carro, além de outras informações como a velocidade, RPM e etc.

O ELM327 é um CI fabricado pela ELM eletronics que traduz a informação do padrão OBD2 para protocolo RS232 (padrão de comunicação serial), facilitando a comunicação com o computador.

Além desta pesquisa, outra atividade realizada foi a familiarização com o padrão OBD2, utilizando o chip ELM327 para a conversão dos dados ao formato serial.

Desta maneira, a ideia inicial do projeto consitia em:

Avaliação e reconhecimento rápido - Possibilitar reconhecer rapidamente qual pessoa está dirigindo o carro, além de avaliar a qualidade do motorista.

Coleta de dados remoto - Coletar os dados do carro remotamente e enviar para um servidor responsável pela análise e algoritmos.

Taxa de acerto dos algoritmos - Suportar uma taxa de acerto em torno de 80%.

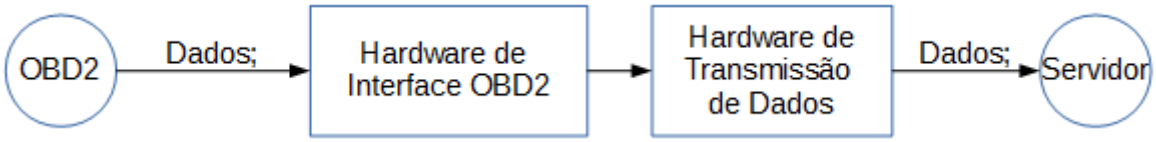

Figura 1 – Esquema da ideia inicial.

A ideia inicial está indicada na figura 1. O padrão OBD2 seria utilizado para coletar a velocidade e RPM do carro, com o auxílio de um hardware dedicado à leitura e conversão destes dados provenientes da ECU. Este hardware seria então conectado à um segundo hardware, por sua vez responsável pelo envio destes dados ao servidor. Para a implementação destes hardwares, as alternativas consideradas foram:

Quecklink GV500: Dispositivo que se conecta ao padrão OBD2 e que faz a transmissão de dados ao servidor;

ELM327/Bluetooth/USB: Hardware que se conecta ao OBD2 e faz a tradução dos dados originais do OBD2 para um padrão mais fácil de ser transmitido, o RS232 (padrão serial). O ELM327 suporta vários meios de transmissão, dentre os quais: Bluetooth, USB e Wi Fi.

Arduíno: Microcontrolador de fácil utilização, muito utilizado para prototipagem. Sua linguagem de programação é uma variação da linguagem C adaptada.

Celular Smartphone: Possui várias funcionalidades como acelerômetro, giroscópio, GPS e um alto processamento.

Nas figuras 2, 3 e 4, destacam-se as principais formas de implementação consideradas:

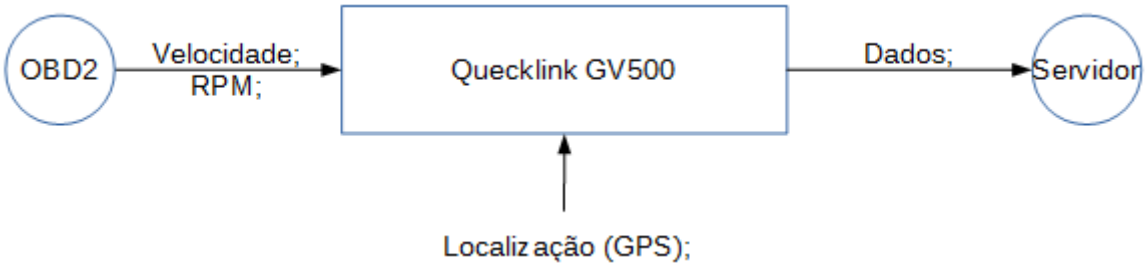

Figura 2 – Quecklink GV500.

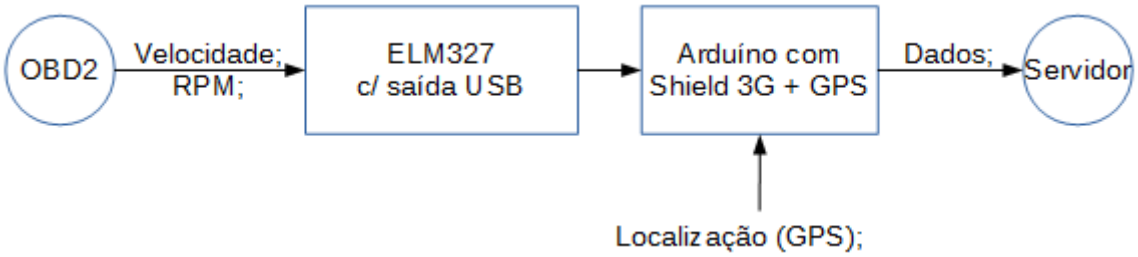

Figura 3 – ELM327/USB + Arduíno.

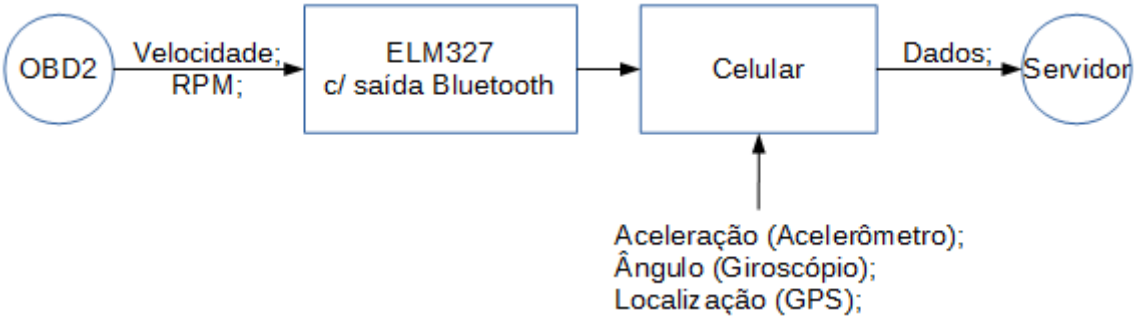

Figura 4 – ELM327/Bluetooth + Celular.

Finalmente, consideradas estas opções, o grupo decidiu implementar uma solução baseada em celular (smartphone) e Raspberry Pi. O celular suporta a utilização da conexão internet para enviar e receber dados ao servidor. A opção pela utilização do um hardware embarcado (Raspberry Pi) é justificada pelas dificuldades encontradas na calibração da orientação do GPS do celular. Assim, acoplando-se um acelerômetro ao Raspberry, facilita-se o processo de coleta da aceleração. A opção pelo uso do Raspberry-Pi foi pela sua maior quantidade de memória e de uma maior capacidade de processamento.

2.2 Ideia escolhida: abordagem com smartphone e hardware embarcado

Dentre as ideias iniciais, escolheu-se a alternativa que utilizava um acelerômetro acoplado à um hardware embarcado devido a maior complexidade de se filtrar os dados de aceleração provenientes do celular. Tal abordagem baseia-se nas premissas de simplicidade e praticidade adotadas no trabalho de mestrado "How's My Driving" (VOBIS, 2012), que aborda a implementação de um sistema como o proposto por este projeto utilizando-se somente o celular e um algoritmo simplificado.

O algoritmo implementado neste artigo é baseado em um sistema de pontos que penaliza comportamentos que, segundo estatísticas, induzem uma maior probabilidade de causar acidentes, tal algoritmo mostra-se interessante no contexto deste trabalho, pois simplifica a complexidade da implementação de um algoritmo eficiente de avaliação do comportamento de motoristas.

Na implementação do protótipo para validação da ideia escolhida, um problema potencial encontrado foi a dificuldade de garantir a captação dos dados de maneira contínua durante o percurso, pois o motorista poderia ativar o aplicativo somente quando quiser ter dados bons, deixando-o de lado quando adotar comportamentos ruins. Assim, o foco deste projeto tornou-se implementar um sistema que possibilite a auto-avaliação do motorista em relação ao modo como dirige, com intuito de melhorar seus comportamentos no volante.

3 Revisão Bibliográfica

Nas próximas seções são apresentados trabalhos que serviram de base para o projeto, bem como trabalhos similares em relação ao uso dos mesmos componentes ou com objetivos semelhantes.

3.1 The 100 Car Naturalistic Driving Study

Neste estudo (DINGUS et al., 2006) foram analisados mais de 100 motoristas pelo período de um ano. O objetivo era medir as principais causas dos acidentes. Eles foram monitorados utilizando um kit instalado nos carros, que era composto por:

- Computador Pentium, que atuava como uma central de coleta dos dados;
- Acelerômetros para medir acelerações laterais e longitudinais;
- Sensor de proximidade lateral;
- Sensor de proximidade frontal e traseira;
- Video-câmeras, para validar eventos detectados pelos sensores e para verificar que a faixa de direção estava sendo mantida.

Na figura 5 é mostrado a central de aquisição de dados, instalada no porta malas do carro.

Figura 5 – Centro de coleta de Dados Instalado no porta Malas do Carro (DINGUS et al., 2006)

Idade Número % do total	Homem	Mulher	Total
18-20	9 8,3%	7 6,4%	16 14,7 %
21-24	11 10,1%	10 9,2%	21 19,3%
25-34	7 6,4%	12 11,0%	19 17,4%
35-44	4 3,7%	16 14,7%	20 18,3%
45-54	7 6,4%	13 11,9%	20 18,3%
55+	5 4,6%	8 7,3%	13 11,9%
Total Número Total %	43 39,4 %	66 60,6 %	109 100%

Tabela 1 – Tabela com a Distribuição dos Motoristas Analisados no Estudo (DINGUS et al., 2006)

Como já dito anteriormente para coletar a distância para os carros utilizou-se um sensor de distância, este pode ser visto na figura 6.

Figura 6 – Antena do Radar Doppler Instalado na parte Traseira do Carro (DINGUS et al., 2006)

Os motoristas do estudo seguiam a distribuição mostrada pela tabela 1.

Para o projeto, os dados mais importantes foram os valores de limiares de aceleração para eventos de quase colisão, estes valores foram utilizados para dar penalidades

aos motoristas no projeto desenvolvido. Para o princípio do projeto foram adotados como limiares os mesmos valores de aceleração utilizados pelo estudo, de aceleração lateral igual ou maior que 0,7 G e aceleração longitudinal com módulo maior que 0,6 G.

3.2 How's My Driving

Nesta tese de mestrado (VOBIS, 2012) é proposta uma forma de avaliar o motorista utilizando o celular iPhone, para tal são utilizados somente os sensores do celular.

A primeira interação da interface de usuário para o registro de viagens é mostrado na figura 7, onde é mostrada a aceleração captada pelo acelerômetro, bem como a velocidade calculada, a distância percorrida e a pontuação atual calculada.

Figura 7 – Interface de Usuário do projeto How's my Driving (VOBIS, 2012)

Na figura 8, é mostrada a segunda interação da interface de usuário, onde é mostrada a rota do usuário no mapa, bem como a distância percorrida e a velocidade atual, a pontuação do usuário e a previsão do tempo. No lado direito da figura 8 é mostrado uma tela onde é possível atribuir nomes as rotas.

Figura 8 – Segunda Iteração da Interface de Usuário do projeto How's my Driving (VOBIS, 2012)

Uma dos métodos aproveitados deste projeto foi o método de penalidades, onde é possível atribuir notas para as jornadas dos usuários baseados nas infrações cometidas ao longo do trajeto. No estudo são definidos três tipos de incidentes: aceleração, desaceleração e curvas acentuadas. A pontuação varia de 1,0 à 3,5, sendo 1,0 indicando um pequeno incidente e 3,5 indica um grande incidente. A penalidade atribuída ao incidente é composta por três partes: uma parte básica, uma influenciada pelo pico da aceleração e outra pela duração da aceleração. Isto está sumarizado na equação a seguir:

$$\mathbf{Penalidade = bP + gP + dP}$$

Onde bP indica a penalidade, gP refere-se ao pico de aceleração e dP correlacionado à duração da aceleração.

3.3 Driver Evaluation System Using Mobile Phone and OBD-II System

Neste projeto utilizou-se dados obtidos pelo OBD-II e alguns obtidos pelo celular, para avaliar a direção do motorista e também quantificar o consumo de combustível.

Primeiramente foi utilizado um algoritmo para avaliar a aceleração captada, está foi captada nos três eixos, e foi dada uma nota para cada eixo dependendo de quão suave

foi a direção do motorista. A pontuação de cada eixo contribui para formar a pontuação geral da aceleração.

Em outro algoritmo a máxima eficiência do carro é obtida e comparada com a atual, a partir disso é atribuída uma nota para a direção em relação ao consumo de combustível por Km.

Este algoritmo pode ser visualizado na figura 9.

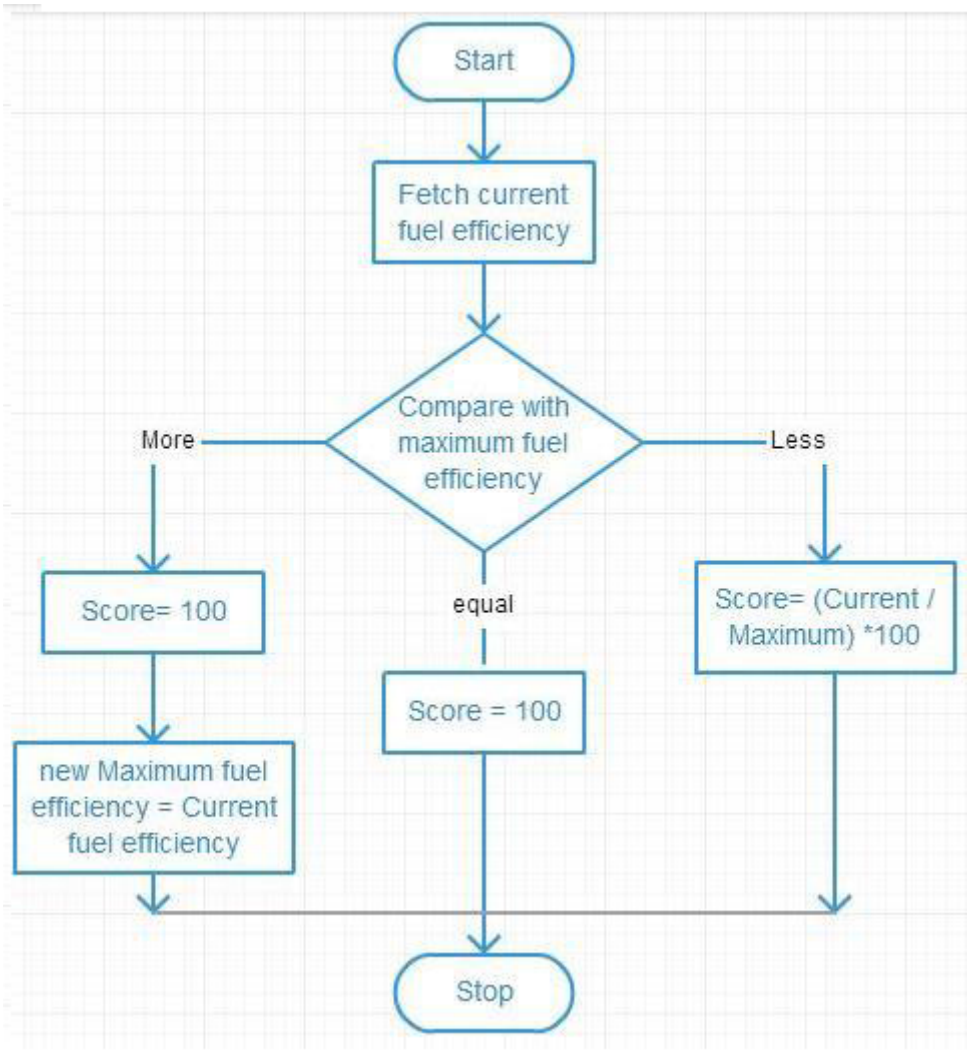

Figura 9 – Algoritmo para Cálculo de Eficiência do Uso de Combustível no Artigo (AKSHAYKUMAR; MANJARI; SRINIVAS, 2015)

A arquitetura do sistema contém os seguintes elementos:

- Carro com suporte ao protocolo OBD-II;
- Adaptador OBD-II USB ou Bluetooth;
- Aplicativo Torque app;

- Aplicativo DES;
- Servidor;
- Central de Visualização.

Os dados obtido pelo aplicativo DES(Desenvolvido pela equipe do artigo) são de arquivos .csv salvos pelo aplicativo Torque app, que foi desenvolvido por terceiros. Estes dados então são analisados pelo celular, junto com dados de aceleração, para compor a pontuação geral do usuário.

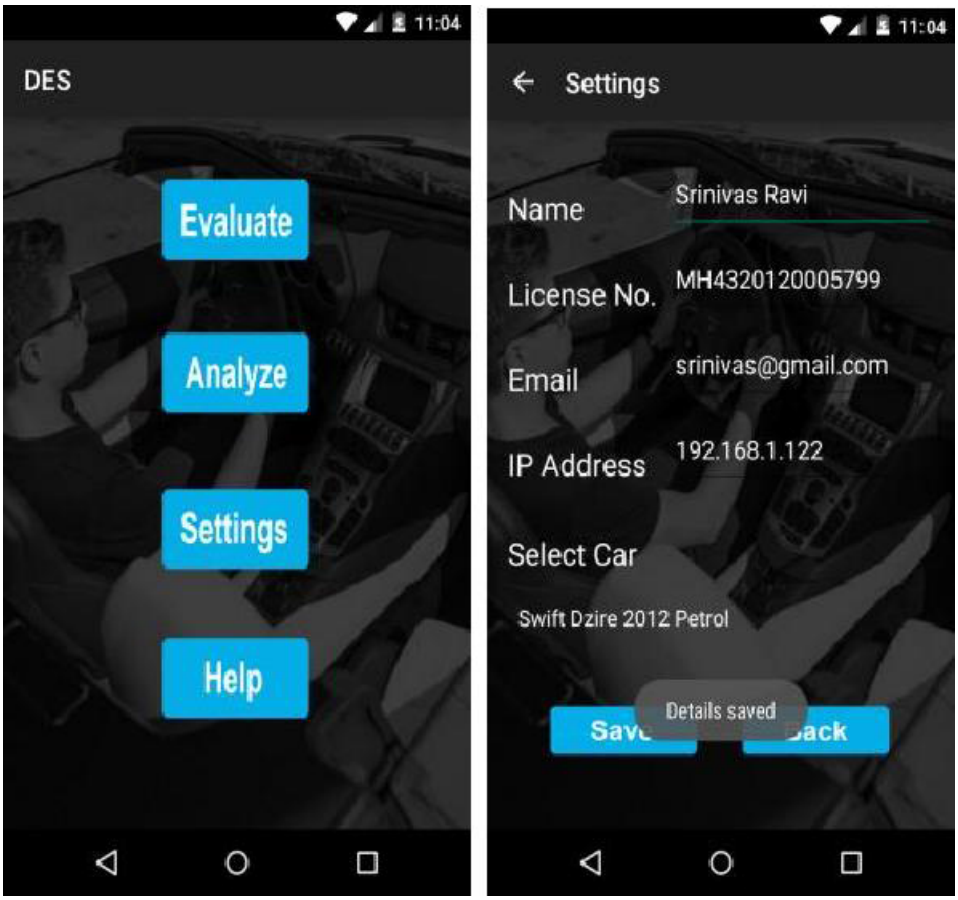

Figura 10 – Telas de aplicativo DES Desenvolvido pela equipe do Artigo (AKSHAYKUMAR; MANJARI; SRINIVAS, 2015)

No aplicativo é possível avaliar uma rota logada no telefone por meio do botão evaluate, também é possível analisar a direção com base em um dado parâmetro, assim sendo exibido para o usuário uma tela com o histórico referente ao parâmetro fornecido.

Esta tela pode ser vista na figura 11, onde é mostrada uma tela com a pontuação do usuário na esquerda, e a pontuação referente a cada parâmetro, já na direita pode ser vista a evolução com o tempo em relação a uma dada variável.

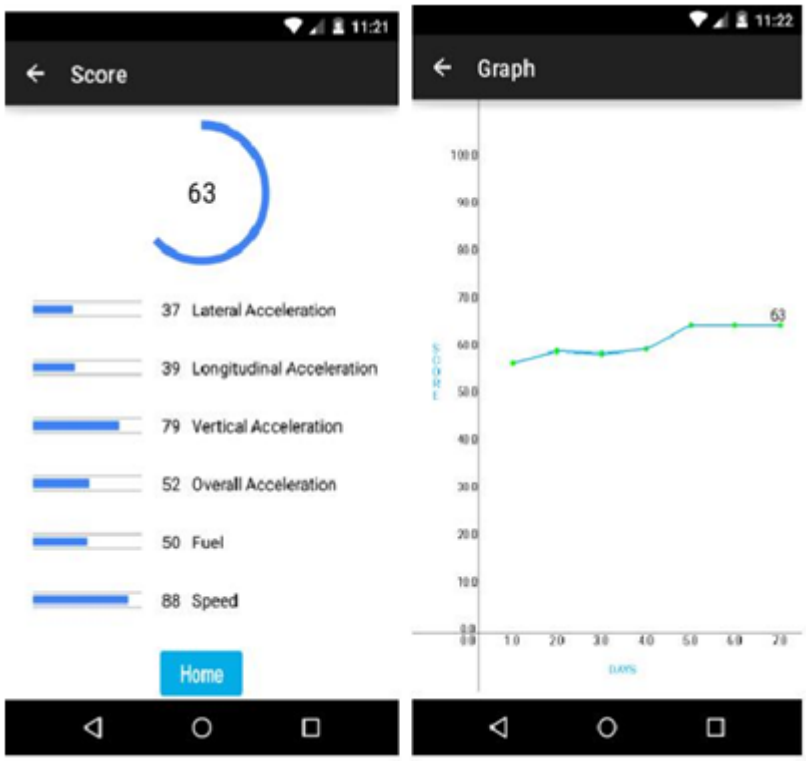

Figura 11 – Telas de aplicativo DES Mostrando a Pontuação e o Gráfico em Relação a uma Variável (AKSHAYKUMAR; MANJARI; SRINIVAS, 2015)

3.4 Cobli

Cobli é uma empresa especializada em vender produtos para avaliação de direção. O produto é oferecido de três diferentes formas:

- Aplicativo de celular para Android e IOS;
- Rastreador Solar, que captura localização e comportamento do motorista;
- OBD Cobli (Dispositivos de Diagnósticos Embarcado Cobli).

O aplicativo tem por objetivos tornar a direção mais segura e produtiva fornecendo ao motorista uma pontuação conforme dirige e uma informação a respeito das condições do carro.

O layout do aplicativo está ilustrado na figura 12, onde é possível ver o trajeto bem como a pontuação, além de outras informações adicionais.

Figura 12 – Interface Gráfica do Aplicativo Cobli registrando uma Viagem (COBLIE, 2015)

Com o aplicativo é possível obter o comportamento do motorista, realizar o monitoramento em tempo real, controlar o uso do celular enquanto dirige e acompanhar a localização do motorista.

Em adição ao aplicativo pode-se instalar no carro o painel solar, mostrado na figura 13. Ele inclui as funcionalidades do aplicativo com as vantagens de ser conectado ao carro, permanecer sempre conectado a rede, pois utiliza energia solar e possibilita o cruzamento de informações com o aplicativo para obter dados mais confiáveis.

Figura 13 – Rastreador Solar Cobli (COBLIE, 2015)

Por fim pode-se instalar um conector OBD, mostrado na figura 14, que provê as funções adicionais de manutenção preventiva, telemetria da performance do carro e detecção de falhas.

Figura 14 – Conector OBD Cobli (COBLIE, 2015)

4 Especificação

Esta seção baseia-se nos 5 pontos fundamentais da ODP (Open Distributed Processing) para especificar o projeto a ser desenvolvido. Assim, divide-se este capítulo em: Visão Empresarial, de Informação, Computacional, de Engenharia/Infraestrutura e de Tecnologia.

4.1 Visão Empresarial

4.1.1 Monetização do projeto

O projeto a ser desenvolvido tem o intuito de analisar o comportamento do motorista ao dirigir, podendo assim ser utilizado como uma ferramenta de grande auxílio para companhias de seguro e monitoramento de frota.

Companhias de seguros poderiam utilizar o sistema proposto para implementar um modelo de seguros para automóveis baseado em uma cobrança variável, segundo os hábitos de direção do motorista. Se um cliente é bastante cuidadoso no volante, o preço do seguro de seu carro seria menor. Contudo, seria necessário realizar algumas melhorias no projeto, como por exemplo:

- Impedir que o usuário desligue o módulo com intuito de mascarar os dados captados. Sem esta funcionalidade, um usuário mal-intencionado poderia desligar o aparelho quando estiver dirigindo de maneira perigosa;
- Enviar os dados sem o uso do celular do usuário, de forma a atuar independente do dispositivo.

Empresas de monitoramento de frotas poderiam utilizar o sistema proposto para garantir a qualidade do comportamento de seus motoristas. Neste cenário, a melhoria necessária seria:

- Adicionar permissões de acesso aos usuários administradores da frota, que poderiam visualizar o histórico de toda a frota cadastrada.

Desta maneira, com a implementação das funcionalidades especificadas acima, pode-se criar maneiras de criar renda e monetizar o sistema proposto.

4.2 Visão da Informação

Nesta seção são especificadas as informações coletadas e transmitidas pelo projeto. Elas são captadas em duas diferentes partes do sistema: no Raspberry, utilizando o chip ELM 327 e no dispositivo celular Android. Mais detalhes são fornecidos nas seções subsequentes.

4.2.1 Elementos de Informação do Sistema

As informações são captadas em várias partes do sistema, como ilustrado no diagrama da figura 15:

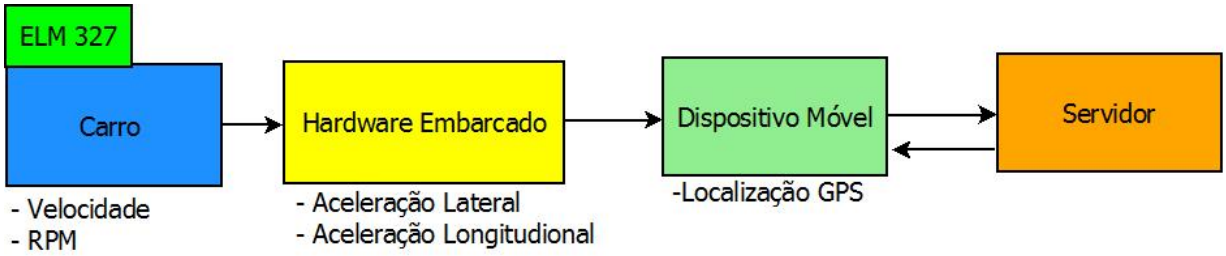

Figura 15 – Informações Captadas em cada parte da Arquitetura

Os dados captados pelo sistema são:

- **Velocidade e RPM:** Captada diretamente do carro;

- **Aceleração Lateral e Longitudional:** Coletada do acelerômetro conectado ao hardware embarcado;

- **Tempo atual:** Coletado do hardware embarcado;

- **Localização:** Captado do GPS do dispositivo móvel.

4.2.2 Manipulação das Informações

As informações são manipuladas no hardware embarcado e no celular, sendo posteriormente armazenadas no servidor. No hardware embarcado, as informações de velocidade e RPM são captadas e enviadas para o celular.

As acelerações são captadas pelo hardware embarcado e são enviadas somente se passarem um limiar pré-estabelecido. Todas as informações incluem um campo indicando o tempo em que foram captadas.

Estes dados são convertidos para o formato JSON antes de serem enviados ao dispositivo móvel (smartphone), que os recebe e interpreta de modo a utilizá-los para

detectar as eventuais infrações cometidas. No celular, também são recebidas informações do servidor com relacão ao histórico de pontuações.

Os dados são então transmitidos do celular ao servidor, que os utiliza para o cálculo da pontuação.

Um resumo com a manipulação das informações pelo sistema é mostrada no diagrama da figura 16:

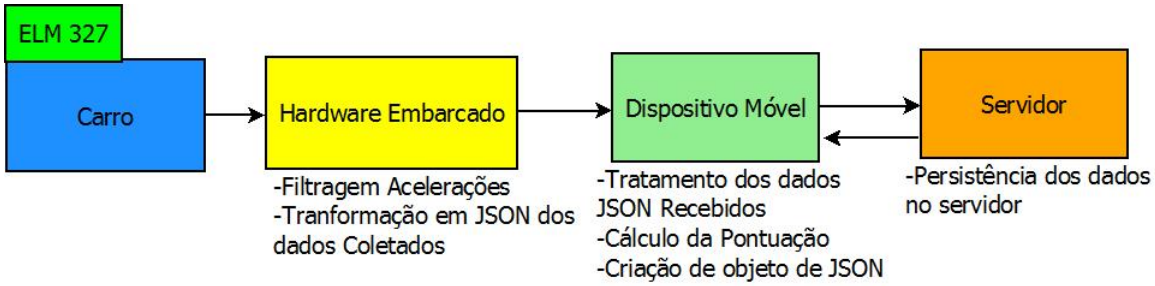

Figura 16 – Manipulação das Informações em cada parte da Arquitetura

4.2.3 Fluxo das Informações

No projeto, o fluxo de informações entre os nós da arquitetura é unidirecional. A única exceção ocorre na comunicação entre o dispositivo móvel e o servidor, onde o fluxo é bidirecional. Na figura 17, em cada flecha de comunicação, é mostrada de forma genérica a tecnologia utilizada entre os nós para troca de informações.

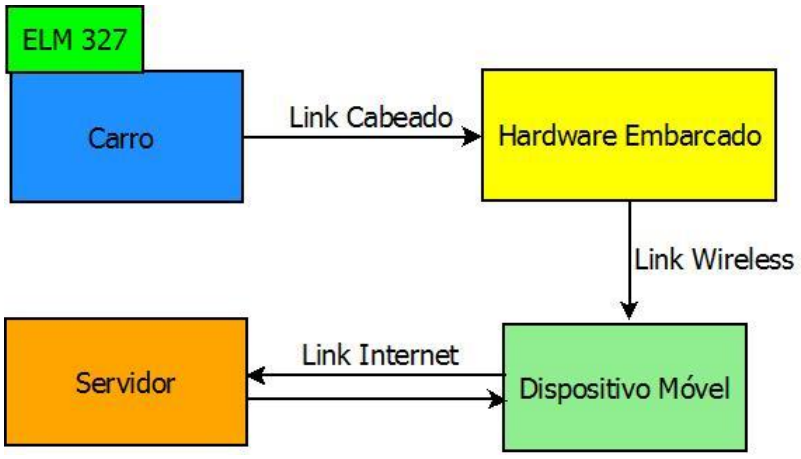

Figura 17 – Tecnologias Utilizadas para Conexão entre os Nós

As tecnologias para intercomunicação são as seguintes:

-Serial RS232- Link Cabeada: Comunicação entre hardware embarcado e chip ELM 327. O hardware embarcado escreve no serial e aguarda uma resposta do ELM 327, que é escrita no serial.

-Bluetooth RFCOMM-Link Wireless: Uso do protocolo Bluetooth RFCOMM para interconexão entre dispositivo móvel e hardware embarcado.

-HTTP TCP-IP- Link Internet: Uso do protocolo HTTP e TCP-IP para comunicação entre dispositivo móvel e servidor.

As mensagens trocadas entre cada nó são detalhadas a seguir.

- Carro e Hardware embarcado : As mensagens seguem o padrão SAE J1939, sendo que as mensagens utilizadas no projeto são as responsáveis por requisitar velocidade e RPM, possuindo o seguinte formato de requisição:

01 0C

Por exemplo, a requisição acima solicita o RPM ao carro. Para requisitar a velocidade, basta utilizar o comando 01 OD, no caso, OD refere-se ao código da velocidade.

Uma possível resposta para a requisição acima é a seguinte:

41 OC 1A F8

Os primeiros dígitos **41** indicam que a requisição aconteceu sem erros, no modo 1. A parte **OC** indica que o RPM está sendo retornado. Por fim o valor **1A F8** indica o valor RPM em hexa multiplicado por 4.

- Hardware Embarcado e Aparelho Móvel: Entre estes nós são trocados mensagens em strings no formato JSON. Um dos formatos das strings JSON fornece a velocidade, RPM e o tempo atual, já o outro formato fornece as acelerações e o tempo atual. A seguir seguem os dois tipos de formato.

- JSON Velocidade : **{ “velocidade” : dados[0], “rpm” : dados[1], “agora” : str(dados[2]) }**

- JSON Aceleração: **{ “agora” : str(dados[0]) , “aceleracaoX” : dados[1], “aceleracaoY” : dados[2] }**

Onde dados[x], x indo de 0 à 2, representa o valor dos dados em float, inteiro ou datetime.

-Dispositivo Celular e Servidor: Entre o celular e o servidor são trocadas mensagens no formato HTTP.

No diagrama da figura 18 é mostrado um resumo do formato das mensagens trocadas entre os nós.

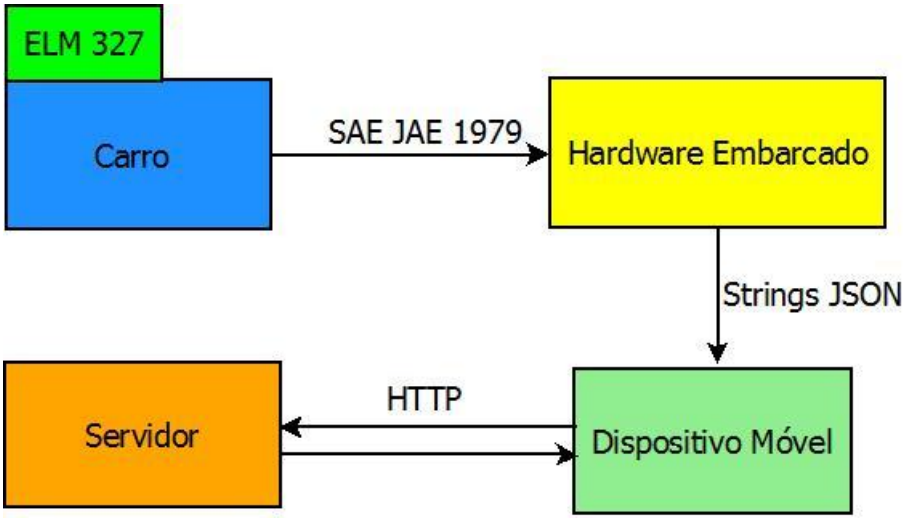

Figura 18 – Formato de Mensagens Utilizados para Comunicação ponto-a-ponto

4.3 Visão Computacional

4.3.1 Requisitos

Nesta seção, apresentam-se os requisitos funcionais e não funcionais do sistema proposto.

4.3.1.1 Requisitos Funcionais

Identificação	RF01
Nome	Captura de dados do automóvel
Descrição	O sistema deve suportar a captura dos seguintes dados do automóvel: velocidade, aceleração longitudinal e lateral, RPM (Rotações por Minuto) do motor, localização do usuário via GPS e data atual.
Explicação	Os dados especificados serão utilizados para descrever uma rota do carro e detectar comportamentos imprudentes do motorista.

Identificação	RF02
Nome	Detecção de comportamentos imprudentes
Descrição	O sistema deve suportar a identificação de comportamentos imprudentes do usuário a partir dos dados coletados do automóvel. Cada comportamento imprudente é tratado como um evento ocorrido na rota. Assim, o sistema deve identificar os seguintes eventos: velocidade acima do limite da pista, aceleração brusca, freada brusca e rotação do motor muito alta.
Explicação	Os eventos identificados serão contabilizados para calcular a pontuação do motorista na rota.

Identificação	RF03
Nome	Cálculo da pontuação do usuário
Descrição	O sistema deve calcular a pontuação do motorista em uma rota, utilizando-se os eventos detectados. Quanto maior o número de eventos (comportamentos imprudentes) identificados, menor será a pontuação do motorista e vice-e-versa.
Explicação	A pontuação do motorista na rota será utilizada pelo usuário para determinar se houve melhoria na qualidade de direção do motorista. Assim, a pontuação calculada representa uma medida quantitativa do comportamento do motorista na rota.

Identificação	RF04
Nome	Visualização da rota
Descrição	Ao iniciar uma rota, o sistema deve apresentar ao usuário um mapa destacando-se o caminho percorrido pelo carro.
Explicação	A interface de usuário baseada em mapas poderá ser utilizada para visualizar onde ocorreu cada evento.

Identificação	RF05
Nome	Visualização de eventos no mapa
Descrição	Ao iniciar uma rota, o sistema deve apresentar, em tempo-real, quais eventos ocorreram e onde eles foram detectados no mapa. Um marcador deve ser fixado nas posições onde os eventos se passaram, assim como uma notificação deve ser apresentada ao usuário.
Explicação	Ao apresentar em tempo real onde e quais eventos ocorrem, o usuário possui meios para determinar quais são seus pontos de melhoria e onde ele comete falhas de direção.

Identificação	RF06
Nome	Persitências dos dados no servidor
Descrição	Ao final de uma rota, o sistema deve salvar os dados sobre os eventos ocorridos, bem como a pontuação do motorista, no banco de dados.
Explicação	As informações registradas serão utilizadas para a visualização de um histórico da pontuação do motorista.

Identificação	RF07
Nome	Visualização da pontuação e resumo da rota
Descrição	O sistema deverá apresentar um resumo dos eventos ocorridos em cada rota salva pelo usuário, com os eventos ocorridos e a pontuação do motorista.
Explicação	Por meio da visualização de sua pontuação e resultado da rota, o usuário poderá determinar se sua qualidade de direção foi aceitável durante o trajeto.

Identificação	RF08
Nome	Visualização do histórico da pontuação
Descrição	O sistema deverá possibilitar a visualização dos resumos de rotas e pontuações de todas as rotas previamente salvas pelo motorista.
Explicação	Por meio da visualização do histórico de pontuações, o usuário poderá determinar se sua qualidade de direção está melhorando no tempo, bem como identificar onde ele ainda precisa melhorar.

4.3.1.2 Requisitos Não Funcionais

Identificação	RNF01
Nome	Taxa de envio de dados ao hardware de comunicação
Descrição	O hardware embarcado de captura de dados do carro deve minimizar o número de requisições enviadas ao hardware de comunicação com o servidor. A taxa de envio de dados aceitável deve ser no máximo: 1 conexão a cada 200ms.
Explicação	A comunicação entre dispositivos é uma operação que demanda um tempo de execução relativamente elevado. Assim, para otimizar o uso dos recursos computacionais limitados do hardware, deve-se limitar o número de requisições realizadas.

Identificação	RNF02
Nome	Taxa de envio de dados ao servidor
Descrição	O hardware de comunicação deve enviar os dados coletados ao servidor apenas no final do percurso.
Explicação	A comunicação entre dispositivos é uma operação que demanda um tempo de execução relativamente elevado. Assim, para otimizar o uso dos recursos computacionais limitados do hardware, deve-se limitar o número de requisições realizadas.

Identificação	RNF03
Nome	Uso de threads exlusivas para comunicação
Descrição	O sistema deve utilizar threads exlusivas para a comunicação de dados, no nível do hardware de captura de dados e do hardware de comunicação com o servidor.
Explicação	A comunicação entre dispositivos é uma operação que demanda um tempo de execução relativamente elevado. Assim, para evitar o comprometimento da função principal de cada hardware, deve-se utilizar threads dedicadas apenas à parte de comunicação e conexões.

4.3.2 Casos de Uso

A partir dos requisitos funcionais especificados na seção anterior (4.3.1), identificam-se os casos de uso do sistema (Figura 19):

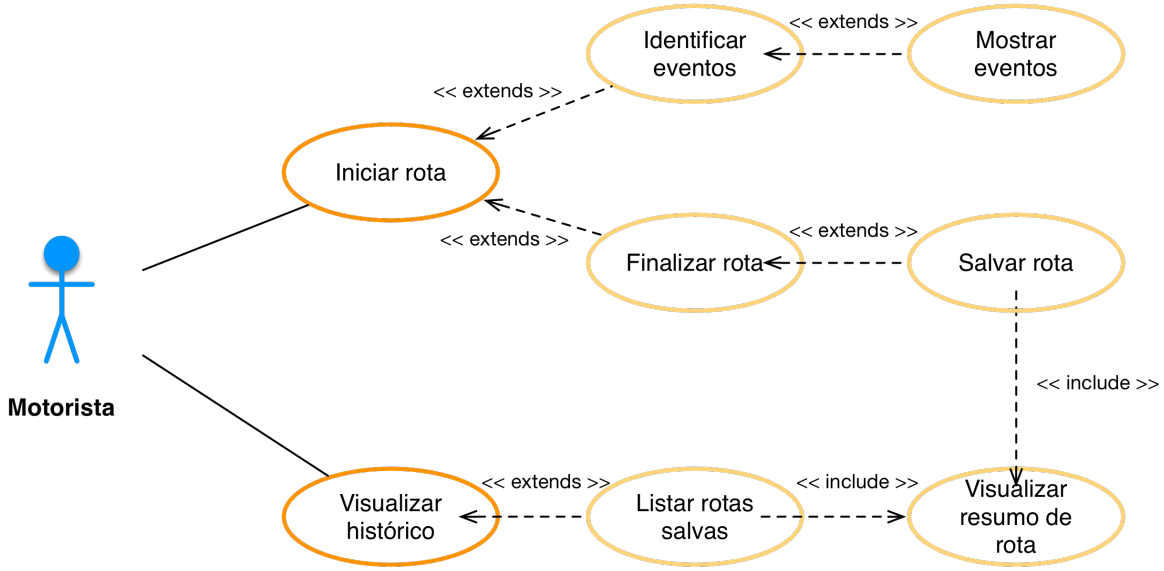

Figura 19 – Casos de Uso

O usuário do sistema é o motorista do carro, que pode realizar duas ações principais: iniciar a rota e visualizar o histórico.

4.3.3 Fluxos do sistema

Nesta seção é feita uma descrição da arquitetura do programa, mostrando um diagrama de blocos do projeto, em seguida são descritos os softwares utilizados para captura de dados no Raspberry e o do aplicativo Android

4.3.3.1 Software de Captura de Dados do Raspberry

Diagrama de Classes

O script em Python responsável por capturar dados é bem simplificado, possuindo somente duas threads, que são responsáveis por capturar dados da velocidade e da aceleração. Foi adotado o uso de threads, pois o tempo de captura do acelerômetro é da ordem de 10mS, enquanto a captura do OBD dura cerca de 200ms. O diagrama de classes do script Python é mostrado na figura 20.

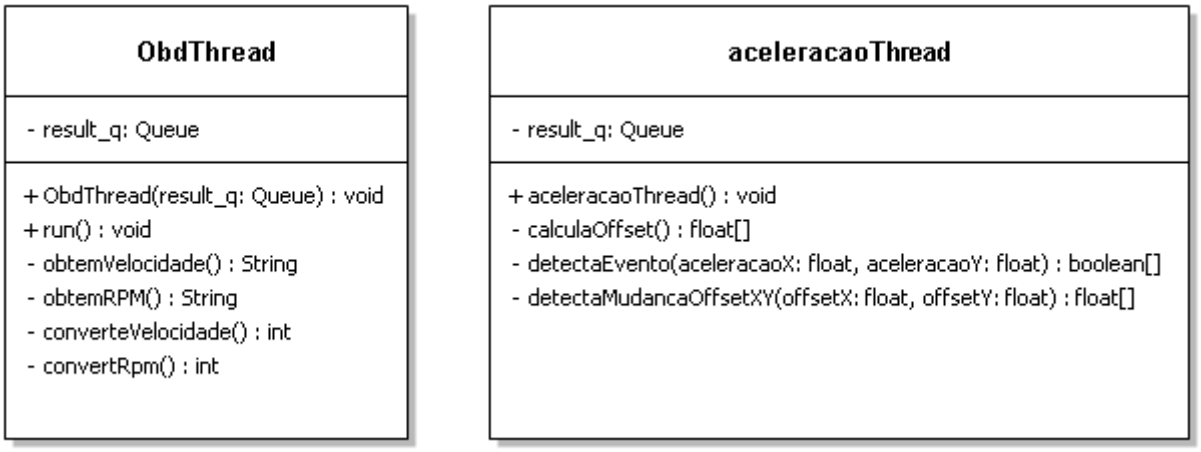

Figura 20 – Diagramas de Classes do script Python de Coleta de Dados

Ambas as classes possuem uma fila de saída, responsável por capturar dados na thread e enviar a thread principal, onde existem duas filas diferentes para recebimento de dados.

A classe ObdThread possui os seguinte métodos:

- **ObdThread:** Thread responsável por captar dados do OBD e enviá-los ao Android por meio de uma conexão RFCOMM.
- **run:** Classe que que é rodada ao se chamar o método start da thread.

- **private String obtemVelocidade:** Método que obtém uma string velocidade, por meio da criação de um serial, do envio do pedido de dados ao chip ELM 327 e posterior espera e interpretação da resposta.
- **private String obtemRPM:** Método semelhante ao obtemVelocidade, mas que solicita o RPM do veículo ao chip ELM 327.
- **converteVelocidade:** Método que recebe uma string da velocidade e a converte para inteiro.
- **converteRpm:** Recebe uma string com o RPM e converte este dado para o respectivo valor em inteiro.

A classe aceleracaoThread possui os seguintes métodos:

- **aceleracaoThread**: Inicializa a fila de saída da thread com uma fila passada como parâmetro.
- **calculaOffset:** Calcula o offset inicial, que será utilizado para obtenção dos valores das acelerações.
- **detectaEvento:** Detecta caso algum valor de aceleração lateral ou longitudinal ultrapassou o limite estabelecido.
- **detectaMudancaOffset():** Detecta se uma mudança no valor das acelerações permanece por certo tempo, caso isso seja verdade altera o valor do offset do eixo correspondente.

Diagrama de sequência

No script Python existem duas threads, uma responsável por capturar os dados do carro por meio do protocolo OBD e outra com a função de coletar dados da aceleração. Esta duas threads captam dados e colocam em uma fila, que é lida pela main thread, que por sua vez envia estes dados para o Android utilizando o socket RFCOMM Bluetooth.

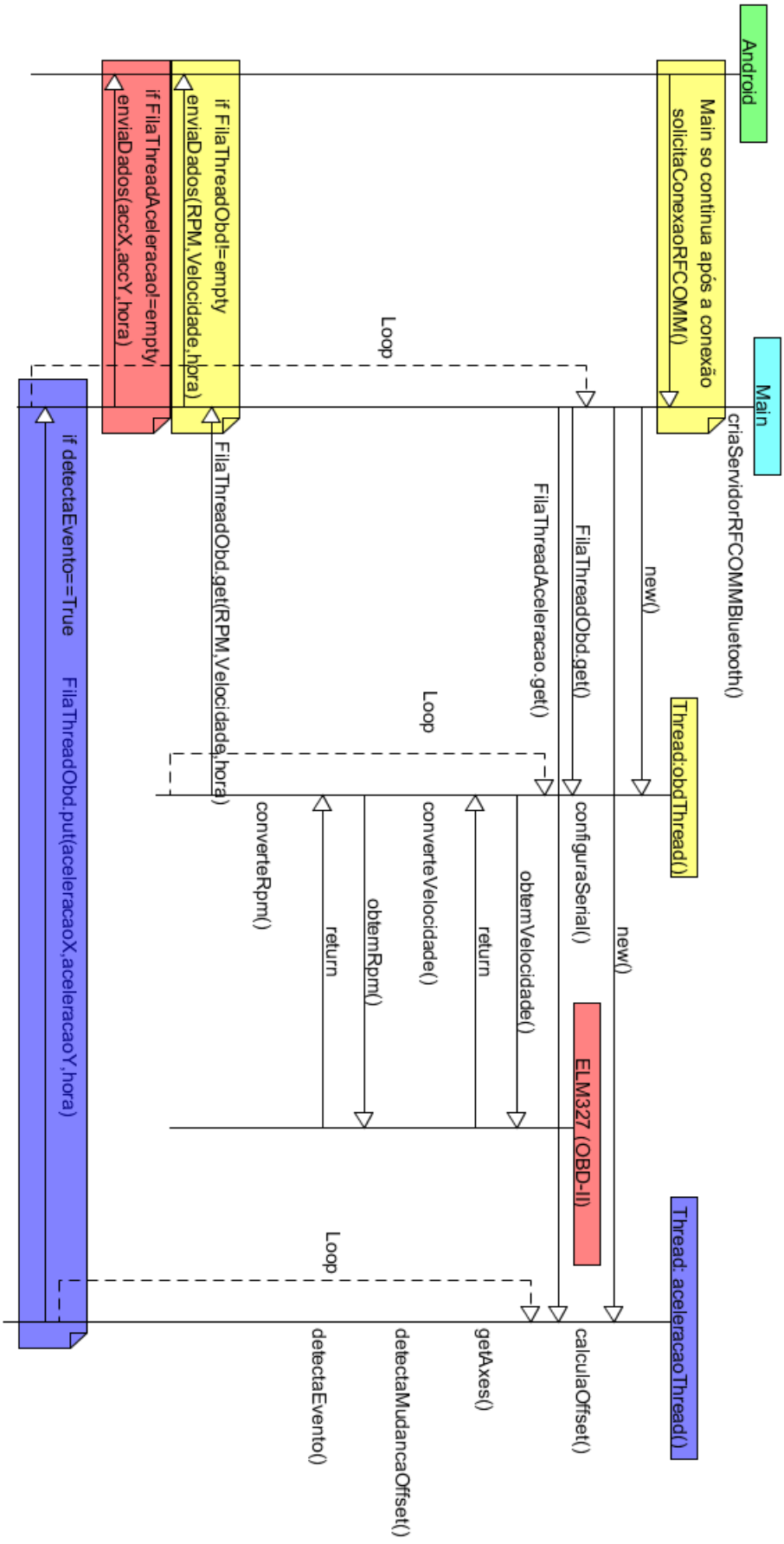

Figura 21 – Diagramas de Sequência do script Python de Coleta de Dados

A thread Main cria um servidor RFCOMM e então aguarda por uma conexão de um cliente, somente então continua com a sua execução. Posteriormente, ela cria duas threads, que operam independentemente e então entra em um loop infinito, onde verifica se novos dados chegaram. Caso isso seja verdade, estas informações são enviadas para o dispositivo Android.

A obdThread configura o canal serial, para então entrar um loop infinito, onde requisita e converte a velocidade e o RPM e os coloca na fila de saída para a Main thread.

A aceleraçãoThread calcula inicialmente o offset, para entrar em um loop infinito, onde coleta os dados do acelerômetro, detecta a mudança de direção do acelerômetro, verifica se os valores passaram os limites estabelecidos, e em caso afirmativo envia estes dados para a Main Thread.

4.4 Visão da Engenharia/Infraestrutura

Nesta seção apresenta a arquitetura definida para a implementação do sistema, de acordo com a ideia escolhida no capítulo 2.2.

4.4.1 Arquitetura do sistema

A arquitetura do sistema está representada na figura 22.

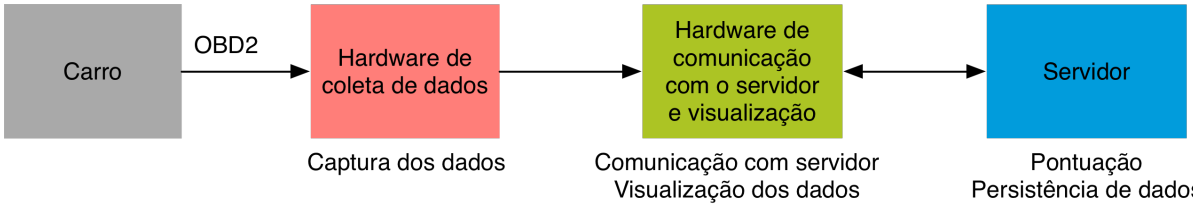

Figura 22 – Arquitetura geral do sistema

Cada componente da arquitetura proposta é descrito a seguir:

Harware de coleta de dados

É o hardware que realiza a interface com o carro, utilizando o protocolo OBD-II. Trata-se da parte responsável por coletar os dados relevantes do automóvel para a detecção do eventos que caracterizam o comportamento imprudente do motorista (especificados na seção 4.3.1.2). Este hardware poderá também utilizar outros sensores embarcados para coletar dados não disponíveis diretamente do padrão OBD-II (ex: aceleração lateral do carro).

Hardware de comunicação com o servidor e visualização

É o hardware que realiza a comunicação com o servidor. Além disso, é responsável pela interface com o usuário, isto é, permite a visualização da rota sendo avaliada, os eventos detectados e a pontuação atribuída ao motorista.

Servidor

Servidor remoto responsável pelo processamento dos eventos identificados e o cálculo da pontuação do motorista na rota percorrida. Além disso, inclui também um banco de dados onde os dados são armazenados para a construção do histórico da pontuação do motorista.

A lógica de detecção de eventos estará distribuída entre o hardware de coleta de dados e o hardware de comunicação com o servidor e visualização, de maneira a minimizar o volume de dados transmitidos de uma parte a outra. Analogamente, os dados enviados ao servidor estarão previamente filtrados pelos componentes precedentes, de forma

a transmitir apenas os dados necessários para o cálculo da pontuação. O objetivo desta abordagem é diminuir a quantidade de dados transmitidos para não atrasar as funcionalidades principais da aplicação, baseando-se no fato de que a transmissão de dados entre os componentes é uma operação que demanda mais tempo para ser concluída.

Por fim, os protocolos utilizados na comunicação entre cada componente serão especificados no capítulo a seguir (4.5), que trata da escolha das tecnologias e padrões adotados para a implementação concreta da arquitetura proposta.

4.5 Visão Tecnológica

Neste capítulo, são apresentadas as tecnologias utilizadas para cada camada do projeto, a partir da arquitetura definida no seção anterior (4.4). Posteriormente, são fornecidas informações básicas sobre cada tecnologia. As principais tecnologias utilizadas foram: Protocolo OBD2, Raspberry Pi, Python, Bluetooth, Android, Node JS e MongoDB.

As tecnologias adotadas estão ilustradas na figura 23:

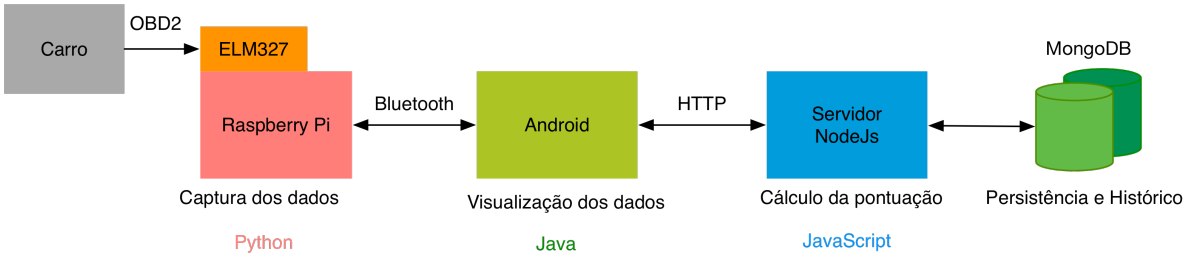

Figura 23 – Arquitetura geral do sistema

Harware de coleta de dados

Implementado por um RaspberryPi conectado a outros sensores embarcados (acelerômetro) e ao circuito ELM327, que possibilita a comunicação utilizando-se o protocolo OBD-II do carro.

Hardware de comunicação com o servidor e visualização

Implementada por um smartphone Android. Os diversos recursos do celular (mapas, internet, GPS, interfaces visuais) poderão ser utilizados para oferecer uma melhor experiência de usuário.

Servidor

Implementado por um servidor NodeJs, conectado a um banco de dados não-relacional MongoDB para a persistência dos dados.

A comunicação entre o hardware de coleta de dados e o hardware de comunicação com o servidor e visualização é realizada via Bluetooth, enquanto a transmissão dos dados ao servidor utiliza o protocolo HTTP, via internet.

As próximas seções descrevem as tecnologias utilizadas em cada componente.

4.5.1 Protocolo OBD-II

O protocolo OBD-II está presente em muitos carros fabricados atualmente. Durante os anos 70 e 80 os fabricantes de carros começaram a utilizar meios eletrônicos para

controlar as funções do carro e como meio de diagnósticos. Este padrão de diagnósticos evoluiu e em no meio dos anos 90 foi introduzido o protocolo OBD-II, que proporciona quase controle total do motor, bem como o monitoramento das partes do carro. (OBD-II Background Information , 2015)

4.5.1.1 Origem do Protocolo

A poluição do ar era um grande problema no estado da Califórnia nos EUA, gerando uma lei que controlava as emissões dos carros no estado no ano de 1966, esta lei tornou-se vigente em todo os EUA em 1968. Para atingir tais emissões foram adotados sistema eletrônicos de controle de ignição e da quantidade de combustível. Foram instalados alguns sensores nos carros que mediam a performance dos motores, de modo a minimizar as emissões. Os valores destes sensores podiam ser acessados por sistemas de diagnóstico preventivo. No início cada fabricante possuía seu próprio padrão para diagnósticos. Em 1988, a SAE (*Society of Automotive Engineers*) criou um conector padrão e um conjunto de sinais de diagnóstico. Com isso, posteriormente surgiu um novo padrão que agrupava o conjunto de padrões e práticas, denominado OBD-II. (OBD-II Background Information , 2015)

4.5.1.2 Adoção e Variações no Protocolo

O protocolo foi colocado como item obrigatório nos carros nos EUA em 1996, já no Brasil carros fabricados a partir 2010 possuem o protocolo. Existem cinco tipos básicos de protocolo em uso, cada um com pequenas diferenças em relacão a comunicação entre o computador de bordo e a ferramenta de diagnóstico. Embora existam variações na forma da comunicação, os comandos são padronizados de acordo com a SAE J1979, norma que estabelece os comandos padrões do protocolo.

4.5.2 ELM327

De modo a ser possível a comunicação do protocolo OBD com computadores e outros dispositivos faz-se necessário o uso de um chip interpretador ELM 327, este converte o padrão OBD para RS-232. O ELM-327 foi desenvolvido para atuar como uma ponte entre os diferentes tipos de comunicação OBD e a interface padrão RS-232. O interpretador ELM possui suporte a nove tipos de protocolo OBD, sendo compatível com a SAE J1979, além de permitir alta taxa de comunicação de dados e ser customizável.

Para comunicar com o interpretador é necessário enviar comandos de forma serial, utilizando o protocolo RS-232. Para obter respostas de valores do carro é necessário escrever um comando requisitando dados e monitorar a porta serial para obter a resposta. No trabalho foram utilizados dois tipos de comando, para obter velocidade e RPM. Para obter estas informações foram utilizados os seguintes comandos:

Velocidade: Utiliza-se o comando 01 0D, que define o modo de mostrar os dados atuais e OD é o código que solicita a velocidade

RPM: Analógo ao comando da velocidade, utiliza-se o comando 01 0C, que define o modo um e OC que requisita o RPM.

As respostas a estes comandos são compostos por duas partes, a primeira indica que a requisição ocorreu de forma correta, acompanhada do código de requisição e então da valor esperado da resposta.

4.5.3 Raspberry pi

A idéia para um computador pequeno e barato para crianças surgiu em 2006, quando Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft, do laboratório de computação da Universidade de Cambridge, ficaram preocupados com o declínio do nível dos estudantes do ensino médio que pleiteavam uma vaga para o curso de ciência da computação. Nos anos 90, os estudantes entrevistados para o curso tinham uma vasta experiência como programadores hobbystas, no entanto nos anos 2000 a situação mudou bastante e os candidatos em sua maioria tinham algum conhecimento em web design.

O jeito como as crianças inglesas lidam com a tecnologia mudou. Alguns problemas foram encontrados: um grande número de aulas utilizando Word e Excel, ou escrevendo páginas web; o fim do crescimento da era ponto-com; e o crescimento dos PC's e consoles de video-game, que substituíram as máquinas que as pessoas das gerações anteriores aprenderem a programar.

Não há muito em que um pequeno grupo possa fazer para solucionar problemas como um currículo inadequado ou o fim de uma bolha financeira. Mas o grupo de Cambridge achou que podia fazer algo para mudar a situação em que os computadores se tornaram custosos e complexos e em que a programação neles teve que ser proibida pelos pais e assim pensaram em uma plataforma, que assim como os computadores pessoais antigos, podiam inicializar em um ambiente de programação. Então de 2006 a 2008, este grupo desenvolveu o que agora se tornou o Raspberry pi.

Em 2008, os processadores desenvolvidos para telefones celulares se tornaram mais acessíveis, e com capacidade de processamento suficiente para prover multimídia, um recurso que poderia deixar a placa atraente para crianças, que não se interessariam por um dispositivo puramente voltado para programação.

Foi então que foi criada a fundação Raspberry pi, para transformar o projeto em realidade. Três anos depois o Raspberry pi modelo B entrou em produção em massa, e em dois anos vendeu mais de dois milhões de unidades.

O objetivo do Raspberry é difundir o uso de computadores de baixo custo, que possam ser utilizados para programação. Sendo uma tentativa de quebrar o paradigma que para ter acesso a internet é necessário comprar um computador de alto custo. Outra meta é a de que o uso dos computadores pessoais seja difundido entre as crianças. (Raspberry Foundation , 2015)

4.5.3.1 Hardware e Periféricos

As entradas e saídas do Raspberry podem ser visualizados na figura 24.

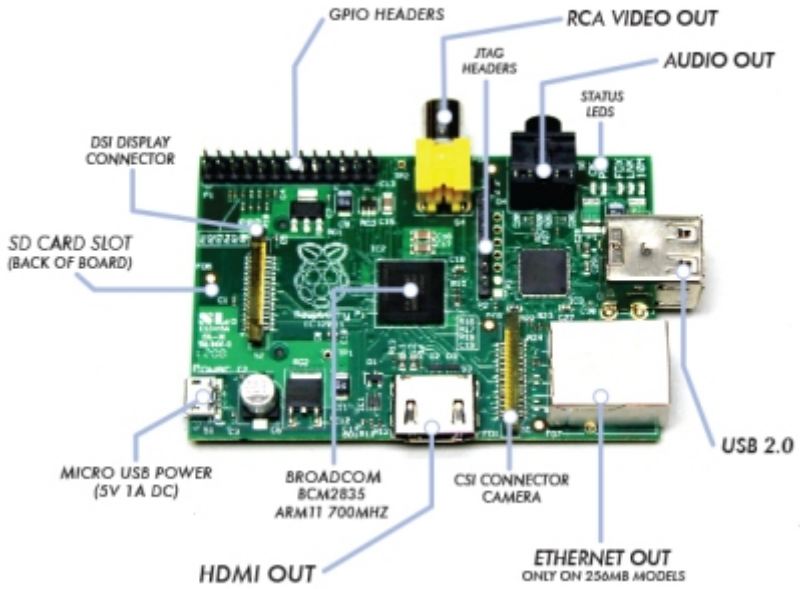

Figura 24 – Entrada e saídas do Raspberry (PCMag.com, 2015)

Como pode ser visto na imagem 24, o Raspberry conta com saídas e entradas similares a de um computador pessoal. Na figura são observadas as principais entradas e saídas:

USB: Utilizado para conectar periféricos USB tais como teclados e mouses;

Ethernet: Permite a conexão da placa à internet.

Micro USB: Fonte de alimentação do Raspberry 5V e 1A.

Entrada SD Card: Cartão onde são carregadas a imagem do sistema operacional Raspbian.

Entradas GPIO: Permitem o acoplamento ao Raspberry de sensores, botões e a comunicação com diversos dispositivos.

Na figura 25 são descritos os pinos GPIO do Raspberry.

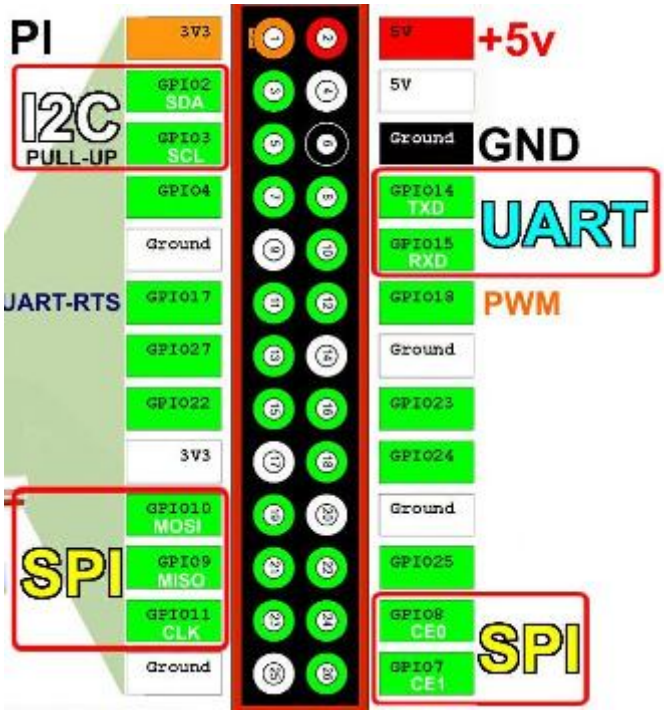

Figura 25 – Pinos GPIO do Raspberry (Super pi Boy, 2015)

Na figura nota-se a presença de vários GPIO's de uso geral, mas podemos destacar os seguintes GPIOS com protocolos utilizados no projeto:

I2C: Protocolo de comunicação utilizado entre o acelerômetro, RTC e o Raspberry. É um protocolo de comunicação serial que permite o acoplamento dos dois dispositivos em paralelo, sendo necessário para a leitura somente o endereço do dispositivo.

UART: Barramento utilizado para comunicação entre o Raspberry e o OBD2.

4.5.4 Python

A linguagem Python é utilizada no projeto dentro do Raspberry para obter dados por meio da interface serial e do I2C, e então enviá-los ao dispositivo Android.

Figura 26 – Logotipo Python (Python.org, 2015b)

4.5.4.1 História

Python foi criada no início dos anos 1990's por Guido van Rossum no Stichting Mathematisch Centrum (CWI) na Holanda, como uma sucessora da linguagem ABC.

Guido permanece como um dos principais autores do Python, entretanto a linguagem recebeu contribuições de outras pessoas.

Em 1995, Guido continuou a desenvolver a linguagem Python no Corporation for National Research Initiatives(CNRI) em Reston, Virginia onde ele lançou diversas versões do software.

Em maio de 2000, Guido e o time principal de desenvolvimento Python mudaram para a BeOpen.com para formar o time BeOpen PythonLabs. Em outubro do mesmo ano, o time mudou-se para a empresa Digital Creations (agora Zope Corporation). Em 2001, foi formada a fundação Python de software, uma organização sem fins lucrativos, criada com o objetivo de ter propriedade sobre as patentes do Python. Zope corporation é uma patrocinadora da fundação Python.

Todas as versões do Python são de código aberto. Sendo a maioria delas compatíveis com GPL. (Python.org, 2015a)

4.5.4.2 Características da Linguagem

Python é uma linguagem simples e minimalista. Um programa em Python se assemelha muito a um texto em inglês. Esse natureza de pseudo-código é um dos grandes pontos positivos desta linguagem. Isso faz com que a concentração se dê na resolução do problema e não na linguagem em si.

Uma outra característica de Python é a sua facilidade de aprendizado, devido a sua simplicidade de sintaxe supracitada.

Python é uma linguagem open-source. Isso significa que cópias deste software podem ser distribuídas livremente, o código fonte da linguagem pode ser lido, modificado, e pode ser usados pedaços deste código em novos programas. Também é uma linguagem constantemente melhorada pela comunidade de usuários.

Python é uma linguagem de alto nível, isto é, não é necessário se ater a detalhes de baixo nível, tais como controle de memória.

A portabilidade é outra característica presente nesta linguagem. Está característica existe devido a sua natureza open-source. Os programas Python podem rodar em várias plataformas, sem ser necessário alterar o código, mas para tal é necessário evitar recursos que somente existem em uma máquina específica.

Python é uma linguagem interpretada, isto é, o código fonte é convertido em uma forma intermediária chamada bytecode e então traduz esta forma para a linguagem de

máquina do computador que rodará o código.

Python suporta o paradigma de orientação a objetos. Assim podem ser criadas classes, de modo a prover reuso e encapsulamento de funções. Em Python a criação de classes é feita de forma simplificada se comparada a linguagens como C++ e Java.

Outra característica desta linguagem é extensilibilidade, isto é, a possibilidade de rodar códigos C++ dentro dos programas em Python.

Por fim uma outra característica é a existência de uma grande biblioteca padrão, sendo assim possível realizar várias tarefas, sem a necessidade de instalar bibliotecas adicionais. (Ibiblio, 2015)

4.5.5 Bluetooth

A comunicação entre o Raspberry e o celular acontece por meio de uma canal Bluetooth, por meio da abertura e conexão de um socket RFCOMM.

Figura 27 – Logotipo Bluetooth (Learn Sparkfun , 2015)

A figura 27 surgiu pela união das runas do alfabeto nórdico correspondentes as letras latinas B e H.

4.5.5.1 História

Harald Blåtand, ou Bluetooth, foi rei da Dinamarca por volta de 940 até cerca 986. Ele é lembrado por converter o país ao catolicismo e por também dar o nome a uma tecnologia de comunicação sem fio entre dispositivos.

O impulsionador desta tecnologia foi Sven Mattison, que formou-se em engenharia em 1979 e começou a trabalhar no departamento de eletrônica aplicada da Universidade de Lund. Ele foi enviado como intercambista para o Instituto de Tecnologia da Califórnia, onde desenvolveu sua tese de doutorado e recebeu seu título em 1986.

Em 1995 ele começou a trabalhar na Ericsson Mobile Communications, onde foi colocado em um projeto juntamente com Jaap Haartsen, onde eles tinham o objetivo de desenvolver um link de rádio entre dispositivos celulares, onde seria possível a comunicação sem fio entre eles. Alguns dos objetivos do projeto era ter baixo alcance e baixa taxa de transmissão. O projeto foi inicialmente chamado de MC Links.

Sven e Jaap perceberam que a Ericsson não deveria desenvolver esta tecnologia sozinha. Em 1997, a Intel assumiu o projeto e Jim Kardach, tornou-se parte do grupo de desenvolvimento. Uma de suas contribuições foi que os MC Links não deveriam ser utilizados para conectar somente celulares, mas também outros tipos de dispositivos.

Também estabeleceram que a tecnologia seria mais útil se fosse um padrão aberto, na frequência de 2.45 GHz. As empresas Nokia, IBM e Toshiba foram convidadas para formar um grupo de desenvolvimento conjunto.

A primeira versão da tecnologia foi lançada em maio de 1998, com o nome de Bluetooth.

Bluetooth teve uma longa utilização, sendo ainda hoje amplamente utilizado. Possuindo várias versões, sendo a mais atual a 4.0, lançada em 2009, mas que passou a ser utilizada somente em 2011.

Bluetooth utiliza uma tecnologia de saltos de frequência, com 1.600 saltos por segundo, reduzindo interferência e perda de sinal, e também permitindo diferentes classes de rádio. Nos celulares, a classe utilizada permite o alcance de cerca de 10 metros, enquanto os módulos de computadores permitem o alcance de 100 metros ou mais. Quase 3000 empresas adotaram o padrão.

Bluetooth funciona equipando cada dispositivo com um chip Bluetooth e uma antena. O chip combina um transmissor e um receptor e escuta sinais de outros chips Bluetooth, enquanto transmite informações contendo sua identidade. Assim que dois dispositivos entram em contato eles iniciam negociações para estabelecer uma conexão.

O padrão Bluetooth é baseado em dez patentes, sendo a maioria delas de propriedade da Ericsson. Segundo Mattisson, isso contribuiu para proteger o padrão aberto e impedir que outros o estragassem. (Ericsson History , 2015)

4.5.5.2 Protocolo RFCOMM

O protocolo RFCOMM é utilizado no projeto para enviar dados do Raspberry para o celular Android. O protocolo emula as configurações e status de uma conexão serial cabeada RS-232 e provê transferência de dados de forma serial. RFCOMM conecta as camadas mais baixas do protocolo Bluetooth utilizando a camada L2CAP. Uma lista de produtos que utilizam RFCOMM são impressoras, modems, computadores, notebooks e etc.

RFCOMM emula portas seriais, com isso pode rodar aplicações que utilizam portas seriais, enquanto também tem suporte ao protocolo OBEX e alguns outros.

O protocolo suporta ate 60 conexões simultâneas entre dois dispositivos Bluetooth. Já o número de conexões que podem ser usadas simultaneamente em um dispositivo Bluetooth depende da implementação adotada.

Para o protocolo RFCOMM, um caminho completo de comunicação envolve duas aplicações rodando em diferentes dispositivos (pontos finais de comunicação) com um segmento de comunicação entre eles.

A figura 28 ilustra um caminho completo de comunicação. Nesta figura o termo aplicação pode referir-se à aplicação destinada ao usuário final, protocolos mais altos de comunicação ou à serviços rodando para fornecer dados às aplicações destinadas ao usuário final.

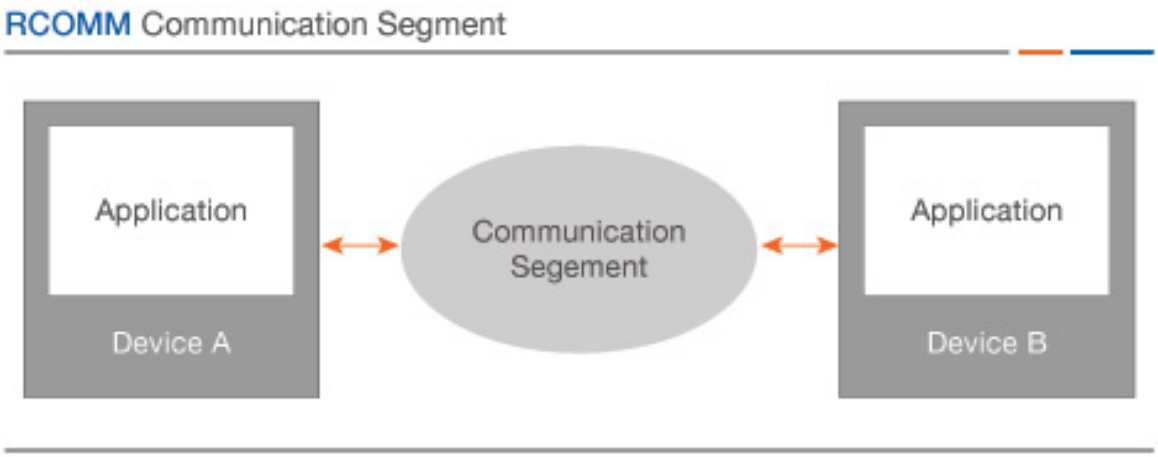

Figura 28 – RFCOMM Caminho Completo de Comunicação (Bluetooth Development Portal, 2015)

RFCOMM tenta suportar aplicações que fazem uso das portas seriais dos dispositivos em que residem. Na configuração mais simples, o segmento de comunicação é um link Bluetooth de um dispositivo para o outro, como pode ser visto na figura 29.

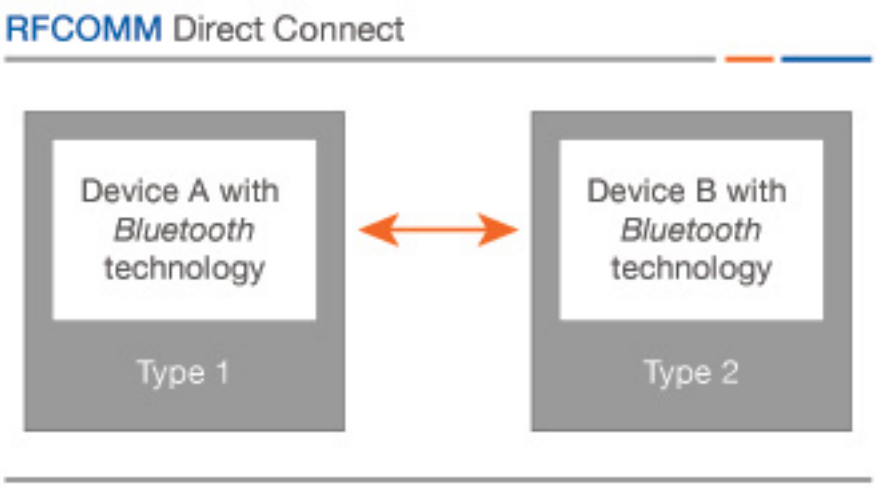

Figura 29 – RFCOMM Conexão Direta (Bluetooth Development Portal, 2015)

Quando o segmento de comunicação é outra rede, a tecnologia Bluetooth wireless é utilizada entre o dispositivo Bluetooth e um dispositivo de conexão à outra rede, como um modem. O protocolo somente se preocupa com a conexão entre os dispositivos, no caso de conexão direta, ou entre o dispositivo e o modem, no caso onde o segmento de comunicação é outra rede.

Existem dois tipos de dispositivos que o protocolo deve suportar. O Tipo 1 são nós finais de comunicação como computadores e impressoras. O tipo 2 são dispositivos que fazem parte do segmento de comunicação, como modems. Entretanto, o protocolo não faz distinção entre estes dois tipos de dispositivos. (Bluetooth Development Portal, 2015)

4.5.6 Android

O projeto utiliza um aplicativo Android para receber os dados do Raspberry e enviá-los ao servidor, também serve como meio de mostrar as penalidades cometidas durante a rota do usuário.

Android é a plataforma móvel mais utilizada no mundo, rodando em milhões de dispositivos móveis em mais de 190 países. Apresenta um grande crescimento no número de usuários.

Android apresenta uma loja de aplicativos aberta para distribuí-los para os usuários, sendo uma grande vantagem deste sistema operacional. A plataforma foi baseada nas contribuições da comunidade Linux de código aberto e de mais de 300 parceiros, incluindo empresas de hardware, software e operadoras de celular, tornando-se o sistema operacional com maior crescimento.

Todos os dias mais de um milhão de dispositivos Android são ativados pelo mundo.

O fato da plataforma ser em código aberto contribui para ela ser a escolhida por consumidores e desenvolvedores, gerando grande crescimento no consumo de aplicativos. Os usuários baixam bilhões de aplicativos e jogos da Google Play por mês.

Juntamente com seus parceiros, Android está movendo continuamente as fronteiras de hardware e software, de modo a trazer novas funcionalidade para os usuários e desenvolvedores. Para os desenvolvedores, as inovações permitem construir aplicações que utilizam as últimas tecnologias móveis. (Android Developers, 2015)

Na figura 30 são mostradas as diferentes versões da plataforma Android.

Figura 30 – Diferentes Versões do Android ao longo do Tempo (IbnLive, 2015)

4.5.7 Java

O desenvolvimento de aplicativos Android é realizado utilizando-se Java, linguagem de programação orientada a objetos.

Figura 31 – Logo da linguagem Java

4.5.7.1 História

Java foi iniciado em junho de 2001, em um projeto liderado por James Gosling, Mike Sheridan e Patrick Naughton, sendo originalmente projetado para televisão interativa. Inicialmente, a linguagem era chamada Oak, referência ao carvalho (oak) plantado no exterior do escritório de Gosling. Mais tarde, a linguagem foi chamada de Green e, por fim, Java.

A primeira implementação pública do Java (Java 1.0) foi lançada em 1995, pela Sun Microsystems. Nesta época, os principais navegadores incorporaram a execução de Java applets nas páginas web. Posteriormente, com o lançamento da versão Java 2, aumentou-se o número de plataformas e configurações compatíveis. A versão destinada à Desktops foi denominada J2SE (Java Standard Edition), enquanto as aplicações de empresas eram desenvolvidas utilizando-se a tecnologia J2EE (Java Enterprise Edition). Por sua vez, a versão J2ME (Java Mobile Edition) oferecia recursos especificamente para aplicações de celulares. Posteriormente, por motivos de marketing, as versões foram renomeadas Java SE, Java EE e Java ME.

Atualmente, o Java SE encontra-se na versão 8.0, e é considerada ainda como uma das mais utilizadas linguagens de programação.

4.5.7.2 Características da linguagem

O código Java é compilado em um bytecode, que é executado na máquina virtual java (JVM - Java Virtual Machine), independentemente da arquitetura da máquina. Possui gestão automática de memória, com uma implementação padrão de "Garbage Collector" que monitora as referências aos objetos em uma aplicação.

Graças à grande popularidade da linguagem, existem inúmeras bibliotecas e frameworks disponíveis atualmente para o desenvolvimento de aplicações em Java. Dentre as funcionalidades oferecidas pelas bibliotecas mais utilizadas, destacam-se: IO (Input/Output), Networking (comunicação em rede), Concurrency (programação paralela), segurança, interfaces gráficas, etc.

Com o advento dos smartphones, o Google escolheu o Java como linguagem de programação para o desenvolvimento de aplicativos Android. Assim, o Android SDK (Standard Development Kit) utiliza o Java como base das aplicações.

4.5.8 Protocolo HTTP

A aplicação Android utiliza o protocolo HTTP para realizar requisições ao servidor NodeJs pela internet.

4.5.8.1 Características

HTTP (HyperText Transfer Protocol) é um protocolo da camada de aplicação para comunicação via internet. Trata-se de um protocolo do tipo requisição-resposta, baseado no modelo de computação cliente-servidor, no qual o cliente envia requisições ao servidor que, por sua vez, retorna uma reposta. Por exemplo, um navegador web é um cliente, enquanto um outro computador hospedando uma página web é o servidor.

No contexto do modelo de camadas de redes, HTTP é um protocolo da camada de aplicação e é comumente utilizado em conjunto do TCP (Transmission Control Protocol), protocolo da camada de transporte.

Dentre os métodos de requisições mais comuns, destacam-se:

GET

Requisita uma representação de um determinado recurso (uma página da internet, por exemplo). Requisições utilizando GET devem apenas recuperar dados e não ter outro efeito.

POST

Requisita que o servidor web aceite e salve os dados enviados no corpo (body) da mensagem enviada. É frequentemente utilizado ao fazer uploads de arquivos ou submeter formulários. Neste projeto, requisições do tipo POST são utilizadas para enviar e salvar os dados referentes às rotas no servidor.

4.5.9 NodeJs

O servidor responsável pelo cálculo da pontuação e da persistência dos dados do sistema é implementado utilizando-se a tecnologia NodeJs, um framework JavaScript para o desenvolvimento de aplicações no servidor.

Figura 32 – Logo do NodeJs

4.5.9.1 História

NodeJs foi criado inicialmente por Ryan Dahl em 2009, época em que trabalhava na empresa Joyent. Primeiramente, a tecnologia foi orientada ao uso por sistemas Linux.

Dahl inspirou-se na barra de progresso do Flickr: o browser não tinha capacidades para saber qual volume do arquivo já havia sido enviado e tinha que consultar o servidor a cada intervalo de tempo. No mesmo ano, o projeto foi apresentado na conferência European JSConf, onde recebeu uma ótima recepção pelo público.

Em 2011, o gerenciador de pacotes "npm" foi adicionado à tecnologia, facilitando a extensão do framework ao simplificar a maneira de instalar novos módulos e mantê-los atualizados.

Atualmente, NodeJs é uma das principais escolhas no desenvolvimento de servidores utilizando a linguagem de programação JavaScript.

4.5.9.2 Características

NodeJs executa um único thread, utilizando chamadas de IO que não bloqueiam o programa, o que permite a concorrência de dezenas de milhares de conexões sem o custo de trocar de thread.

A tecnologia possui um gerenciador de pacotes chamado "npm", que facilita a instalação, atualização e gerenciamento de módulos desenvolvidos para o framework. Neste projeto, por exemplo, utilizou-se o pacote "Mongoose, que possibilita a conexão entre a aplicação NodeJs e um banco de dados MongoDB.

4.5.10 JavaScript

A linguagem de programação utilizada no desenvolvimento de aplicações em NodeJs é o Javascript.

4.5.10.1 História

JavaScript foi desenvolvido originalmente em 10 dias por Brendan Eich, em 1995, para ser utilizado no browser Netscape. Seu primeiro nome foi "Mocha", seguido de "LiveScript" quando foi lançado nas versões beta do navegador Netscape 2.0. Posteriormente, foi renomeado JavaScript, ao ser integrado à versão 2.0B3 do navegador.

Assim, a linguagem foi sempre muito utilizada no desenvolvimento de aplicações web mais ricas e dinâmicas do lado do cliente, pois possibilitava a execução de lógica e scripts.

Atualmente, com o crescente uso da tecnologia NodeJs, a linguagem vem sendo cada vez mais utilizada também no desenvolvimento do lado do servidor das aplicações modernas.

4.5.10.2 Características da linguagem

JavaScript suporta a maioria da sintaxe de programação estruturada da linguagem C (por exemplo, if else, while, switch, etc). Contudo, inicialmente só possuía variáveis de escopo de função (utilizando a palavra reservada "var"). Posteriormente, introduziu-se a palavra reservada "let", que pode ser utilizada para declarar variáveis com escopo de bloco.

A linguagem utiliza tipagem dinâmica, isto é, as variáveis não precisam ser declaradas com um tipo/classe fixo. Assim, uma variável originalmente do tipo "inteiro" pode ser utilizada para guardar um valor do tipo "booleano".

Suporta a criação de objetos e classes. Além disso, funções podem ser alocadas em variáveis, como se fossem objetos, o que facilita na programação e tratamento dinâmico de condições.

4.5.11 MongoDB

O banco de dados escolhido para a persistência dos dados relevantes do sistema é o MongoDB, banco de dados não SQL orientado à documentos.

Figura 33 – Logo do MongoDB

4.5.11.1 Características

MongoDB é uma tecnologia multi-plataforma de banco de dados orientado a documentos. É classificado como um banco de dados não SQL (Structured Query Language), não relacional, isto é, que não se baseia em tabelas para a manipulação dos dados. Assim, MongoDB utiliza documentos com estrutura muito similar ao formato de dados JSON (JavaScript Object Notation), cujo formato se donomina BSON (Binary JSON).

Os principais conceitos utilizados na criação de banco de dados no MongoDB são: coleções e documentos. Fazendo-se um paralelo entre banco de dados relacionais e os principais conceitos, uma coleção representa uma tabela do mundo relacional, enquanto um documento representa uma linha na tabela (isto é, uma instância daquele tipo de dado).

A integração desta tecnologia com NodeJs é feita de maneira muito prática, bastando-se instalar (por meio do npm) um módulo/pacote específico do NodeJs que realiza as

conexões com o banco de dados MongoDB.

5 Métodos

Este capítulo apresenta o método de trabalho adotado para a execução das atividades do projeto e divide-se em três partes: organização de tarefas, programas e IDEs utilizados.

5.1 Organização das Tarefas

Esta seção apresenta a metodologia adotada para a organização das tarefas, bem como as ferramentas e softwares utilizados para o desenvolvimento do código e testes.

5.1.1 Trello

Trello é um programa baseado nos quadros Kanbam da metodologia ágil. É uma ferramenta de organização de projetos extremamente versátil, que permite um trabalho colaborativo mais eficiente ao possiblitar cada integrante visualizar o que cada um está fazendo no projeto e o estado atual das tarefas. No Trello, pode-se criar "boards" que agrupam listas ("lists") de tarefas que devem ser feitas. Cada tarefa é então representada por um "card", criado dentro de uma "list". As "lists" criadas dentro da "board" do projeto são:

Backlog: Tarefas que ainda precisam ser analisadas e aprovadas para serem colocadas na fase de atividades.

To Do: Tarefas a serem realizadas, que devem ser alocadas a um integrante da equipe.

Doing: Tarefas em execução, ou seja, alocadas a um integrante e já iniciada.

Done: Tarefas terminadas.

A figura 34 apresenta a tela principal do programa Trello, onde podem ser vistas as "lists" descritas acima.

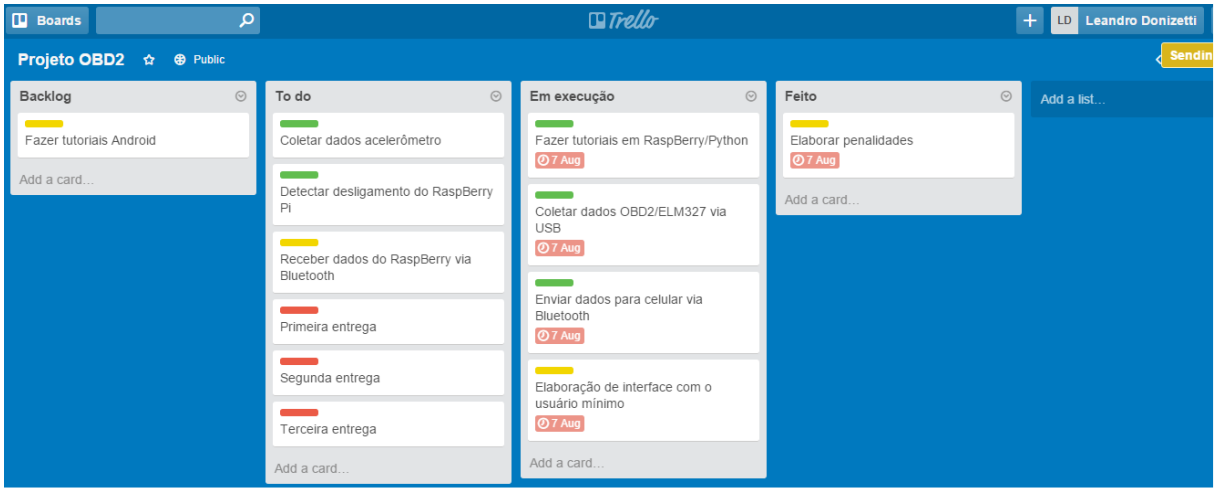

Figura 34 – Imagem ilustrativa de uma Board do Trello

5.2 Programas e IDEs

Esta seção apresenta os principais programas e IDEs (Integrated Development Environment) utilizados durante o desenvolvimento de cada componente do sistema proposto.

5.2.1 Raspberry Pi

Para escrever os códigos Python no windows foi utilizado o programa Sublime Text 2, cuja tela principal pode ser visualizada na imagem 35.

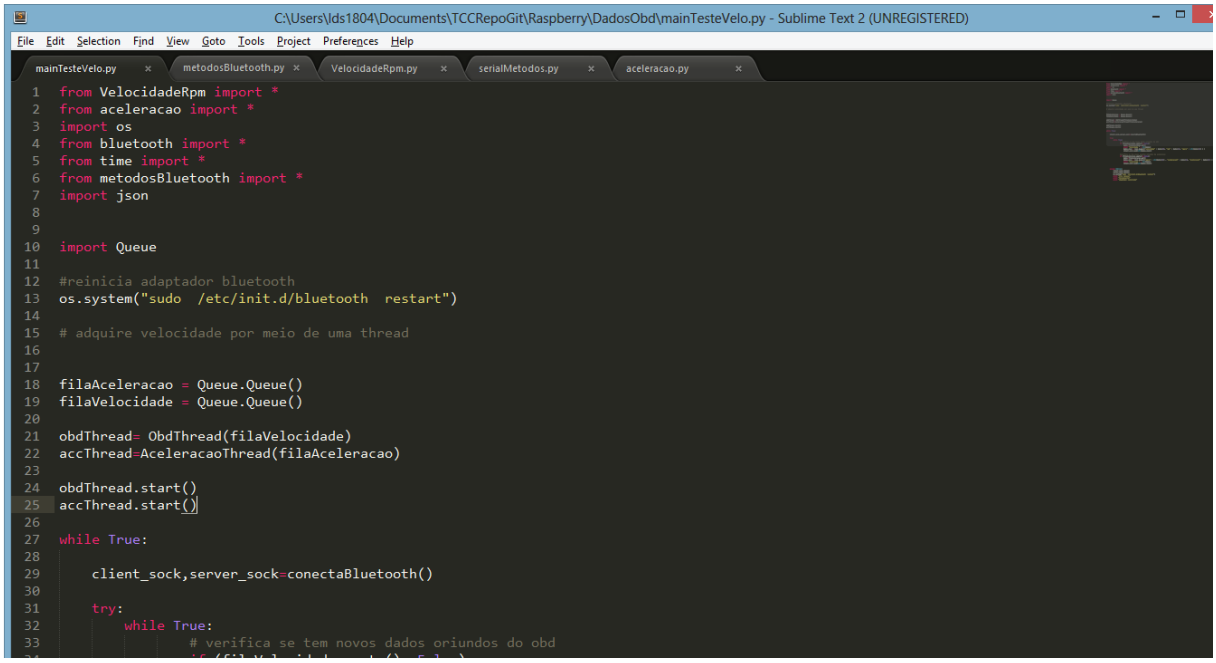

Figura 35 – Interface do programa Sublime Text 2

Para enviar o código escrito no Windows ao Raspberry Pi e mantê-los sincronizados, utilizou-se o programa WinSCP, o qual mantém dois diretórios remotos sincronizados. Assim, as mudanças ocorridas no diretório local ocorrem também no diretório remoto.

Para a visualização da tela do Raspberry, foi utilizado o programa Real VNC viewer, cliente VNC (Virtual Networking Computing) que se conecta ao servidor instalado no Raspberry.

Para executar os scripts no Raspberry foi utilizado o programa Geany.

5.2.2 Aplicativo Android

O aplicativo foi feito utilizando o programa Android Studio, baseado na IDE de desenvolvimento Java IntelliJ. O Android Studio é a IDE recomendada pelo Google para o desenvolvimento de aplicativos Android, que permite elaborar o aplicativo e verificar os seus erros. Também permite executar o aplicativo em um dispositivo real ou em um simulador (máquina virtual).

Uma tela de exemplo do programa pode ser vista na figura 36.

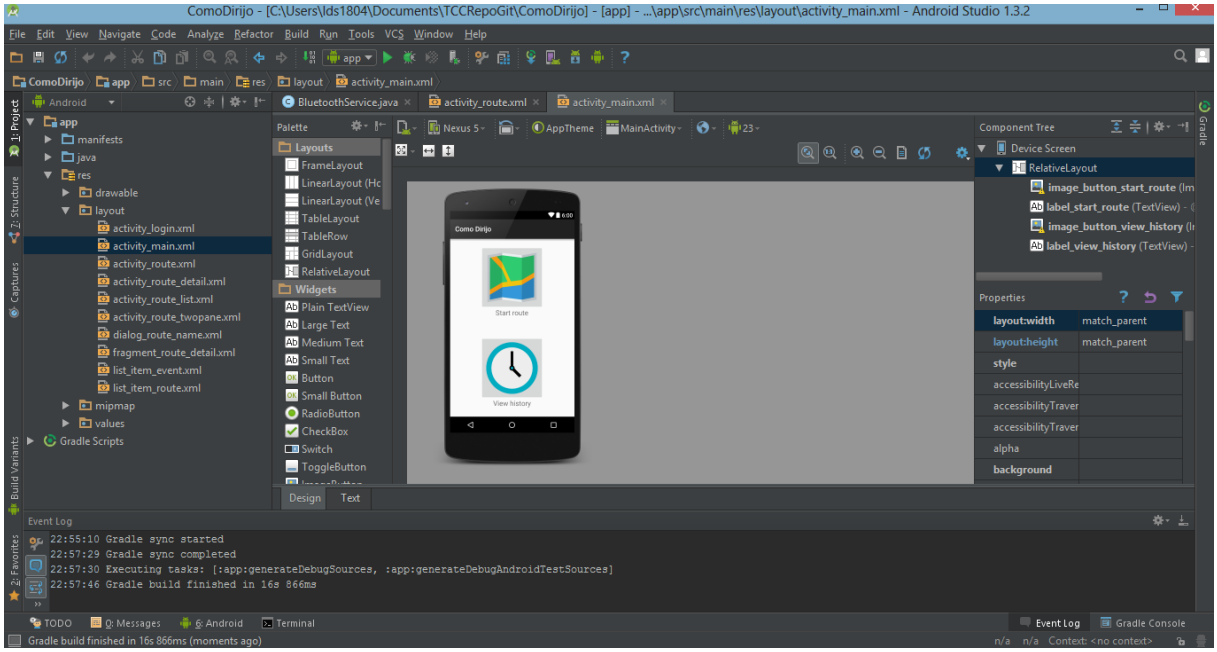

Figura 36 – Interface do programa Android Studio 1.3.2

5.2.3 NodeJs

O código JavaScript do servidor NodeJS foi escrito utilizando-se o editor de texto Atom, programa de código aberto mantido pelo GitHub. Neste editor, pode-se instalar inúmeros pacotes e extensões que auxiliam no desenvolvimento de uma aplicação em determinada tecnologia ou linguagem. Assim, foram instalados pacotes que facilitavam o desenvolvimento em JavaScript e, mais especificamente, em NodeJs.

A figura 37 ilustra a interface do Atom.

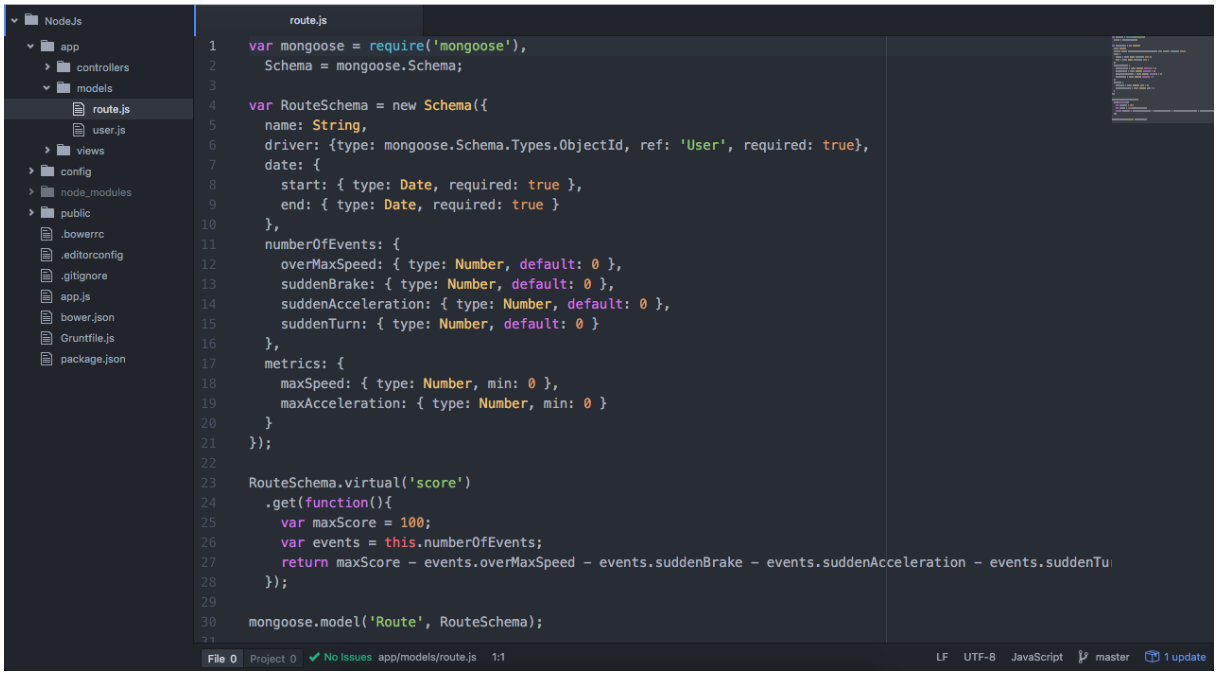

Figura 37 – Interface do editor de texto Atom

6 Resultados e Análise

Este capítulo apresenta os resultados atingidos pela equipe, obtidos a partir da especificação do sistema e da metodologia de trabalho adotada.

O capítulo divide-se em cinco partes: Raspberry Pi, aplicativo Android, servidor NodeJs, banco de dados MongoDB e integração, apresentando os resultados de cada componente do sistema.

6.1 Raspberry Pi

Esta seção apresenta os resultados obtidos no coletor de dados implementado pelo Raspberry Pi.

6.1.1 Coleta de Dados OBD

Para simular a coleta de dados, foi utilizado um simulador de geração de dados do padrão OBD-II para Microsoft Windows. O simulador fornece dados de velocidade, RPM, temperatura do motor e outras informações, cujos valores podem ser variados utilizando a interface gráfica do programa.

A interface de usuário principal do simulador é ilustrada na figura 38.

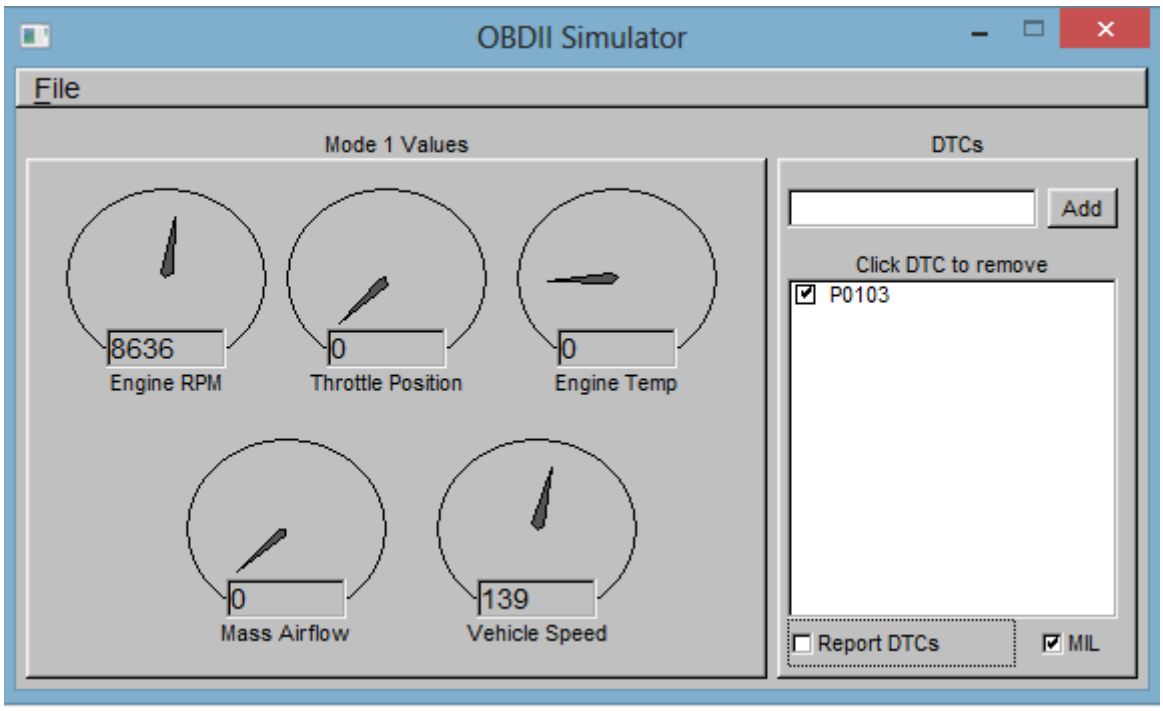

Figura 38 – Simulador OBD-II para Windows

Na figura, o simulador está configurado para fornecer a velocidade de 139 km/h e 8636 de RPM.

Inicialmente, foi realizado o teste da conexão serial entre o Raspberry e o Windows. Para isto, foi escrito um programa simples (em Python, executado pelo Raspberry) que escrevia no serial. Então, os dados escritos pelo Raspberry foram lidos no Windows por meio de um monitor serial.

Este teste se encontra ilustrado na figura 39.

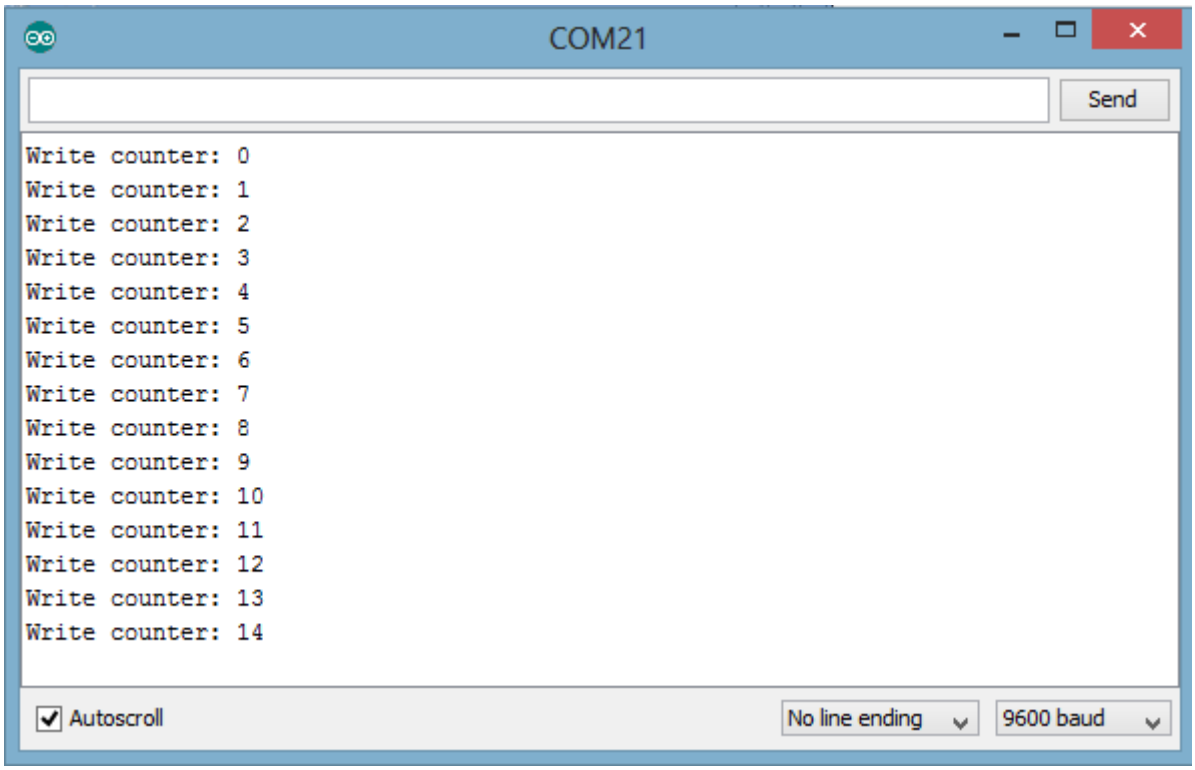

Figura 39 – Teste de escrita no serial do Windows

No teste, o programa Python escreve um contador no serial, que então é lido por um programa monitor no Windows.

Posteriormente, foi realizado o teste de escrever requisições no serial para o OBD e verificar estas respostas. Nestes testes, foi escrita a requisição para obter a velocidade, conforme mostrado na figura 40.

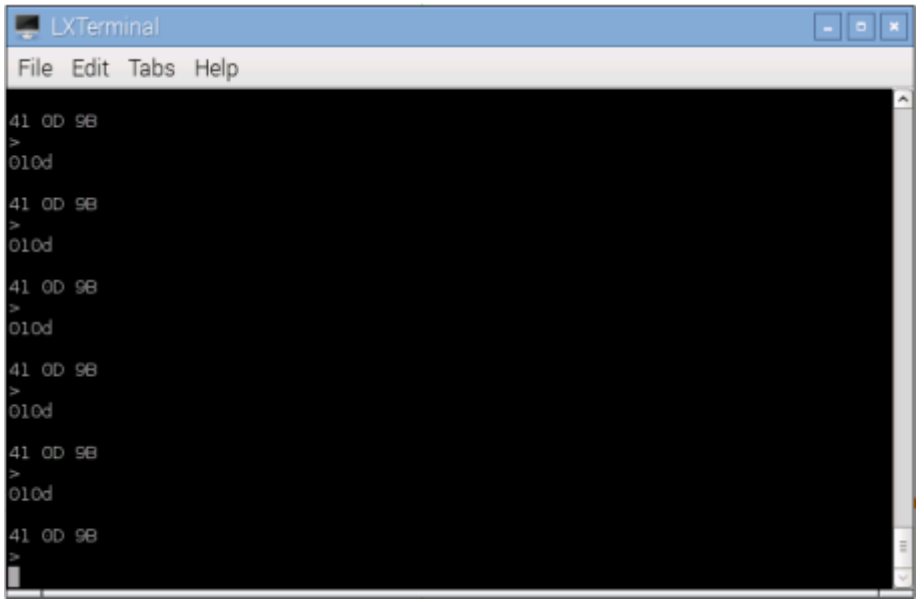

Figura 40 – Teste de Obtenção de Dados do OBD-II

Neste teste, o valor “010D” é escrito no serial. Então, o valor de “9B” é obtido como resposta, correspondendo a “155” em decimal, bem próximo do valor exibido pelo simulador, que é de “156”.

6.1.2 Teste Acelerômetro

Para testar o funcionamento do acelerômetro ADXL345, foi utilizado um script em Python que importa uma biblioteca e solicita valores em m/s^2, imprimindo-as no console Python.

O valor de uma leitura é mostrado na figura 41.

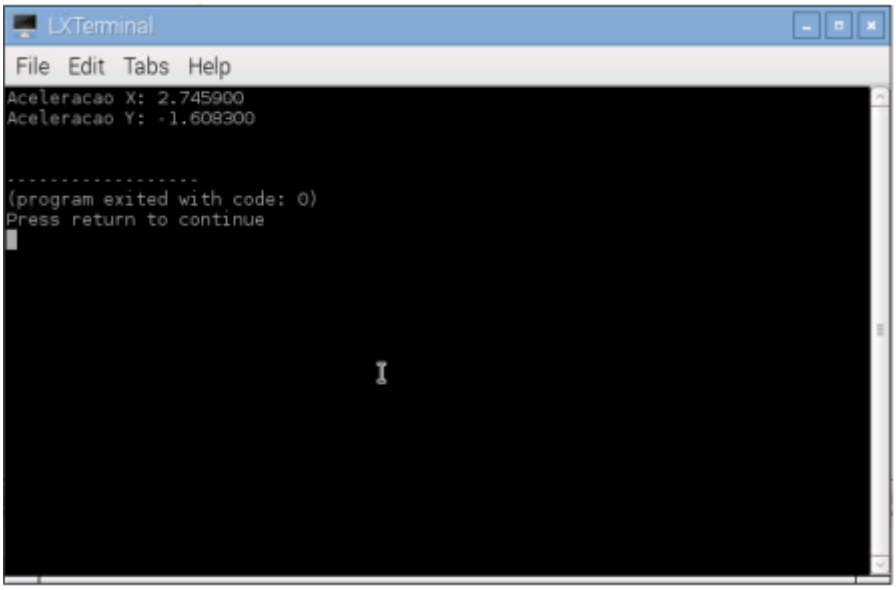

Figura 41 – Teste de Obtenção da Aceleração do Sensor ADXL345

Os valores mostrados correspondem ao acelerômetro em repouso. Eles são diferentes de zero pois a aceleração da gravidade é decomposta nos outros eixos.

6.1.3 Teste Conexão RFCOMM Bluetooth

No teste da conexão Bluetooth, foi criado um servidor RFCOMM em Python no Raspberry. O servidor cria um socket de comunicação e o deixa aberto para conexões.

A figura 42 ilustra o servidor RFCOMM com uma conexão aberta na porta 1.

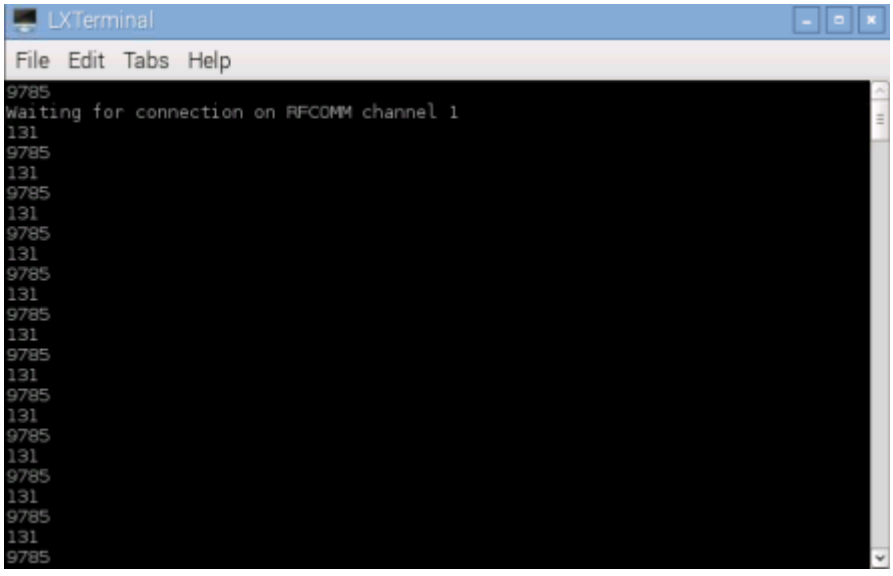

Figura 42 – Conexão do Servidor RFCOMM aberta na porta 1

Abaixo da mensagem informando que o servidor está esperando por conexões, encontram-se os valores captados pelo OBD-II.

Após a abertura da conexão, é criado um cliente RFCOMM no aplicativo Android que, por sua vez, solicita a conexão ao servidor RFCOMM do Raspberry. Este então imprime uma mensagem informando que a conexão foi aceita e mostra os dados do dispositivo conectado.

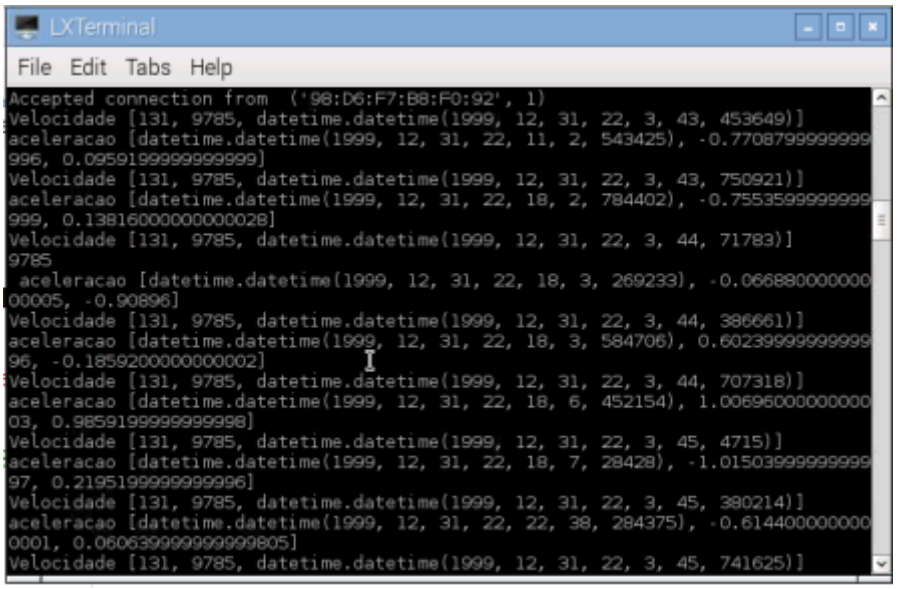

Figura 43 – Conexão aceita pelo Servidor RFCOMM na porta 1

Isto pode ser visto na primeira linha da figura 43, onde é impresso o endereço MAC do dispositivo, bem como a porta em que a conexão foi feita.

6.2 Aplicativo Android: Como Dirijo

Esta seção apresenta os resultados obtidos no aplicativo Android (nomeado "Como Dirijo"), cujo ícone pode ser visualizado na figura 44. A instalação do aplicativo é realizada diretamente pelas ferramentas do Android Studio.

Figura 44 – Ícone do aplicativo Como Dirijo

6.2.1 Interface de Usuário

Esta seção apresenta as principais interfaces de usuário do aplicativo.

6.2.1.1 Tela principal

A tela principal do aplicativo possui duas opções: uma para iniciar o registro de uma rota ("Start route") e outra para visualizar o histórico de rotas do usuário ("View History"). A interface pode ser vista na figura 45.

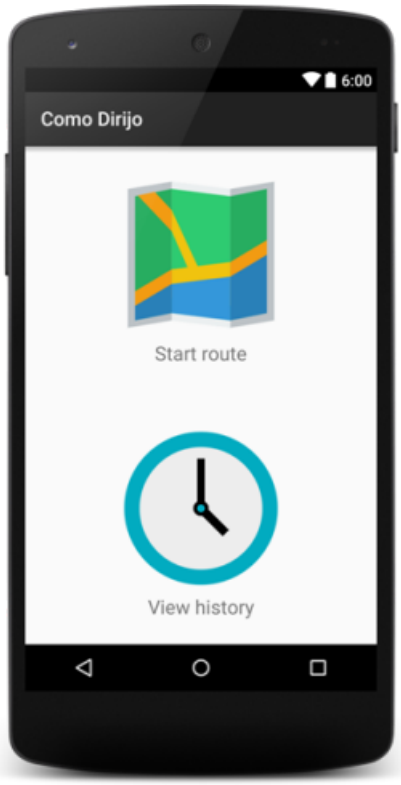

Figura 45 – Layout da tela principal do aplicativo Como Dirijo

As interfaces acessíveis por estas duas opções são descritas nas próximas seções.

6.2.1.2 Visualização de infrações em tempo real

Ao selecionar a opção "Start route" para iniciar uma rota, um mapa é mostrado com o percurso sendo percorrido pelo carro. A cada infração cometida, um "Toast" (notificação) é apresentado ao usuário, alertando-o sobre o seu comportamento imprudente. Além disso, um marcador é inserido no mapa mostrando o ponto da rota onde ela ocorreu. Posteriormente, o usuário pode clicar sobre este marcador para visualizar os detalhes da infração cometida.

A interface de visualização de infrações em tempo real pode ser visualizada na figura 46.

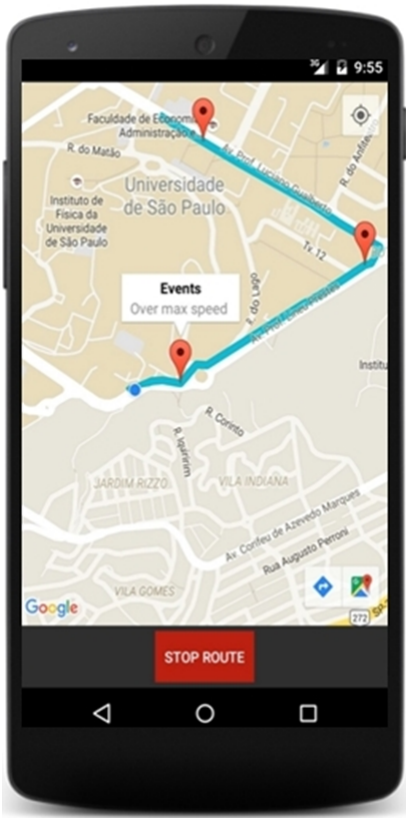

Figura 46 – Layout da tela de visualização de infrações em tempo real

Por exemplo, como pode ser visto na figura, foram inseridos três marcadores de excesso de velocidade.

Em um primeiro teste do sistema de detecção de infrações no Android, foi utilizado o simulador de OBD-II para Windows e foi feita uma rota dentro da cidade universitária, onde foi testada a funcionalidade de desenhar rota, bem como a detecção de valores de velocidade acima do permitido.

As velocidades eram obtidas do simulador e, caso fosse ultrapassada a velocidade máxima, um toast (notificação) era mostrado ao usuário e um marcador era adicionado ao local onde ocorreu a infração. Ao clicar no marcador, pode-se obter detalhes do tipo de infração cometida.

O limite de velocidade foi fixado em 100 Km/h e os limiares das acelerações foram definidas seguindo os valores de quase colisão do estudo 100'Car Naturalistic Driving Study (DINGUS et al., 2006).

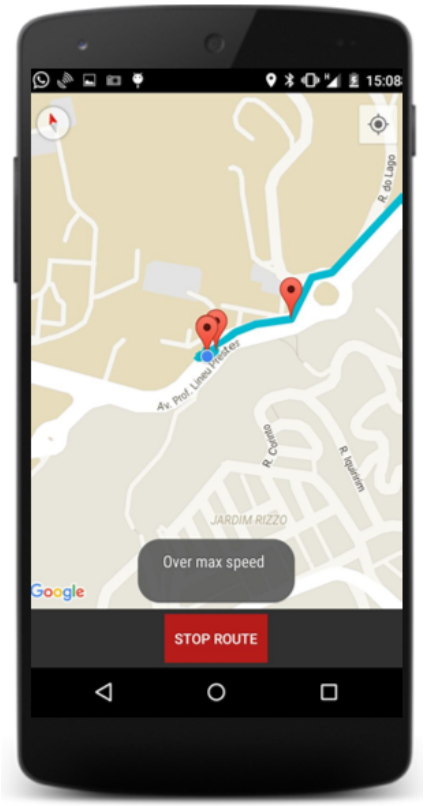

Figura 47 – Rota com alguns marcadores e exibição de um “Toast”

Na tela pode ser vistos três marcadores de excesso de velocidade e um toast com o texto “over max speed”, indicando que uma infração acabou de ocorrer.

6.2.1.3 Visualização do histórico

Ao selecionar a opção “View history” para visualizar o histórico de rotas, uma lista de todas as rotas salvas pelo usuário é apresentada. Um exemplo desta tela é mostrada no layout abaixo, representado pela figura 48.

Figura 48 – Layout da tela de visualização do histórico de rotas

No exemplo, são mostradas duas rotas, onde são apresentadas os nomes das rotas e as datas. Pode-se então clicar sobre cada uma destas rotas para visualizar a pontuação e infrações cometidas. Ao clicar no botão "<-" no canto superior esquerdo, retorna-se à tela inicial.

6.2.1.4 Visualização da pontuação e detalhes da rota

Ao final do registro de uma nova rota ou ao selecionar uma rota salva no histórico, uma tela com a pontuação da rota é apresentada ao motorista. Nela, são discriminadas as infrações cometidas e a nota final obtida, como pode ser visto na figura 49.

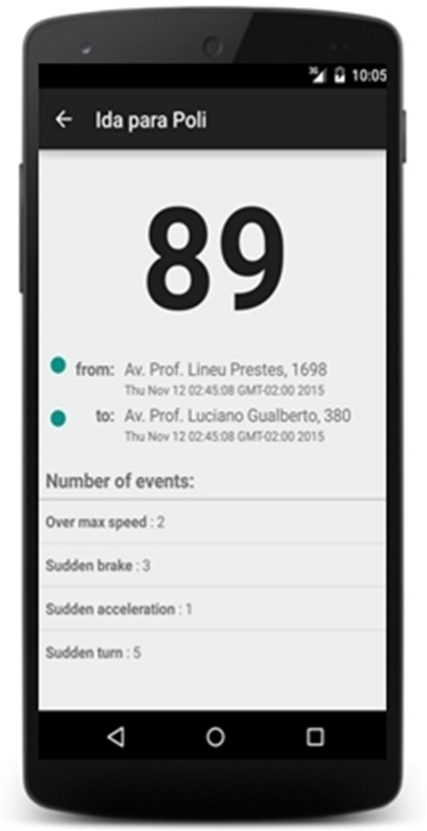

Figura 49 – Layout da tela de visualização de pontuações

As informações que podem ser visualizadas nesta interface são:

- Pontuação
- Localização de origem e destino
- Data de origem e destino
- Infrações cometidas (tipo e quantidade)

Ao clicar no botão "<-" no canto superior esquerdo, retorna-se à tela de lista do histórico.

6.2.2 Conexão com Raspberry Pi

Esta seção é semelhante ao teste de conexão Bluetooth do Raspberry, com a ressalva de mostrar as telas de log no lado Android, de modo a assegurar que a conexão é feita e que os dados estão sendo recebidos corretamente.

Para cumprir a tarefa de conexão no lado Android, foram criadas duas threads: uma responsável por iniciar a conexão RFCOMM e outra por receber os dados e armazená-los em uma lista de Strings JSON, que é passada para funções que tratam estes dados. O

uso de threads dedicadas à comunicação foi necessário porque, caso contrário, a interface do usuário ficaria menos responsiva durante a conexão e transmissão dos dados.

Para testar a conexão, inseriu-se um ponto de controle na thread responsável por conectar-se ao Bluetooth, que gera uma exceção caso não seja possível conectar. Desta maneira, é possível determinar se ocorreu ou não a conexão. Durante os testes, foi assumido que o Bluetooth está ligado e que os dispositivos estão pareados.

Para o teste da transmissão, verificou-se por meio dos logs quais dados eram recebidos. Um breakpoint foi utilizado na thread que recebe os dados para verificar quais valores estão chegando ao Android e se correspondem ao dados captados pelo Raspberry.

6.2.3 Conexão com servidor

A conexão com o servidor foi realizada por meio de requisições HTTP realizadas pelo aplicativo Android (cliente) ao servidor NodeJs. Para isto, implementou-se a classe "RequestTask", que realiza requisições HTTP a uma rota especificada utilizando-se o método HTTP POST. A classe "RequestTask" é filha da classe "AsyncTask".

A classe "AsyncTask", nativa da plataforma Android, permite a execução de uma tarefa assíncrona, isto é, uma tarefa que roda paralelamente a thread principal do aplicativo Android (denominada também de "UI thread", pois é responsável por processar as operações de interface com o usuário). Além disso, esta classe possibilita a fácil comunicação entre nova thread criada e a thread principal, o que é utilizado para apresentar os dados recuperados do servidor no aplicativo.

A classe "RequestTask" implementa um construtor específico e dois métodos principais de sua classe mãe "AsyncTask", os quais estão listados a seguir:

RequestTask(context: Context, route: String)

Construtor da classe "RequestTask", o qual recebe dois parâmetros: o contexto da aplicação (a "Activity" atual, isto é, a classe da plataforma Android que representa a tela em exibição ao usuário) e a rota para a qual deve ser feita a requisição HTTP. Os parâmetros são então guardados em atributos privados da classe para serem utilizados no momento do envio da requisição ao servidor.

doInBackground(params: String[ ]): String

Método responsável por executar a requisição HTTP ao servidor, cuja rota fora especificada na construção do objeto. O parâmetro "params" é um vetor de Strings cujo primeiro elemento (params[0]) é um objeto no formato JSON com os dados a serem enviados via método POST na requisição HTTP.

onPostExecute(result: String): void

Método executado no término da requisição HTTP. A resposta do servidor é passada

ao método pelo parâmetro “result”. Este método é reescrito (“Override”) para cada requisição, implementando o tratamento adequado nos dados recebidos. Se este método não é reescrito, seu comportamento padrão (simplesmente mostrar a resposta obtida, utilizando um Toast) será executado.

Os objetos no formato JSON enviados e recebidos são serializados/deserializados utilizando-se a biblioteca Java GSON, desenvolvida pelo Google. Assim, implementou-se duas classes para serializar e deserializar os objetos do tipo “Route” (rota) do aplicativo, denominadas “RouteSerializer” e “RouteDeserializer”. Basicamente, ambas especificam um mapeamento entre atributos dos objetos do tipo “Route” e do objeto JSON (objeto serializado).

6.3 Servidor NodeJs

Esta seção apresenta o detalhamento da implementação do servidor em NodeJs.

6.3.1 Acesso

Para permitir o acesso ao servidor, utilizou-se o módulo “express” da plataforma NodeJs, que possibilita a criação de um servidor web (utilizando o protocolo HTTP), com a especificação de rotas (URL’s) e o tratamento que deve ser feito no servidor para cada tipo de requisição.

A seguir, as rotas criadas com suas respectivas funções, de acordo com a especificação do projeto:

GET /user

Caminho utilizado para a renderização da página web que lista os usuários cadastrados no sistema, bem como oferece opções de criação de supressão de usuários. Esta página foi criada para permitir uma rápida verificação de todos os usuário salvos na aplicação.

A figura 52 ilustra esta página auxiliar.

Figura 50 – Página auxiliar de verificação dos usuários cadastrados

POST /userCreate

Caminho utilizado para criação de usuários a partir da página de verificação de usuários cadastrados.

Mensagem: objeto JSON com as informações referentes ao usuário, de acordo com o formato indicado pela figura 51:

Figura 51 – Formato do objeto JSON enviado para a criação de usuários

Resposta: código de status 200 e mensagem de sucesso.

POST /userDelete

Caminho utilizado para apagar um usuário do sistema, acessível a partir da página auxiliar de verificação de usuários cadastrados.

Mensagem: objeto JSON com um campo "id" cujo valor é o id do usuário a ser apagado.

Resposta: redireciona o usuário à página auxiliar de verificação de usuários cadastrados.

GET /route

Caminho utilizado para a renderização da página web que lista as rotas salvas por todos os usuários, além de oferecer opções para criação e supressão de rotas. Esta página foi criada para verificar se os dados enviados pelo aplicativo estavam realmente sendo persistidos no banco de dados, permitindo-se uma rápida visualização das rotas existentes no banco de dados.

A figura 52 ilustra esta página auxiliar.

Routes List

Name	Driver	Start Date	End Date	Score	Actions
Ida para casa	Danilo Shibata	Thu Nov 12 2015 00:44:33 GMT-0200 (BRST)	Thu Nov 12 2015 00:44:33 GMT-0200 (BRST)	65	✖
Ida para Poli	Danilo Shibata	Thu Nov 12 2015 00:45:08 GMT-0200 (BRST)	Thu Nov 12 2015 00:45:08 GMT-0200 (BRST)	89	✖

Figura 52 – Página auxiliar de verificação das rotas criadas

POST /routeCreateApp

Caminho utilizado para criação de rotas a partir do aplicativo Android.

Mensagem: objeto JSON com as informações referentes à rota, de acordo com o formato indicado pela figura 53:

```
{
  "name": String,
  "driver": String,
  "date": {
    "start": Date,
    "end": Date
  },
  "numberOfEvents": {
    "overMaxSpeed": Number,
    "suddenBrake": Number,
    "suddenAcceleration": Number,
    "suddenTurn": Number
  },
  "metrics": {
    "maxSpeed": Number,
    "maxAcceleration": Number
  }
}
```

Figura 53 – Formato do objeto JSON enviado para a criação de rotas

Resposta: código de status 200 e mensagem de sucesso.

POST /routeDelete

Caminho utilizado para apagar uma rota do sistema, acessível a partir da página auxiliar de verificação de rotas criadas.

Mensagem: objeto JSON com um campo "id" cujo valor é o id da rota a ser apagada.

Resposta: redireciona o usuário à página auxiliar de verificação de rotas criadas.

POST /userRoutes

Caminho utilizado para acessar uma lista com todas as rotas salvas na base de dados pertencentes a um determinado usuário. É utilizado no aplicativo Android para construir a lista com todas as rotas criadas pelo usuário atual.

Mensagem: objeto JSON com um campo "userId" cujo valor é o id do usuário.

Resposta: código de status 200 e array serializado com as rotas do usuário. Cada rota é um objeto JSON serializado, com os seguintes campos: id da rota, nome da rota e data da rota.

POST /route

Caminho utilizado para acessar todas as informações salvas referentes a uma rota especificada. É utilizado no aplicativo Android para construir a tela que apresenta as informações referentes a uma rota (como pontuação, infrações cometidas, etc).

Mensagem: objeto JSON com um campo "id" cujo valor é o id da rota.

Resposta: código de status 200 e objeto JSON serializado contendo as informações da rota, no mesmo formato especificado pela figura 53.

6.3.2 Pontuação

O desenvolvimento de um algoritmo inteligente de cálculo da pontuação da rota não fez parte do escopo deste projeto, devido à grande complexidade de implementá-lo e à dificuldade de se provar que ele é realmente confiável. Além disso, os algoritmos pesquisados que utilizavam técnicas de Machine Learning necessitavam de uma base já existente de dados para a classificação do motorista, a qual não está disponível.

Assim, decidiu-se priorizar a conexão de todos os componentes do sistema e a detecção e visualização das infrações em tempo real.

Desta maneira, para fins de testes, a pontuação (p) é calculada a partir da seguinte fórmula simplificada:

p = maxScore - nOverMaxSpeed - nSuddenBrake - nSuddenAcceleration - nSuddenTurn

onde:

maxScore pontuação máxima (arbitrariamente fixada em 100)

nOverMaxSpeed número de ultrapassagens do limite de velocidade

nSuddenBrake número de freadas repentinas

nSuddenAcceleration número de acelerações repentinas

nSuddenTurn número de curvas realizadas em velocidades muito altas

6.4 Banco de dados MongoDB

Esta seção descreve os resultados obtidos no nível do banco de dados da aplicação.

6.4.1 Coleções

Na banco de dados MongoDB, uma coleção ("collection") é um agrupamento de documentos ("document"). Cada documento representa um registro na base, sendo a unidade básica de dados. Comparando-se a um banco de dados relacional, um documento corresponde a uma linha em uma tabela, enquanto uma coleção a uma tabela.

No MongoDB, a estrutura de dados de um documento é definida por um esquema ("schema"), que especifica o tipo dos atributos a serem salvos, se é obrigatório, se faz referência a outro esquema, etc. Um documento, antes de ser guardado em uma coleção, é validado de acordo a um esquema.

Assim, no contexto do sistema proposto, foram criadas duas coleções principais para salvar os dados a serem persistidos: uma para as rotas e outra para os usuários. Antes de um documento ser persistido na coleção de rotas, ele é validado segundo o esquema que especifica a estrutura de dados das rotas. De maneira similar, um documento a ser persistido na coleção de usuários é validado em relação ao esquema da estrutura de dados de usuários.

Os esquemas (estrutura dos dados) das duas coleções são detalhados a seguir:

UserSchema

A figura 54 ilustra o esquema utilizado para a validação dos dados de usuários. Cada usuário possui um primeiro nome e sobrenome, além de login e senha para se logar à aplicação. Além disso, especifica-se que o login e a senha são campos obrigatórios e que o login deve ser único para cada usuário do sistema.

```
var UserSchema = new Schema({
  name: {
    first: String,
    last: String
  },
  credentials: {
    login: { type: String, required: true, trim: true, index: { unique: true } },
    password: { type: String, required: true }
  }
});
```

Figura 54 – Esquema dos usuários

RouteSchema

A figura 55 ilustra o esquema utilizado para a validação dos dados de rotas. Cada

rota possui um nome, um motorista (campo que faz referência à coleção de usuários), data de início e de fim, número de infrações cometidas e algumas métricas como velocidade e aceleração máximas atingidas.

```
var RouteSchema = new Schema({
  name: String,
  driver: {type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true},
  date: {
    start: { type: Date, required: true },
    end: { type: Date, required: true }
  },
  numberOfEvents: {
    overMaxSpeed: { type: Number, default: 0 },
    suddenBrake: { type: Number, default: 0 },
    suddenAcceleration: { type: Number, default: 0 },
    suddenTurn: { type: Number, default: 0 }
  },
  metrics: {
    maxSpeed: { type: Number, min: 0 },
    maxAcceleration: { type: Number, min: 0 }
  }
});
```

Figura 55 – Esquema das rotas

6.5 Integração

No teste de integração dos componentes, foi utilizado o Raspberry coletando dados do acelerômetro e do simulador de OBD-II, que foram enviados ao celular por meio de um canal RFCOMM Bluetooth.

No aplicativo Android, ao mesmo tempo que a rota do usuário era traçada no mapa, os dados recebidos da Raspberry foram analisados e, caso alguma penalidade fosse encontrada, uma notificação era enviada ao usuário e um marcador era inserido no mapa.

A rota é encerrada quando o usuário clicar sobre o botão "STOP ROUTE" da interface da figura 46. Uma janela é então aberta para que o usuário possa atribuir um nome à rota, que é salva logo em seguida. No teste de integração, um nome arbitrário foi inserido e conseguiu-se persistir a rota no banco de dados.

Então, retornou-se à tela anterior (tela principal) e verificou-se que a nova rota constava no histórico, bem como todas suas informações (pontuação e infrações cometidas).

Deste modo, foi possível testar todo o fluxo: do Raspberry até o servidor NodeJs e a volta, do servidor para o dispositivo Android. Com isso, faltaram testes de partes ainda não implementadas, como a obtenção da velocidade máxima da via e da calibração da aceleração. Outro ponto ainda ser testado é a captura de dados diretamente do painel do carro utilizando-se o protocolo OBD, em vez de se utilizar o simulador.

7 Conclusão

O sistema desenvolvido possibilitou a coleta de dados utilizando o protocolo OBD-II do simulador, o acelerômetro conectado ao hardware embarcado e o próprio aparelho celular. Estes dados foram então integrados pelo aparelho Android, onde alguns comportamentos imprudentes foram detectados e notificados em tempo real ao usuário. Finalmente, os dados foram enviados ao servidor, onde a pontuação foi calculada e o histórico persistido. Assim, realizamos a comunicação de ponta-a-ponta passando por todos os componentes da aplicação, implementando as principais funcionalidades propostas durante a especificação do projeto.

Contudo, o escopo do trabalho proposto inicialmente foi reduzido para entrega deste documento. Assim, os testes que antes seriam realizados com o sistema conectado diretamente ao automóvel foram executados somente com o simulador do OBD-II. Além disso, algumas infrações não foram detectadas, como a rotação do motor (RPM) muito elevada.

Isto pode ser explicado pelos desafios enfrentados durante o desenvolvimento do projeto, tais como:

- utilização de tecnologias diferentes: cada componente foi desenvolvido em plataformas e linguagens distintas, o que demandou um considerável tempo de familiarização e estudo;
- integração dos componentes do sistema: a comunicação entre os componentes, embora utilize técnicas e protocolos existentes, foi consideravelmente trabalhosa e de difícil depuração.
- variedade de comportamentos imprudentes a serem detectados: a implementação da detecção de cada infração demandou muitas vezes um estudo de uma nova biblioteca ou de outras ferramentas, o que consequentemente necessitou de mais tempo de desenvolvimento.

Assim, algumas sugestões de trabalhos futuros para aprimoramento deste projeto poderiam ser:

- Identificação e detecção de outros comportamentos imprudentes;
- Execução de bateria de testes em um veículo real;
- Design e implementação de um algoritmo de pontuação mais representativo e complexo;

- Envio de dados por uma shield GSM, dispensando assim o celular para o envio dos dados;
- Uso do protocolo OBD para consultar problemas mecânicos no carro, bem como consultar outros parâmetros disponíveis por meio deste sistema de diagnóstico;
- Armazenamento de dados no celular e na Raspberry;
- Armazenamento da maioria dos dados coletados no servidor, para uma posterior análise mais precisa feita pelo próprio servidor.
- Adição de um módulo GPS ao hardware embarcado, para ter o posicionamento independentemente do celular.

Mesmo considerando estes desafios enfrentados e a existência de possíveis melhorias a serem realizadas, a equipe considera-se muito satisfeita com todo o processo de desenvolvimento do projeto, que foi marcado por um período muito rico e de repleto aprendizado, além de ter sido uma ótima oportunidade para aplicarmos algumas das competências adquiridas durante todos estes anos no curso de Engenharia Elétrica com ênfase em Computação na Escola Politécnica.

Por fim, tentamos criar um modo de avaliar a direção do motorista utilizando o protocolo OBD-II e outros sensores embarcados no carro, com intuito de melhorar o comportamento do condutor no volante e reduzir o risco de acidentes. Assim, o "Como Dirijo" reflete a missão que levamos, enfim, como engenheiros: utilizar a ciência e tecnologia para melhorar a vida das pessoas.

Referências

Android Developers. *Android, the world's most popular mobile platform | Android Developers*. 2015. Disponível em: <http://developer.android.com/about/android.html>. Acesso em: 10.11.2015.

Ibiblio. *Features of Python*. 2015. Disponível em: <https://www.ibiblio.org/swaroopch/byteofpython/read/features-of-python.html>. Acesso em: 10.11.2015.

IbnLive. *Android 5.0 Lollipop: 10 highlights of the latest version of Google's Android operating system - IBNLive*. 2015. Disponível em: <http://www.ibnlive.com/news/india/android-5-0-lollipop-10-highlights-of-the-latest-version-of-googles-android-operating-system-720503.html>. Acesso em: 10.11.2015.

Python.org. *History and License - Python 2.7.10 documentation*. 2015. Disponível em: <https://docs.python.org/2/license.html>. Acesso em: 09.11.2015.

Python.org. *The Python Logo | Python.org*. 2015. Disponível em: <https://www.python.org/community/logos/>. Acesso em: 09.11.2015.

AKSHAYKUMAR, O.; MANJARI, S.; SRINIVAS, R. Driver evaluation system using mobile phone and obd-ii system. *(IJCSIT) International Journal of Computer Science and Information Technologies*, v. 6, n. 3, p. 2738–2745, 2015.

Bluetooth Development Portal. *RFCOMM | Bluetooth Development Portal*. 2015. Disponível em: <https://developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx>. Acesso em: 09.11.2015.

COBLIE. *Rastreamento Veicular | Cobli*. 2015. Disponível em: <https://www.cobli.co/br/produtos.html>. Acesso em: 13.11.2015.

DINGUS, T. A. et al. *The 100-Car Naturalistic Driving Study Phase II – Results of the 100-Car Field Experiment*. [S.l.], 2006.

Ericsson History . *The history of bluetooth - Ericsson History*. 2015. Disponível em: <http://www.ericssonhistory.com/changing-the-world/Anecdotes/The-history-of-Bluetooth-/>. Acesso em: 09.11.2015.

Learn Sparkfun . *Bluetooth Basics - Learn Sparkfun*. 2015. Disponível em: <https://learn.sparkfun.com/tutorials/bluetooth-basics>. Acesso em: 09.11.2015.

OBD-II Background Information . *OBD-II Background Information*. 2015. Disponível em: <http://www.obdii.com/background.html>. Acesso em: 23.10.2015.

PCMag.com. *Raspberry Pi Review and Rating | PCMag.com*. 2015. Disponível em: <http://www.pcmag.com/article2/0,2817,2407058,00.asp>. Acesso em: 09.11.2015.

Raspberry Foundation . *Raspberry pi Foundation - About us*. 2015. Disponível em: <https://www.raspberrypi.org/about/>. Acesso em: 24.10.2015.

Super pi Boy. *Raspberry Pi | Super Pi Boy*. 2015. Disponível em: <https://superpiboy.wordpress.com/tag/raspberry-pi/>. Acesso em: 09.11.2015.

VOBIS, C. *How's My Driving? Providing Driver Feedback to Improve Driving*. Dissertação (MasterThesis), 2012.