Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

Um Sistema Embarcado para Análise e Classificação de Condução de Motoristas

Felipe de Paula Wanderley Barros

Trabalho de Graduação

Recife, 2016

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Engenharia da Computação

Felipe de Paula Wanderley Barros

Um Sistema Embarcado para Análise e Classificação de Condução de Motoristas

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Engenharia da Computação da Universidade Federal de Pernambuco como requisito parcial para a conclusão da graduação desse curso.

Orientador: Prof. Divanilson Campelo

Recife, 2016

Agradecimentos

Primeiramente, eu dedico esse trabalho e tudo o que foi realizado durante esses seis anos de graduação à minha família. Família que esteve sempre do meu lado, desde a decisão de largar uma Universidade Federal até o desafio de morar sozinho em uma cidade desconhecida. Aos meus pais e avós pelo apoio incondicional, tanto emocional quanto financeiro. Ao meu irmão pelas experiências compartilhadas e pelo suporte sempre que necessário. E aos meus tios Isabela e Jorge que, junto com Biel, me acolheram em seu lar durante minha jornada em Recife. Ter vocês como um porto seguro sempre que necessário fez as coisas ficarem menos difíceis e possíveis. Essa conquista é nossa.

Também gostaria de agradecer à minha turma de graduação, ao Engenhação. Grupo unido que possibilitou que a árdua trajetória de uma graduação de Engenharia da Computação fosse mais divertida e memorável. Dedico essa conquista às noites em claro estudando juntos, e aos acampamentos dentro do Cin para concluirmos trabalhos. Histórias que nunca esqueceremos.

Além destes amigos, também gostaria de lembrar do pessoal que fez um dos anos da minha graduação ser mais especial, durante o período que participei do CITi, pelo aprendizado e pela experiência diferenciada que é participar da comunidade de empresários juniores.

Por fim, gostaria de agradecer a todos os funcionários do CIn-UFPE, desde faxineiros a professores que fazem do meu Centro de Informática um pólo referência em computação.

Resumo

Com o advento do conceito *Internet of Things*, existe um interesse crescente na conectividade entre os mais distintos dispositivos eletrônicos, bem como na análise de dados que os mesmos possam vir a trocar, fazendo proveito do aumento da capacidade de processamento e do barateamento de sistemas embarcados. Os automóveis estão inclusos nesta lista de dispositivos conectáveis, cujas redes internas geram enorme quantidade de dados a todo momento, mas que são direcionados apenas para controles eletrônico e mecânico internos do veículo, ou para funcionalidades de conforto e segurança. O protocolo mais utilizado atualmente para a comunicação entre os módulos eletrônicos de um automóvel é o protocolo CAN, o qual é utilizado especificamente em aplicações automotivas e industriais. Por este motivo, um sistema para análise da rede CAN de um carro foi desenvolvido com o intuito de identificar e alertar sobre comportamentos do motorista que possam vir a causar danos ao veículo, além de classificar o *modus operandi* do condutor utilizando um algoritmo de Aprendizagem de Máquina. Essas informações são recuperadas em uma placa Arduino conectada ao veículo e processadas e enviadas por uma placa Intel Edison, via internet, para um servidor Web o qual disponibiliza estas informações já formatadas para o usuário. O experimento foi realizado em um Renault Sandero (2015) recuperando informações de velocidade, rotações por minuto e pressão do pedal do acelerador, as quais foram processadas de modo a demonstrar estatísticas de comportamento de frenagens e trocas de marcha realizadas pelo motorista.

Palavras chaves: Redes Automotivas, Controller Area Network, CAN, Inteligência Artificial, Internet das Coisas, Aprendizagem de Máquina.

Abstract

With the advent of the *Internet of Things* concept, there is an increasing interest regarding to connectivity among different electronic devices, as well as to data analysis of interchanged information among them, taking advantage of the increased processing capacity and cheaper embedded system devices. Cars are included into this list of connectable devices, whose internal networks generates tremendous amounts of data every moment, but that are directed only to electronic and mechanic control, or even to comfort and safety features. The most used protocol for internal communication in vehicles is called CAN, which is mainly used for automotive and industrial applications. For this reason, a system has been developed to analyze a CAN network aiming at identifying and alerting about the driver's behavior, which may harm the vehicle mechanically, as well as classifying driver's driving using an algorithm of artificial intelligence. This information is recorded using an Arduino board connected to the vehicle and processed and sent through an Intel Edison board, via Internet, to a web server, which provides formatted data to the user. The experiment was realized into a Renault Sandero (2015) accessing information such as current speed, rotations per minute and accelerator pedal position, which were processed aiming at delivering statistics about breaking and gear changing conducted by the driver.

Key words: Automotive Networks, Controller Area Network, CAN, Artificial Intelligence, Internet of Things.

Índice de Tabelas e Figuras

Sumário

Capítulo 1

Introdução

A temática envolvendo redes intraveiculares configura uma das áreas da computação com maior potencial e interesse atualmente, sendo alavancada pelos promissores mercados dos carros autônomos, segurança e da indústria de aplicativos para o nicho automobilístico. Além disso, o barateamento de sistemas embarcados e o aumento da capacidade computacional destes dispositivos encorajam a criação de aplicações “pesadas” voltadas para ambientes até então pouco explorados, como uma rede interna de um automóvel.

Este cenário de alta visibilidade da área de Redes Automotivas juntamente com a maior acessibilidade a componentes eletrônicos estimula o desenvolvimento de trabalhos como este que criem maneiras de analisar o comportamento do motorista, visando o seu aperfeiçoamento e, consequente, ajude a evitar acidentes e mortes no trânsito. Segundo a pesquisa *Acidentes de trânsito nas rodovias federais brasileiras* do IPEA (Instituto de Pesquisa Econômica Aplicada), dos 169.153 acidentes de trânsito registrados pela PRF (Polícia Rodoviária Federal) em 2014, 17.937 aconteceram por velocidade incompatível, 54.317 por falta de atenção e 6.787 por defeitos mecânicos nos veículos, dentre os quais 2.379 envolveram vítimas fatais [12]. Estes três motivos de acidentes são direta ou indiretamente avaliados pelos dados gerados por esse trabalho. Ainda segundo a pesquisa citada acima, todos os acidentes acontecidos em 2014 significaram R$ 12,3 bilhões de reais em custos para o governo. Considerando que estes acidentes contribuíram igualmente para este valor, os acidentes causados pelos três motivos citados acima custariam R$ 5,74 bilhões de reais às contas públicas naquele ano.

O CAN (Controller Area Network) é o protocolo mais empregado nos ambientes veiculares atualmente, sendo utilizado para controlar o funcionamento e a comunicação entre ECUs (Electronic Control Unit) que, por sua vez, são dispositivos eletrônicos cuja finalidade é controlar funcionalidades veiculares como freio ABS (Anti-lock Braking System) e AirBag. Carros modernos podem conter mais de 50 ECUs conectadas entre si [11]. O emprego de tantas ECUs resulta em uma quantidade relativamente alta de dados trafegando na rede CAN de um automóvel que, com o tratamento correto, pode vir a fornecer informações valiosas.

Sistemas embarcados são sistemas eletrônicos com capacidade computacional e tamanho reduzidos, cuja prototipação ocorre visando atingir apenas um conjunto de funcionalidades, ao contrário de um sistema computacional de propósito geral, para o qual não se pode predizer em qual ambiente o mesmo será inserido. Exemplos de sistemas embarcados estão nos controles de desde MP3 players, até em ECUs em carros e aviões. Com a evolução destes sistemas de propósito definido, principalmente do seu poder computacional, abre-se a possibilidade da inserção de algoritmos mais complexos, como algoritmos de Inteligência Artificial.

A evolução da Internet, democratização das tecnologias móveis e o conceito de nuvem têm condicionado os usuários comuns a depender e, até, reivindicar que uma infinidade de tipos serviços estejam disponíveis em todo lugar, serviços estes que tendem a se tornar gradualmente mais complexos.

Este trabalho visa unificar os conceitos observados nos parágrafos anteriores em uma aplicação embarcada que propõe observar o tráfego interno de uma rede CAN intraveicular e, a partir dos dados coletados, processar as informações capturadas analisando o comportamento com o qual o motorista conduz o veículo, gerando dados estatísticos sobre esse comportamento, classificando o tipo de condução a partir de algoritmos de Inteligência Artificial, e disponibilizando esses dados em tempo real no ambiente web.

Capítulo 2

Fundamentação Teórica

Este capítulo fornece toda a informação necessária a respeito dos conceitos teóricos que serviram como base para o desenvolvimento deste trabalho.

2.1 Redes Automotivas

Veículos deixaram de ser apenas dispositivos mecânicos. Veículos atuais possuem um número crescente de dispositivos eletrônicos conectados entre si que, como um todo, são responsáveis por monitorar e controlar o estado do veículo [11]. Atualmente, existem diversos tipos de redes que são utilizadas em automóveis, cada qual com suas particularidades e objetivos específicos, variando em largura de banda, custo, complexidade entre outros quesitos. Dentre as redes automotivas mais conhecidas estão LIN (Local Interconnect Network), FlexRay, MOST (Media Oriented Systems Transport) e CAN (Controller Area Network), sendo esta última a utilizada nesse trabalho, e que será explicada nas seções subsequentes.

2.2 CAN – Controller Area Network

O protocolo CAN foi desenvolvido pela BOSCH como um sistema multimaster com mensagens broadcast com uma taxa máxima de 1 Megabits por segundo (1Mbps). Ao contrário de outros protocolos de rede como USB ou Ethernet, o CAN não envia grande pacotes de dados de ponto a ponto de um nó A até um nó B sob a supervisão de um nó master central. No protocolo CAN, várias pequenas mensagens, tais como temperatura e RPM (Rotações por Minuto), são difundidas na rede por completo, permitindo que qualquer nó conectado ao barramento possa acessar essa informação [3].

Uma vez que os princípios básicos do CAN são o formato da mensagem, seus tipos e identificadores, os esquema de arbitração e de verificação de erros, eles serão explicados nas próximas seções.

2.2.1 Arquitetura CAN

O protocolo de comunicação CAN, ISO-11898:2003, descreve como a informação é trocada entre seus nós conectados em uma rede de acordo com o modelo OSI (Open Systems Interconnection), a qual é definida em camadas. O ISO-11898, documento que descreve o protocolo CAN, define as últimas duas camadas das sete do modelo OSI, como a camada de enlace (*Data Link Layer*) e camada física (*Physical Layer*) da Figura 1 [3] abaixo:

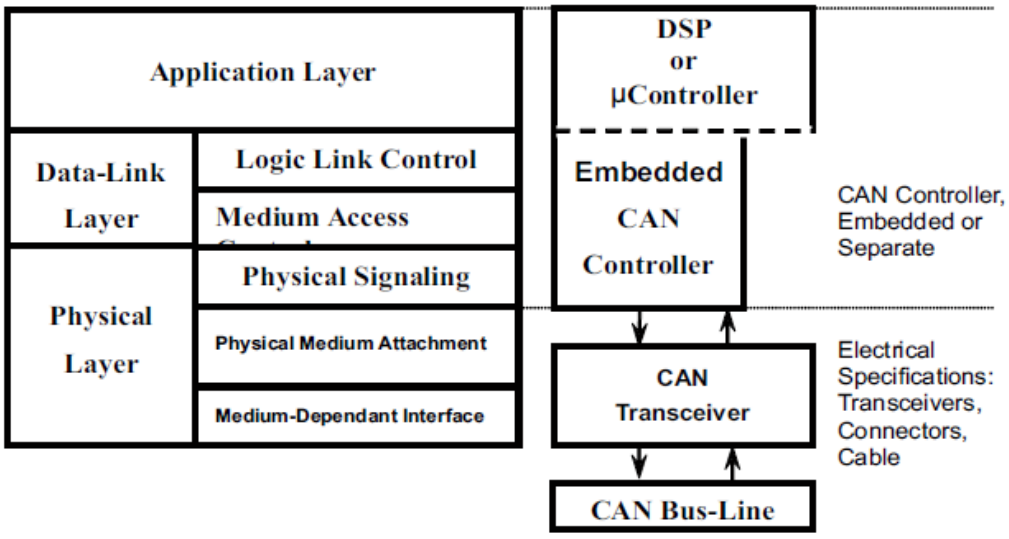


Figura 1. Arquitetura definida no padrão ISO-11898 [3].

2.2.2 Mensagem CAN

O protocolo CAN possui duas variações de composição dos seus campos de mensagem, sendo eles: o *Standard CAN* onde o campo ID da mensagem possui 11 bits no total, e o *Extended CAN* em que este campo possui 29 bits. Durante este trabalho, apenas o primeiro padrão foi utilizado, sendo o mesmo ilustrado na Figura 2 [3]:

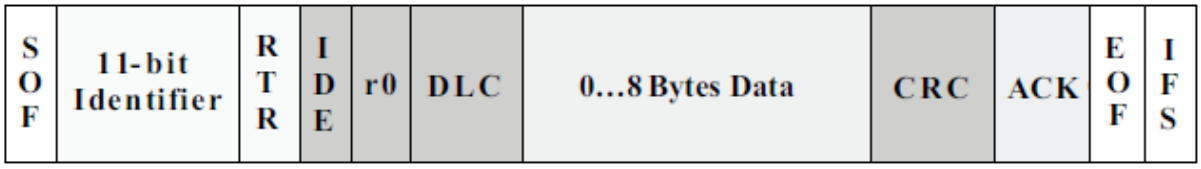


Figura 2. Campos da mensagem do Standard CAN [3].

- **SOF:** Um único bit dominante (*Start of Frame*) que marca o início da mensagem, e é utilizado para sincronizar os nós conectados ao barramento, após o mesmo ter estado ocioso.
- **IDENTIFIER:** Este campo é o responsável por definir a prioridade das mensagens CAN. Quanto menor for seu valor, maior a prioridade da mensagem.
- **RTR:** Este bit (*Remote Transmission Request*) é dominante quando alguma informação é requerida de outro nó. Todos os nós recebem a mensagem, mas o ***Identifier*** determina qual deles deve responder, da mesma maneira que a

informação respondida também é recebida por todos os nós conectados ao barramento.

- **IDE:** Este bit é responsável por identificar o padrão utilizado, se o *Standard CAN* ou o *Extended CAN.*
- **R0:** Bit reservado para uso futuro.
- **DLC:** 4 bits contendo o número de bytes de dados que estão sendo transmitidos (*Data Length Code*).
- **DATA:** Até 64 bits de dados que possam estar sendo transmitidos.
- **CRC:** 16 bits utilizados para detecção de erros (*Cyclic Redundancy Check*).
- **ACK:** Todos os nós sobrescrevem esse bit para indicar que o pacote foi recebido com sucesso e livre de erros. Se houve erro, o nó deixa o bit recessivo, descarta a mensagem, e o emissor a reenvia**.**
- **EOF:** Este campo de 7 bits marca o fim do frame CAN.
- **IFS:** Este campo de 7 bits contém o tempo requerido pelo controlador para pôr um frame recebido na posição correta dentro de seu buffer de mensagens.

Durante o projeto, foi necessário descobrir o valor correto dos IDs para as mensagens de velocidade instantânea, valor do RPM e pressão no pedal do acelerador, sendo necessária uma consulta na documentação do padrão OBD-II, o qual define os valores de ID's de mensagens nos barramentos automotivos, e disponível em [4].

2.2.3 Arbitração

A arbitração é uma característica fundamental do protocolo CAN, a qual garante que mensagens com maior prioridade poderão trafegar no barramento sem colisões ou erros. O acesso ao barramento é disparado por eventos e acontece randomicamente. Se dois nós tentam ocupar o barramento no mesmo instante, o acesso ao barramento é implementado de maneira não destrutiva com uma arbitração bit a bit, garantindo o barramento para a mensagem com o menor valor de *Identifier.* Não destrutivo significa que o nó ganhando a arbitração segue enviando sua mensagem, sem que a mesma tenha sido destruída pelo envio de outro nó [3]. Para que a arbitração aconteça, cada nó segue verificando o barramento à medida que transmite, possibilitando uma comparação entre bit enviado e bit presente no barramento. Caso uma mensagem com maior prioridade esteja presente no barramento, o nó para imediatamente a sua transmissão. A Figura 3 [3] exemplifica o caso de concorrência por um barramento CAN entre dois nós, onde o nó C ganha a arbitração inicialmente e envia sua mensagem por completo. Após isso, o nó B ganha a arbitração e está autorizado a enviar sua mensagem.

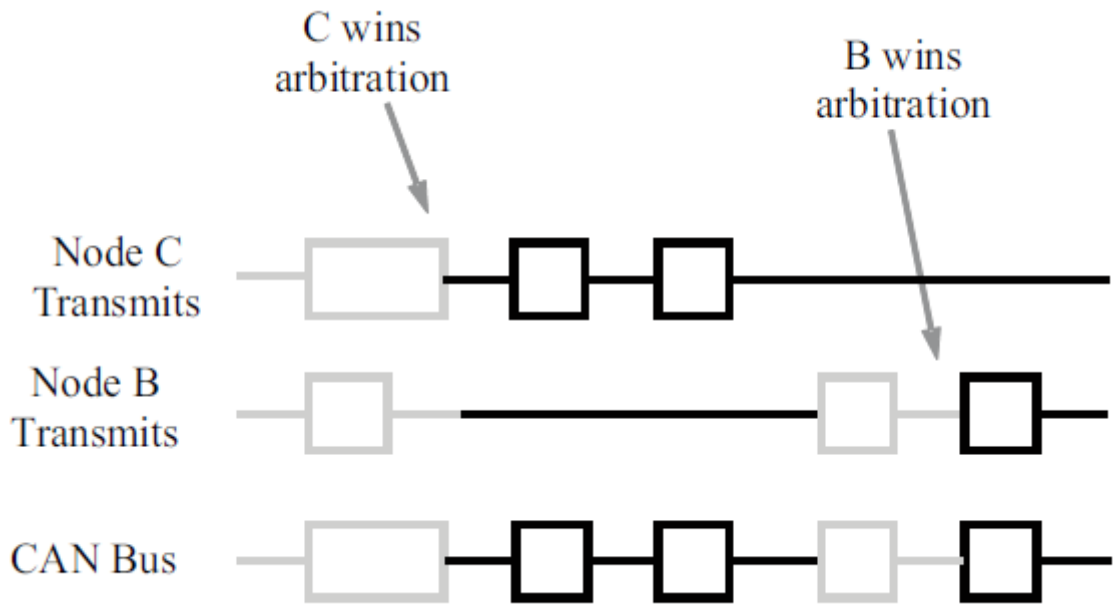


Figura 3. Exemplo de arbitração no barramento CAN [3].

2.2.4 Verificação de Erro

A robustez do protocolo CAN pode ser atribuída em parte ao abundante procedimento de verificação de erros. O protocolo CAN possui cinco diferente métodos de checagem de erro, sendo três em nível de mensagem e mais dois em nível de bits. Se a mensagem falhar em qualquer um deles, esta não é aceita pelo receptor e uma mensagem de erro é gerada no barramento, obrigando o emissor a reenviar a mensagem até que esta seja recebida corretamente. Além disso, se um nó começar a enviar mensagens de erro continuamente no barramento, sua capacidade de transmissão é removida por seu controlador após atingir um limite máximo de mensagens [3].

2.3 Inteligência Artificial

Inteligência Artificial é a área da computação responsável por estudar e desenvolver algoritmos capazes de reconhecer e classificar padrões, como por exemplo, reconhecimento de fala, identificação de impressão digital e muitas outras aplicações úteis e já empregadas na indústria [2].

Por esse motivo, idealizou-se o uso dessas ferramentas de classificação automatizada para incorporar a este trabalho, na tentativa de fornecer uma maneira de classificar o comportamento do motorista após um período dirigindo. Esta funcionalidade foi implementada utilizando o algoritmo de classificação *KNN (K-Nearest Neighboor)*, o qual será explicado na próxima seção.

2.3.1 kNN – K Nearest Neighboor

O algoritmo KNN é um algoritmo baseado em instância que, por definição, é um tipo de algoritmo que não necessita de processamento prévio para "ensinar" a máquina de classificação (no caso de Redes Neurais, é necessário que a máquina seja "ensinada" para calibrar os pesos das arestas que serão utilizadas no processo de classificação). Ou seja, todo o processamento necessário é transferido ao processo de classificação, sendo o processo de aprendizagem apenas o armazenamento do conjunto de treinamento [1]**.** Para esse trabalho, em que o processamento acontece em uma plataforma embarcada, a utilização de um método semelhante a Redes Neurais seria o ideal, uma vez que o processo de aprendizagem poderia ser feito em um ambiente otimizado, e apenas a função de classificação passada ao embarcado. No entanto, pela simplicidade da implementação e do conceito por trás do KNN, este foi escolhido.

O algoritmo KNN consiste em uma solução clássica de Aprendizagem de Máquina, em que o conceito básico é, dado um objeto de análise, classificá-lo com a classe com maior ocorrência dentre os outros elementos do espaço, considerando apenas os **K** elementos mais próximos ao elemento de análise. A **distância** entre dois elementos do espaço de objetos é dada pela **Distância Euclidiana** [1]. Seja uma instância arbitrária definida pelo vetor de características:

$$\left(a_1(x), a_2(x), \dots, a_n(x)\right) \tag{2.1}$$

onde $a_n(x)$ denota a r-ésima característica do elemento **x.** Então, a distância Euclidiana entre duas instâncias x_i e x_j é dada por $d(x_i,x_j)$, onde:

$$d\left(x_i, x_j\right) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2} \tag{2.2}$$

Os valores atribuídos às características podem ser discretos ou contínuos para esse algoritmo, necessitando criar uma forma de cálcular a distância Euclidiana para o primeiro caso, que não seja apenas subtrair valores [1]. No entanto, esse trabalho utilizou valores reais para definir as características do seu **conjunto de treinamento**[1], não necessitando de adaptação no algoritmo.

[1] Dá-se o nome de **conjunto de treinamento** ao grupo composto pelos objetos cujas características têm seus valores conhecidos, assim como suas classes definidas. Esse grupo é utilizado para determinar a classe de uma instância que se deseja classificar.

Calculadas as distâncias entre os elementos do conjunto de treinamento e o elemento de análise, escolhem-se os **K** elementos mais próximos para análise. A classe com maior incidência dentre esses **K** elementos é então escolhida para classificar o elemento de análise.

2.4 HTTP Protocol

O protocolo HTTP (Hypertext Transfer Protocol) é o protocolo de comunicação utilizado na camada de aplicação (segundo o modelo OSI) por sistemas de informação de hipermídia, distribuídos e colaborativos [5].

Este é o protocolo utilizado pela aplicação para permitir a comunicação entre o sistema embarcado e o servidor responsável por disponibilizar os dados na web.

Capítulo 3

Captura de Dados no Barramento CAN

Este capítulo tem como objetivo detalhar o processo inicial de captura dos dados a partir do barramento CAN presente no automóvel.

3.1 Introdução

A etapa inicial do experimento realizado neste projeto se resumiu em acessar o barramento CAN de um automóvel em funcionamento, e colher informações relevantes para um processamento futuro. Este objetivo foi alcançado construindo um *sniffer*[2] que pudesse se conectar à entrada para diagnósticos automotivos conhecida como OBD-II, a qual está presente na maioria dos carros atuais para colher, em tempo real, informações como velocidade instantânea, rotações por minuto e pressão exercida no pedal do acelerador. Estes dados foram então salvos em arquivos de log pra serem utilizados na análise que será explicada em 4.4.

3.2 Componentes Utilizados

Todos os componentes utilizados no desenvolvimento do *sniffer* serão descritos nas sub-seções seguintes:

3.2.1 Arduino Uno

O Arduino é a placa com microcontrolador baseada no ATmega328. Ela possui 14 saídas/entradas digitais (das quais 6 podem ser utilizadas como saídas PWM (Pulse-Width Modulation), 6 saídas analógicas, um cristal de quartzo de 16MHz, uma conexão USB, uma entrada de energia, e um botão de reset [7] .

2 Em redes de computadores, *sniffer* é o termo designado para uma ferramenta de hardware ou software capaz de se conectar a uma rede, interceptar e registrar o tráfego de dados. Esta ferramenta ainda é capaz de capturar cada pacote, decodificá-lo e analisar seu conteúdo [6].

Figura 4. Arduino Uno.

Micro controlador	Atmega328
Voltagem de Operação	5V
Pinos de I/O Digitais	14
Pinos Analógicos de Entrada	6
Memória Flash	32KB (0.5KB para bootloader)
SRAM	2KB
EPROM	1KB
Clock	16MHz

Tabela 1. Especificações técnicas do Arduino Uno.

3.2.2 CAN-Shield

Esta foi a placa utilizada como interface entre o Arduino Uno e o barramento CAN do automóvel. Esta CAN-Shield fabricada pela empresa Seeed (http://www.seeedstudio.com/depot/) integra o controlador CAN MCP2515 e o transceiver CAN MCP2551 através de uma interface SPI proporcionando ao Arduino a capacidade de executar ações que um nó CAN de um carro também possui [8].

A comunicação entre este dispositivo e o carro ocorre com o auxílio de um cabo OBD-II que será detalhado na próxima seção.

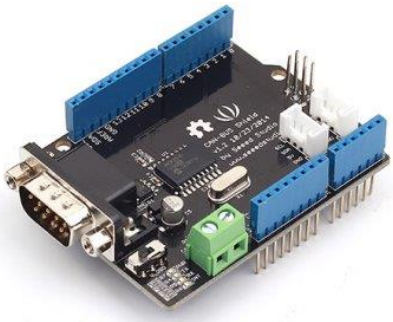


Figura 5. Seeed CAN Shield.

Na Figura 6 abaixo estão detalhadas algumas características técnicas do CAN Shield, assim como o posicionamento dos pinos utilizados no barramento CAN, como CAN High e CAN Low, além do GND e a alimentação de 12V.

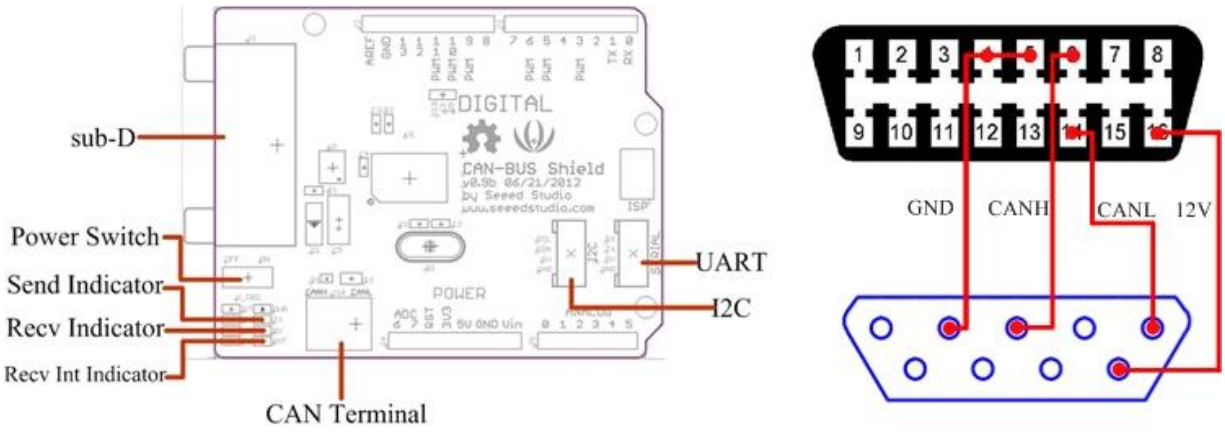


Figura 6. Especificações técnicas do Seeed CAN Shield.

3.2.3 Cabo OBD-II

Como já mencionado anteriormente, o padrão OBD-II é uma interface utilizada para diagnósticos em automóveis e está presente na maioria dos carros atualmente. Por isso, um cabo OBD-II foi utilizado para possibilitar a conexão entre veículo e a porta serial da CAN-Shield. Dentre os 16 pinos presentes no padrão OBD-II, os pinos 4 (GND), 6 (CANH), 14 (CANL) e 16 (12V) são aqueles utilizados especificamente para permitir o acesso ao barramento CAN do carro.

Figura 7. Cabo OBD-II.

Figura 8. Configuração da pinagem do cabo OBD-II.

3.3 Metodologia

Esta seção do capítulo tem como objetivo detalhar todo o fluxo seguido neste trabalho, envolvendo desde a etapa de Captura de Dados, representada por este capítulo, até as demais etapas do trabalho, as quais serão detalhadas nos capítulos seguintes.

Este trabalho pode ser dividido em três etapas principais, sendo elas:

- **Captura de Dados no Barramento CAN:** Etapa responsável por colher os dados que trafegam no barramento CAN do veículo e salvá-los em arquivo para posterior processamento.
- **Processamento dos Dados:** Etapa responsável por processar os dados gerados na etapa anterior, e enviá-lo ao Servidor Web, presente na próxima etapa. Este processamento dos dados envolve detectar os eventos de frenagem e trocas de marcha efetuados pelo condutor, e classificá-los. Por fim, o percurso é classificado como um todo através de Inteligência Artificial.

- **Interface com o Usuário:** Última etapa do sistema, responsável por receber os dados processados e disponibilizá-los na Web para o usuário. Esta etapa contém um Servidor Web responsável por atender as requisições realizadas pelo Browser e pelo sistema de Processamento de Dados presente na etapa anterior.

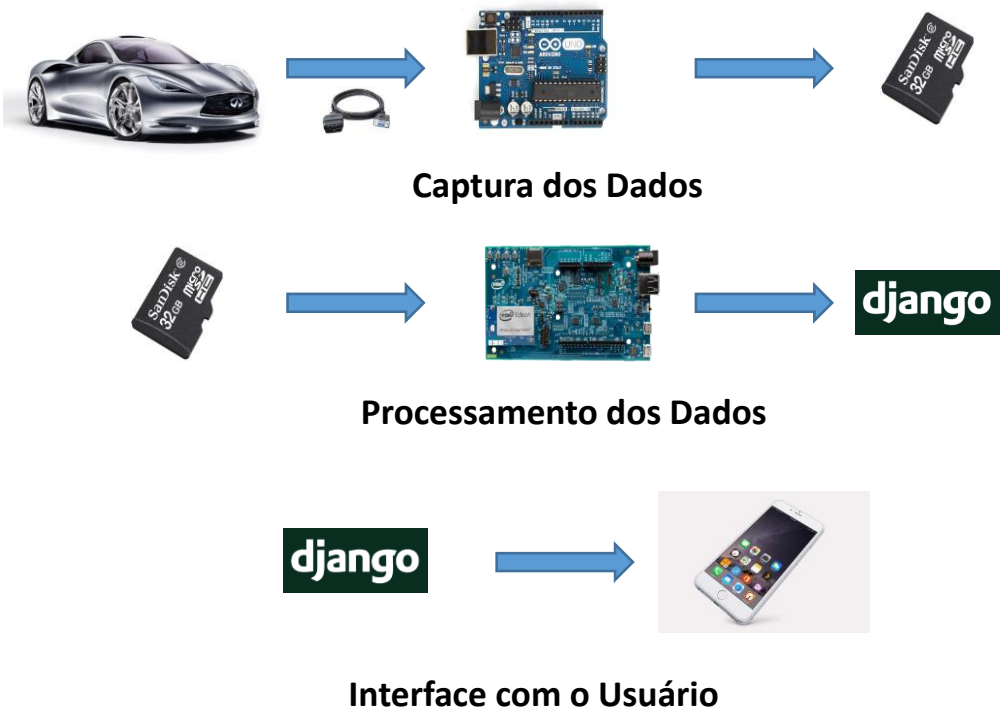


Figura 9. Diagrama com o fluxo entre as etapas do trabalho.

3.4 Extração dos Dados do Veículo

Antes de detalhar os procedimentos realizados para extração de dados, vale lembrar que o objetivo final do projeto era classificar o comportamento do motorista baseado em suas frenagens e trocas de marcha. Portanto, era necessário extrair informações relevantes para estes tipos de análises.

O *sniffer* foi conectado à entrada OBD-II do automóvel Renault Sandero (2015) e, a partir desta conexão, informações em formato de log foram salvas em arquivos .txt para serem utilizadas para análise em uma etapa seguinte. O sistema foi programado para requisitar três informações distintas a respeito do estado interno do veículo a cada 0.6 segundos. Estas informações foram:

- **Velocidade Instântanea:** A velocidade recebida é dada em Km/h sendo esta informação essencial para a análise da frenagem, utilizada para gerar um padrão de acelerações exercidas pelo carro.
- **Rotações por minuto:** O valor do RPM instantâneo é dado em valor absoluto, sendo essencial para a análise das trocas de marchas.
- **Pressão no pedal do acelerador:** O valor retornado desta requisição está entre 0 e 100, e indica a posição relativa do pedal do acelerador no momento em que a requisição é atendida.

A Figura 10 abaixo mostra a conexão do *sniffer* no carro através da ferramenta de diagnósticos OBD-II.

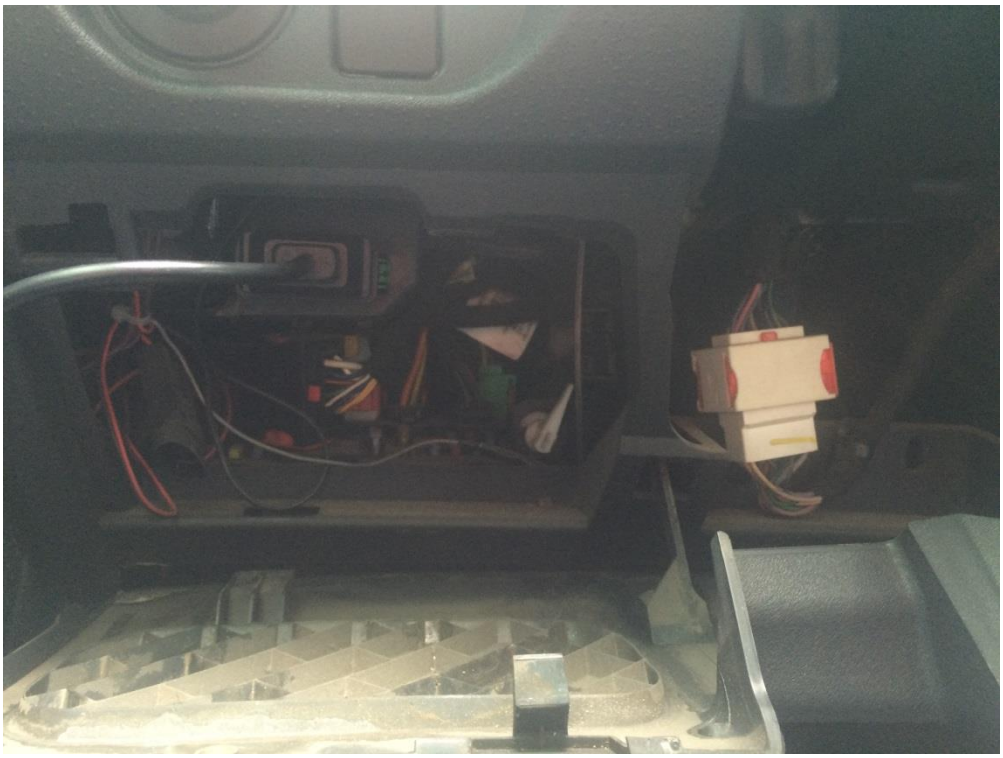


Figura 10. Conexão do cabo OBD-II no veículo.

A Figura 11 abaixo mostra a conexão do sniffer ao laptop para que os dados pudessem ser impressos no monitor serial enquanto o carro estivesse em funcionamento.

Figura 11. Conexão do *sniffer* no laptop.

3.5 Funcionamento

Inicialmente, dado que o desenvolvimento do *sniffer* dependeu da integração entre Arduino e o CAN-Shield, foi necessário que duas bibliotecas fossem utilizadas para o controle do segundo, sendo elas: SPI (*Serial Peripheral Interface*), a qual é responsável por implementar funções de controle e comunicação entre microcontroladores que utilizam este tipo de interface para troca de dados, como o controlador e transceiver presentes no CAN-Shield; e, a biblioteca mcp_can.h, a qual é liberada pela própria Seeed, fabricante do CAN-Shield, disponibilizando funções com alto nível de abstração para o desenvolvedor.

Após desenvolvido e carregado na memória interna do Arduino, o código passa ser executado assim que a placa é energizada, neste caso, ao se conectar com o automóvel. Além disso, quando conectada a um computador pessoal, o desenvolvedor tem a possibilidade de acompanhar os dados gerados no embarcado através de um monitor de saída serial do Arduino, como mostra a Figura 12. Esse monitor serial foi a ferramenta utilizada nesse trabalho para colher os dados do veículo e então salvá-los em arquivos de log.

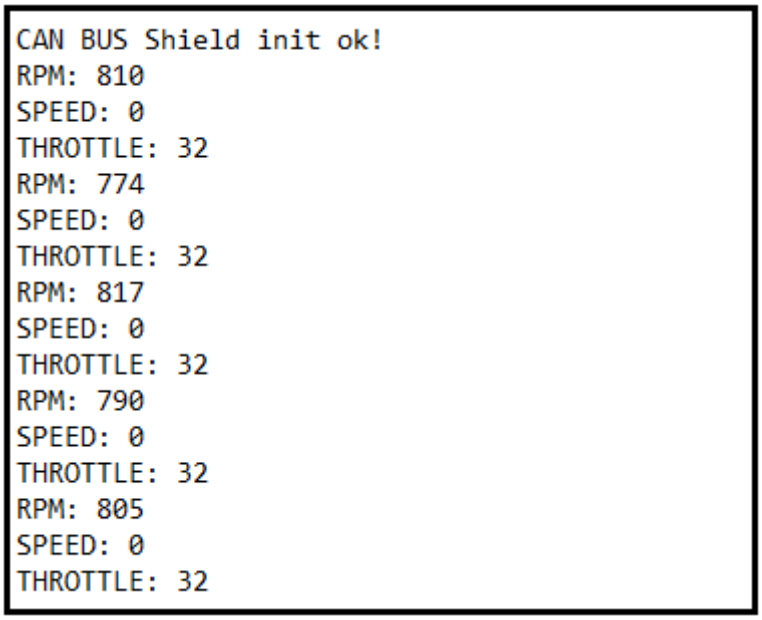


Figura 12. Trecho dos dados impressos no monitor serial.

Em nível de código, o primeiro passo da implementação é inicializar os periféricos conectados ao Arduino, neste caso monitor serial e o CAN-Shield, o qual foi configurado para transmitir a uma taxa de 500Kbps. Após este *setup* inicial, o sistema passa a realizar três requisições dentro de um laço a cada 0.6 segundos, recuperando as informações de velocidade instantânea, RPM e pressão no pedal do acelerador, e imprimindo esses dados no monitor serial. Este processo se repete indefinidamente até que o carro seja desligado. Neste momento, os dados impressos no monitor serial são salvos nos arquivos .txt para serem analisados posteriormente. A Figura 13 ilustra o funcionamento do *sniffer.*

O pseudo código desta etapa está presente em 8.1.

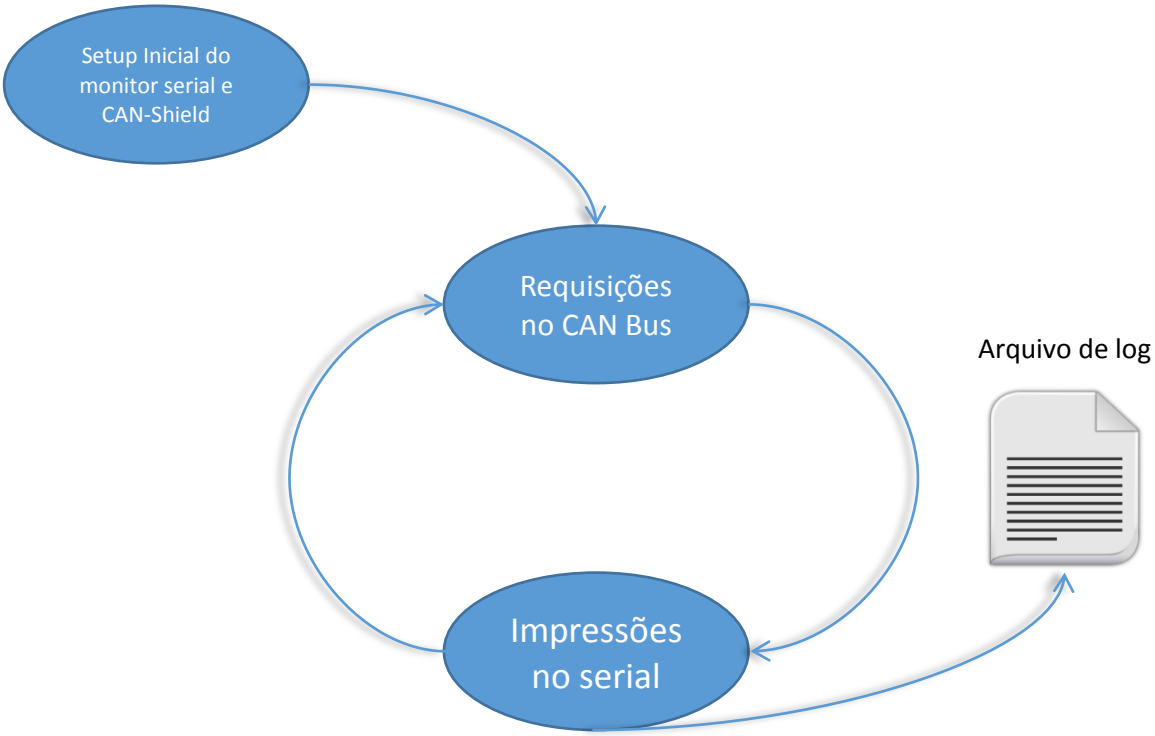


Figura 13. Diagrama com o fluxo do código do *sniffer.*

Capítulo 4

Processamento dos Dados

Este capítulo tem como objetivo detalhar a etapa do projeto a qual foi responsável pelo processamento dos dados colhidos pela etapa de captura de dados, detalhada no capítulo anterior.

4.1 Introdução

Esta etapa intermediária do trabalho consistiu em processar os dados colhidos na etapa anterior de forma a detectar eventos de frenagem e troca de marcha, categorizá-los em intervalos de acordo com o nível de ameaça à integridade do veículo, gerar dados estatísticos sobre esses eventos e então classificar o comportamento do motorista utilizando o algoritmo de Aprendizagem de Máquina KNN. Para processar todos esses dados, foi utilizada outra plataforma embarcada chamada Edison e fornecida pela empresa Intel, a qual será apresentada mais detalhadamente das seções subsequentes. Por último, ainda nessa etapa, os dados gerados são formatados e transmitidos a um servidor Web, cuja função é disponibilizar essas informações na nuvem e que será detalhado no próximo capítulo.

4.2 Objetivo

A etapa de Processamento dos Dados é, de fato, o coração do trabalho pela responsabilidade em gerar todo o conteúdo que justifica a realização do projeto. O objetivo desta etapa foi transformar toda a informação “crua” que fora colhida do barramento CAN do veículo em dados com maior valor para o usuário final, entregando uma análise a respeito do perfil de motorista que este usuário possui.

Idealmente, esta etapa e a de Captura de Dados deveriam ser executadas no mesmo momento, pelo mesmo sistema, simulando um sistema real que estaria conectado ao veículo e gerando informação em tempo real enquanto o motorista está dirigindo. No entanto, por limitações dos equipamentos utilizados, não foi possível realizar o experimento desta maneira, obrigando a separação dos sistemas.

4.3 A Intel Edison

A placa Intel Edison é uma plataforma de prototipação de aplicações IoT (*Internet of Things*). Em um módulo de apenas 9 cm^2 estão presentes um processador Atom dual-core de 500Mhz, 1GB RAM DDR3, 4GB de memória Flash, Bluetooth 4.0 e Wi-Fi. Além disso, esta placa roda uma versão diferenciada do Linux chamada Yocto, possibilitando um ambiente capaz de rodar linguagens como Python e Java em um ambiente embarcado. Juntamente com o módulo Intel Edison, também foi utilizado o Arduino Breakout Board, cuja finalidade é destrinchar os 70 pinos da Edison em uma placa de prototipação semelhante a uma placa Arduino [9]. A Tabela 2 detalha as demais especificações técnicas da Intel Edison.

Apesar do alto poder de processamento e da disponibilização de ferramentas atuais para comunicação, esta placa não foi utilizada da maneira ideal para esse projeto. Inicialmente, foi projetado que a Intel Edison também se responsabilizaria pela etapa de Captura de Dados de um carro em funcionamento, criando um sistema completo que se comunicaria com o barramento CAN do veículo, processaria os dados e os enviaria a um servidor externo. No entanto, por não existirem bibliotecas disponíveis para o sistema da Edison que possibilitasse o uso do CAN-Shield, a alternativa de separar os sistemas surgiu como uma solução.

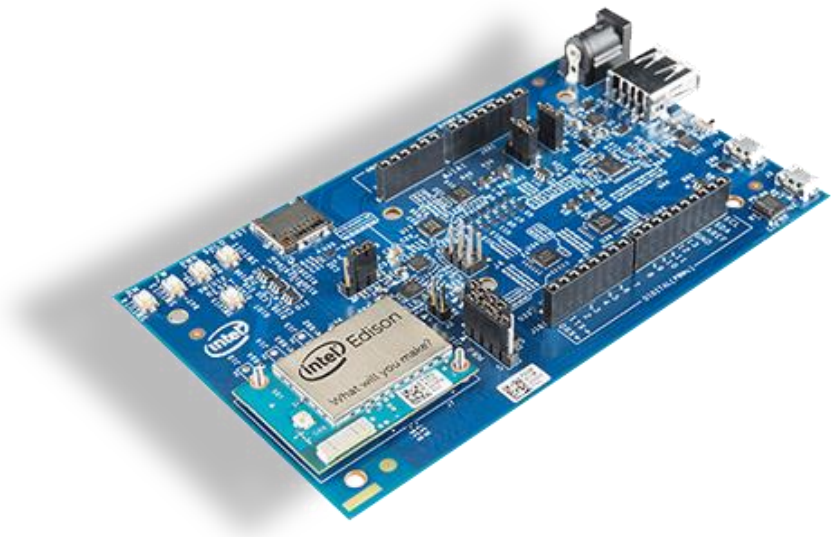


Figura 14. Intel Edison conectada à Arduino Breakout Board.

Processador	**Atom Dual-Core 500Mhz**
Memória RAM	**1GB DDR3**
Memória Flash	**4GB**
WiFi	**802.11 a/b/g/n Dual Band**
Bluetooth	**4.0**
I/O Pins	**Total de 40 Pinos** **SD Card** **UART (2)** **I2C (2)** **SPI** **...**
Sistema Operacional	**Yocto Linux v1.6**
Ambiente de Desenvolvimento	**Arduino IDE; Eclipse com suporte a C, C++ e Python; Intel XDK.**

Tabela 2. Especificações técnicas principais da placa Intel Edison.

4.4 Análises

Nesta seção será detalhado como os dados foram analisados para encontrar padrões que pudessem detectar eventos de frenagem e troca de marcha.

Como explicado em 3.3 os dados disponíveis até o momento são simples valores absolutos de velocidade instantânea, rotações por minuto e pressão no pedal de acelerador. Para auxiliar na análise dos dados, outras informações foram geradas, como os dados de acelerações, calculando uma lista de derivadas das velocidades e, além disso, dados de variação de RPM, também calculando as derivadas dos RPMs. Por fim, estas informações deveriam ser de algum modo analisadas, para que padrões fossem identificados e então servir como ponto de partida para detectar os eventos procurados. A abordagem escolhida para fazer essa análise foi transformar os dados dos logs em gráficos, como demonstra a Figura 15:

Figura 15. Gráficos dos dados originais gerados.

Na Figura 15 são ilustrados os três gráficos gerados com os dados colhidos, em que seus eixos das abcissas são dados em segundos, e os eixos das ordenadas são dados em rotações por minuto, velocida em km/h e porcentagem de de pressão no pedal do acelerador, respectivamente.

4.4.1 Análise das Frenagens

Para a detecção dos eventos de frenagem, foram utilizados os dados da velocidade instantânea, utilizando a informação do período de amostragem do sistema (0.6 segundos) para gerar uma lista com acelerações, em m/s^2. A Figura 16 demonstra o comportamento dos dados de velocidade e aceleração do log de teste:

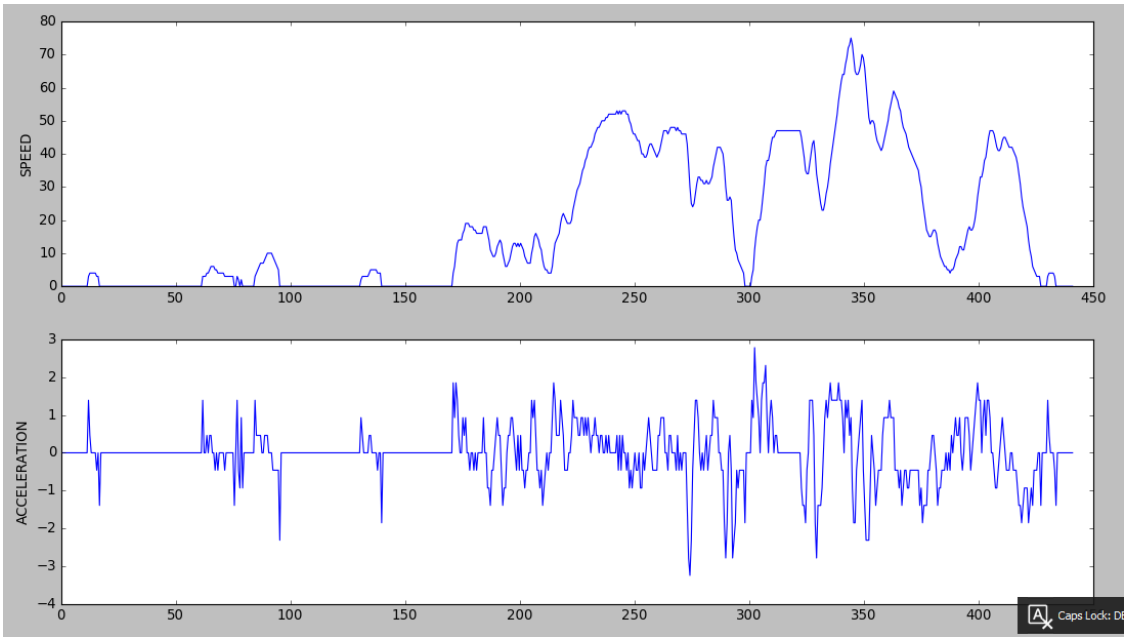


Figura 16. Gráficos de velocidade e aceleração.

Na Figura 16 são ilustrados os gráficos de velocidade, o qual já foi apresentado na Figura 15, e o gráfico de aceleração cujo eixo das ordenadas é dado em m/s^2.

Após a plotagem e análise dos gráficos foi necessário especificar quais padrões de dados identificariam uma frenagem, sendo definido que qualquer valor negativo de aceleração caracterizaria um evento onde o motorista estaria freiando.

O próximo passo foi classificar estas frenagens em grupos de acordo com o risco à integridade do veículo. Usou-se três tipos de classificações, demonstradas em Tabela 3 em ordem crescente de nível de ameaça à integridade do veículo.

Classificação	Intervalo
Green	Entre 0 m/s^2 e -1 m/s^2
Yellow	Entre -1 m/s^2 e -2 m/s^2
Red	Entre -2 m/s^2 e -10 m/s^2

Tabela 3. Classificações dos eventos de frenagem[3].

Estas classificações foram criadas empiricamente durante o trabalho ao observar os dados colhidos na etada de Captura de Dados, com o único intuito de separar diferentes tipos de frenagens.

[3] ***Os intervalos são fechados a esquerda e aberto à direita, ou seja, [-1, 0) por exemplo.

Durante a execução do algoritmo, cada frenagem identificada e classificada é guardada no sistema para ser utilizada na geração de dados estatísticos ao fim do percurso.

4.4.2 Análise das Trocas de Marcha

O processo de detecção dos eventos de troca de marcha foi realizado de modo semelhante ao utilizado no caso anterior. Foram utilizados todos os dados disponíveis e, a partir do período de amostragem de dados (0.6 segundos), as derivadas de RPM foram geradas para auxiliar na identificação de padrões que caracterizassem uma troca de marcha.

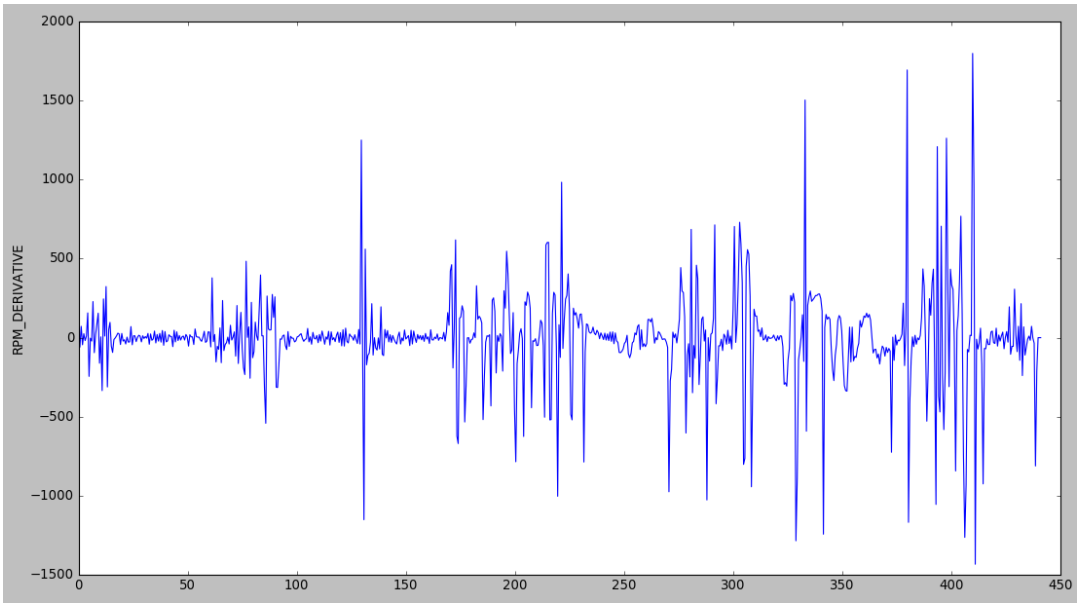


Figura 17. Derivada do RPM.

Na Figura 17 é ilustrado o gráfico de variação de RPM, cuja abcissa e ordenada são dados em segundos e RPM/segundo, respectivamente.

Durante a análise dos gráficos, percebeu-se que quando havia picos nos valores do RPM, também havia leve retenção nos valores de velocidade e depressões no gráfico de pressão do pedal do acelerador, representando o breve momento em que o motorista precisa trocar a marcha, tira o pé do acelerador, troca a marcha e volta a acelerar. Esse padrão pode ser observado na Figura 18:

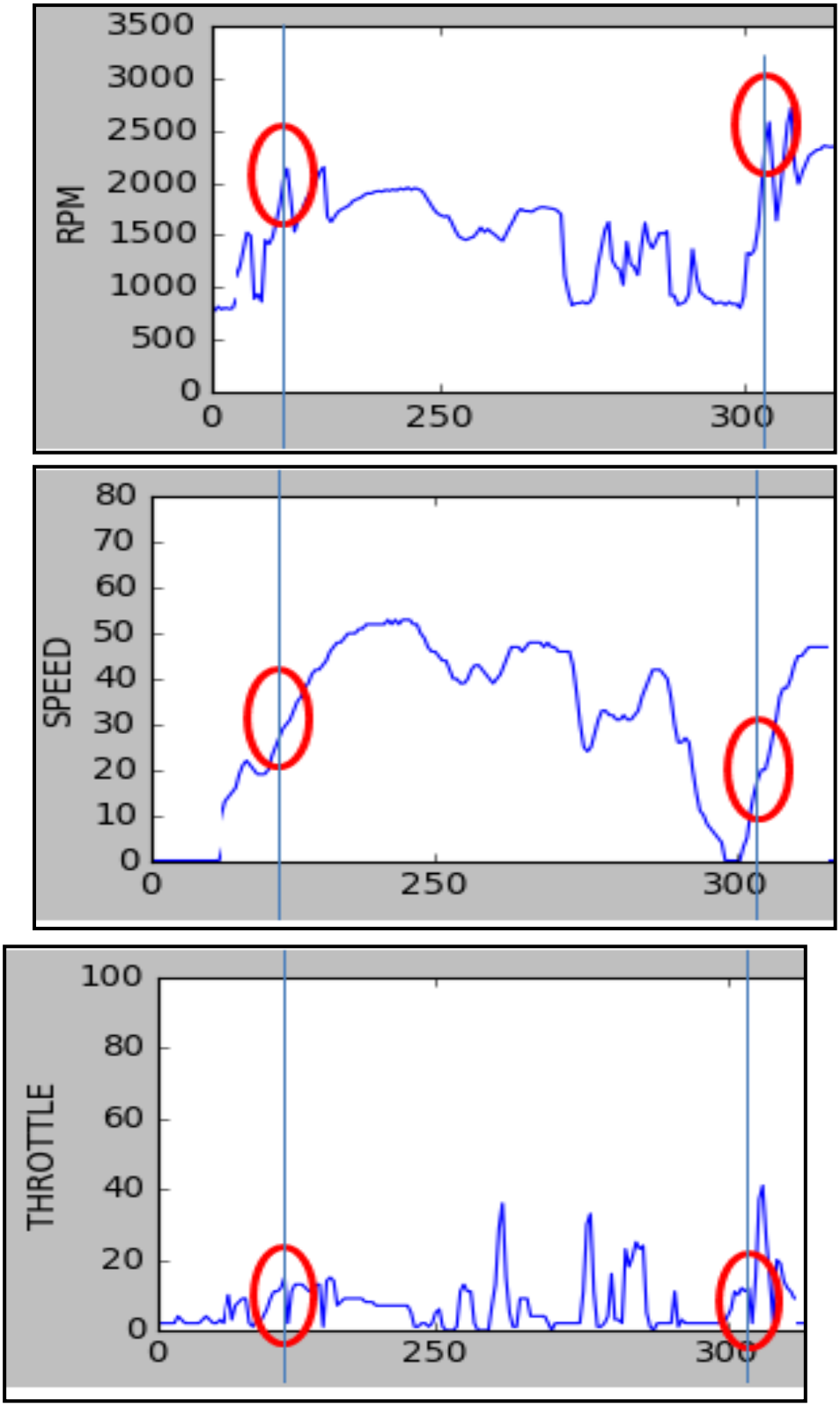


Figura 18. Análise das trocas de marcha.

Por esse motivo, decidiu-se que os eventos de troca de marcha seriam identificados como aqueles onde há picos de RPM, ou seja, momentos onde a derivada no RPM muda de positiva para negativa.

Após a identificação, o próximo passo foi classificar a troca de marcha, também como forma de pontuar o comportamento do motorista na condução do veículo. Utilizou-se os valores absolutos do RPM no momento da troca de marcha para criar três classificações, detalhadas em Tabela 4:

Classificação	Intervalo
Green	Entre 0 a 1000 RPM.
Yellow	Entre 1000 a 2000 RPM.
Red	Entre 2000 a 7000 RPM

Tabela 4. Classificações dos eventos de troca de marcha[4].

Estas classificações foram criadas empiricamente durante o trabalho ao observar os dados colhidos na etada de Captura de Dados, com o único intuito de separar diferentes tipos de trocas de marcha.

Durante a execução do algoritmo, cada troca de marcha identificada e classificada é guardada no sistema para ser utilizada na geração de dados estatísticos ao fim do percurso.

4.5 Relatório do Percurso

O próximo passo do processamento dos dados foi gerar dados estatísticos com os eventos de frenagem e troca de marcha, os quais serão utilizados para classificar o comportamento do motorista com o algoritmo de IA a ser explicado na próxima seção.

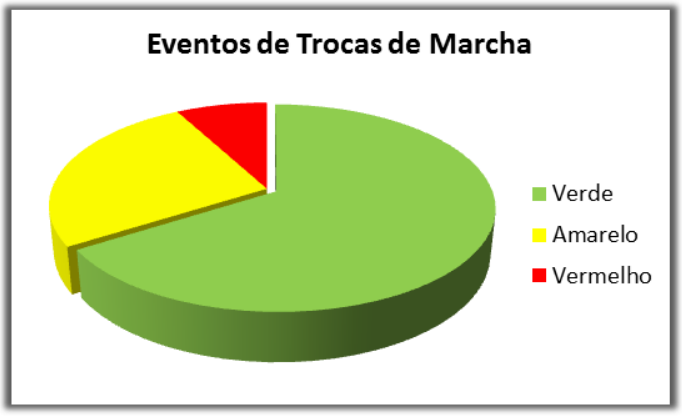
Este relatório do percurso consistiu em contabilizar os eventos e calcular a porcentagem que cada categoria representava em cada um dos casos de análise, frenagem e troca de marcha. A Figura 19 demonstra um exemplo da saída do sistema ao término do percurso:

```
Report das Frenagens: {'RED  - (-2.0, -10.0)': 1.99, 'GREEN - (0.0, -1.0)': 91.14, 'YELLOW - (-1.0, -2.0)': 6.87}
Report das Trocas de Marcha: {'GREEN - (0, 1000)': 66.1, 'RED - (2000, 7000)': 7.91, 'YELLOW - (1000, 2000)': 25.99}
(edison)locus@ubuntu:~/Projetos/Edison$
```

Figura 19. Exemplo da saída do Relatório do Percurso.

[4] ***Os intervalos são fechados a esquerda e aberto à direita, ou seja, [0, 1000) por exemplo.

Classificação	Porcentagem de Eventos
Verde	91,14%
Amarelo	6,87%
Vermelho	1,99%



Classificação	Porcentagem de Eventos
Verde	66,1%
Amarelo	25,99%
Vermelho	7,91%

Figura 20. Gráfico de setores com estatísticas dos eventos.

Ao término do algoritmo, a saída mostrada na Figura 19 é enviada ao servidor web para ser mostrada ao usuário de maneira formatada.

4.6 Aplicação de Inteligência Artificial

Este é o último processamento por onde passam as informações coletadas no barramento CAN, em que o comportamento do motorista ao volante é classificado de maneira automatizada através de um algoritmo de Aprendizagem de Máquina. Para que esse processamento pudesse ser incorporado ao sistema, porém, foi necessário que um banco de dados com instâncias de treinamento fosse criado, o que é detalhado em 4.6.1. A partir daí, a classificação pôde ser aplicada utilizando o algoritmo kNN, detalhado em 2.3.1.

4.6.1 Criação do Banco de Dados

A criação do banco de dados é uma atividade essencial para que a classificação possa ser implementada. Como mencionado em 2.3.1, o algoritmo escolhido (kNN) é extremamente dependente da existência de um banco de dados contendo um conjunto de treinamento devidamente mensurado e classificado. Por esse motivo e pela inexistência de um banco desse tipo que seja útil para a finalidade desse projeto, foi necessário criá-lo e classificá-lo. Vale lembrar que, por ter sido criado um banco de dados sintético, existiu uma preocupação em garantir que esse banco fosse fiel à realidade. A única restrição necessária para garantir que esse banco de dados corresponda à realidade está na totalização dos valores presentes nos eventos de frenagem e troca de marcha. Em cada um dos tipos de evento, os valores de suas três classes deveriam totalizar 100%.

Primeiramente, vale lembrar que as características de uma instância, ou de um percurso, são definidas pelos dados mostrados na Figura 19, em que temos os dados estatísticos do percurso. Nesses dados, temos as porcentagens referentes a cada uma das classificações em cada caso de análise: frenagens e trocas de marcha. Portanto, para a criação de um banco de dados de treinamento, foram criadas, via script Python, 200 entradas com valores aleatórios que atendessem todos os requisitos de uma saída real, e então classificadas com a ajuda de colaboradores. Na Figura 21, segue uma imagem da planilha que ficou disponível na nuvem para que os colaboradores pudessem classificar:

	A	B	C	D	E	F	G	H	I	J
1	break_green(0 -> -1 m/s2)	break_yellow(-1 -> -2 m/s2)	break_red(-1 -> -2 m/s2)		gear_green(0 -> 1000 RPM)	gear_yellow(1000 -> 2000 RPM)	gear_red(2000 -> 7000 RPM)	CLASSIFICAÇÃO		
2	77.93052339	6.225819909	15.8436567		34.40941079	18.07591659	47.51467262			
3	51.22819136	19.87760662	28.89420202		39.79037229	43.74146365	16.46816406			
4	35.43913652	4.556943491	60.00391999		90.27509737	7.323845248	2.401057386			
5	66.50459073	32.50537391	0.990035356		85.92340566	10.97667279	3.099921546			
6	67.26455392	17.17920345	15.55624263		85.37301566	8.240090064	6.386894278			
7	42.90706777	7.304956022	49.78797621		86.1794011	9.289446511	4.531152392			
8	87.32815053	0.4687536759	12.2030958		39.42168153	50.16606586	10.41225262			
9	52.70865755	25.52143306	21.76990939		34.15293839	43.09970855	22.74735306			
10	86.38642511	12.04866868	1.564906217		34.50756577	25.17293283	40.3195014			
11	69.7877295	4.109177863	26.10309264		56.48414862	24.60781004	18.90804135			
12	74.64810437	6.722992173	18.62890345		28.14265088	32.55816333	39.29918579			
13	9.317161573	30.02807708	60.65476135		19.58811759	58.7901097	21.6217727			
14	14.52124946	82.91164491	2.567105634		7.506198174	70.31502529	22.17877654			
15	43.15244815	20.76584181	36.08171004		29.54995099	35.33416046	35.11588855			
16	27.14474308	66.1367728	6.718484122		36.22195274	49.00638115	14.77166612			
17	23.94713152	20.26068339	55.79218509		46.75426682	40.30526354	12.94046964			
18	90.06231584	9.308258989	0.6294251719		75.46692709	12.99962021	11.5334527			
19	78.23354723	9.882017301	11.88443547		87.39509949	4.986312841	7.618587665			
20	89.04068254	0.1828772399	10.77644022		72.88686041	20.73710836	6.376031226			

Felipe:
Classificar de 1 a 5, do mais cauteloso ao mais barbeiro no trânsito. São dois grupos de dados, onde cada grupo soma 100%. O primeiro grupo é de frenagens, indo das frenagens leves (break_green) até as grenagens bruscas (break_red), quanto maior o número, mais número daquele tipo de frenagem

Classificar de 1 a
São dois grupos

Figura 21. BD de treinamento.

Nas três primeiras colunas estão as porcentagens referentes às três classes de frenagens, com seus valores totalizando 100%. O mesmo acontece nas outras três colunas, em que estão as porcentagens referentes às três classes de trocas de marcha, também totalizando 100%. E por último, a coluna de **classificação**, onde os colaboradores puseram a classe do motorista de acordo com os dados estatísticos do percurso, e que será melhor detalhada em 4.6.2.

4.6.2 Classificação do Comportamento do Motorista

A classificação do comportamento do motorista se resumiu em definir classes nas quais um percurso se enquadraria, classificando desde motoristas mais cautelosos até os mais atrevidos.

Inicialmente, foi necessário definir o elemento de análise em seu vetor de características, como já mencionado em (2.1). Vale lembrar também que, cada percurso é definido pelos tipos de frenagens e trocas de marcha que obteve, como ilustrado na Tabela 3 e na Tabela 4. Cada elemento de análise contou, portanto, com as 6 (seis) características do percurso do motorista em seu vetor, como mostrado na Tabela 5:

Índice do Vetor	Característica (%)
a_1	Frenagens GREEN
a_2	Frenagens YELLOW
a_3	Frenagens RED
a_4	Trocas de Marcha GREEN
a_5	Trocas de Marcha YELLOW
a_6	Trocas de Marcha RED

Tabela 5. Vetor de Características do Sistema.

Definido o vetor de características a ser utilizado pelo sistema, cada percurso finalizado é, então, interpretado como um vetor de valores (a_1, a_2, a_3, a_4, a_5, a_6) o qual possui todos os dados necessários para a classificação do percurso.

A próxima etapa do processo de classificação é definir as classes existentes nas quais os elementos de análise devem se enquadrar. Foram criadas 5 classes distintas, numeradas de 1 a 5, sendo 1 para o motorista com melhor abordagem ao volante e 5 para aquele mais negligente, como mostra Tabela 6 :

Classe	Descrição da Classe
1	Ótimo
2	Bom
3	Regular
4	Ruim
5	Péssimo

Tabela 6. Classes do Knn.

Por último, segundo 2.3.1, é necessário definir o valor de **K**, que indica a quantidade de vizinhos mais próximos a ser considerada pelo sistema. Foi definido que o valor de **K** seria 3, ou seja, ao calcular a Distância Euclidiana (2.2) entre elemento de análise e os elementos do banco de dados, seriam escolhidos para análise apenas os três elementos mais próximos.

Finalmente, com todas as distâncias calculadas e os três elementos de treinamento mais próximos do elemento de análise encontrados, o sistema analisa a classe destes três elementos. Aquela classe com maior incidência dentre eles será escolhida como a classe do elemento de análise. No caso de teste, exemplificado em Figura 20, a classificação do sistema para este percurso foi (1, Ótimo), como mostra a Figura 22:

Figura 22. Exemplo de Classificação do Motorista.

O pseudo código desta etapa está presente em 8.2.

Capítulo 5

Interface com Usuário

Este capítulo irá detalhar a Interface com o Usuário que foi criada para este projeto para disponibilizar os dados processados em uma interface web, possibilitando acesso via browser ou smartphone.

5.1 Introdução

Esta etapa do projeto foi responsável pela interface com o usuário do sistema, oferecendo os dados já formatados em um site na web. Através desse site, o usuário pôde acompanhar os dados sendo processados em tempo real, recebendo feedbacks a respeito do seu comportamento ao dirigir e, ao final, visualizar sua performance por completo com as estatísticas mencionadas em 4.5. Para a construção do servidor, foi utilizado um framework para criação de aplicações web chamado Django, e explicado na próxima seção.

5.2 Django

Django é um framework para a criação de aplicações web na linguagem Python. Segundo o site oficial da ferramenta, “Django é um framework web em Python de alto nível que encoraja o desenvolvimento rápido e limpo. Construído por desenvolvedores experientes, Django realiza automaticamente muito do retrabalho exigido pelo desenvolvimento web para que o desenvolvedor possa focar no sistema sem ter que reinventar a roda.” [10].

5.3 Disponibilização dos Dados

Essa seção descreve como a disponibilização dos dados processados pela Intel Edison foi implementada, explicando desde a comunicação entre a Intel Edison e servidor, até as requisições realizadas pelo usuário via browser.

5.3.1 Comunicação Intel Edison e Servidor Web

A primeira etapa do processo de comunicação do sistema se responsabilizou em oferecer um meio pelo qual o embarcado, representado pela Intel Edison, pôde enviar os dados processados ao servidor web. Para que esse objetivo fosse atingido, utilizou-se o módulo WiFi da Edison, cujas especificações podem ser vista na Tabela 7:

WiFi	Broadcom* 43340 802.11 a/b/g/n; Dual-band (2.4 and 5 GHz) Onboard antenna

Tabela 7. Informações técnicas do módulo WiFi da Intel Edison.

Com o módulo WiFi completamente configurado e funcional, o sistema utiliza o protocolo de comunicação da camada de maior abstração do modelo OSI, enviando mensagens HTTP ao servidor web a cada ciclo do sistema, ou seja, a cada período de amostragem dos dados, 0.6 segundos. Portanto, o servidor deve receber mensagens indicando frenagem e troca de marcha a cada ciclo, caso estas tenham ocorrido.

5.3.2 Dados na Web

O próximo passo na etapa de comunicação tratou de disponibilizar os dados ao usuário final, em qualquer ambiente em que o mesmo estivesse. Para isso, foi desenvolvido um sistema de arquivos, onde o servidor se responsibiliza em salvar os dados recebidos da Intel Edison em arquivos *.txt*. Então, sempre que uma requisição acontece vinda do browser, esse arquivo é visitado e a primeira entrada é retornada ao usuário e renderizada na página. Esse processo continua a acontecer até que não haja mais dados no arquivo .txt ou que a comunicação com a Intel Edison tenha sido encerrada. Para que esse processo possa ser melhor entendido, a Figura 23 foi disponibilizada, ilustrando o diagrama do funcionamento do sistema:

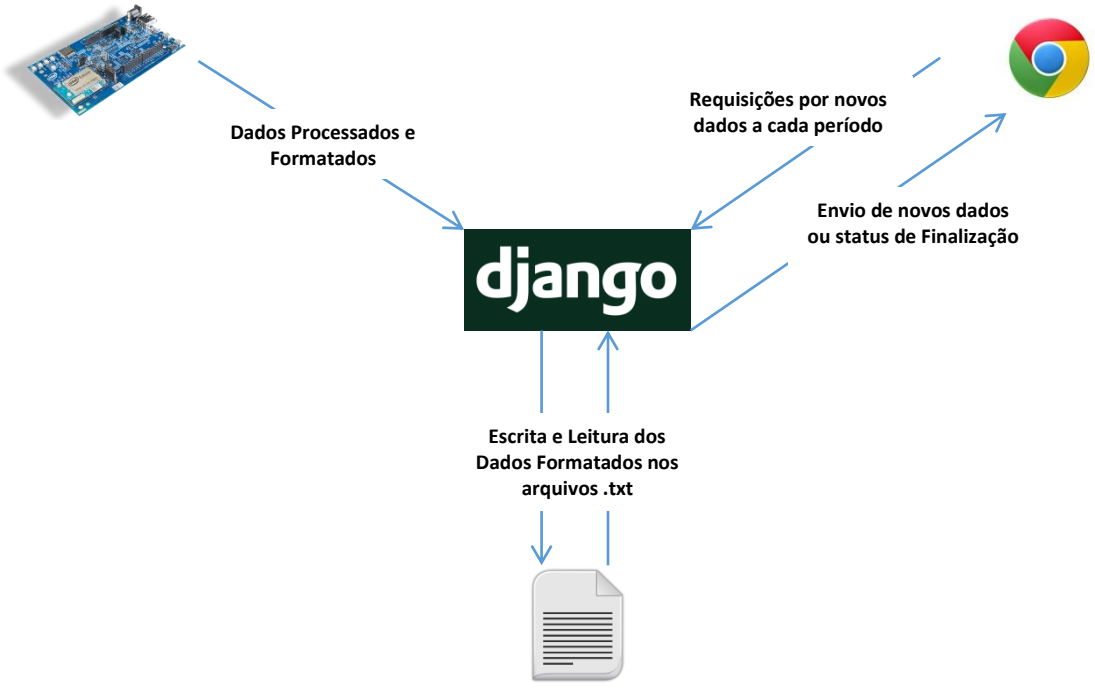


Figura 23. Diagrama de funcionamento do sistema de comunicação.

Esse processo continua indefinidamente desde que o usuário esteja na página da aplicação. Além disso, o sistema precisa estar com status de funcionamento, em que o percurso ainda não foi finalizado. Uma vez finalizado o percurso, o browser recupera uma última informação contendo as estatísticas do percurso, assim como a classificação do motorista, e para de requisitar o servidor.

5.4 Telas do Sistema

Esta seção tem como objetivo disponibilizar e explicar as telas que foram desenvolvidas para mostrar os dados formatados para o usuário final.

5.4.1 Tela de Acompanhamento em Tempo Real

Esta é tela principal do sistema, onde o usuário consegue acompanhar os eventos de frenagem e troca de marcha em tempo real. Esta tela é responsável por requisitar as informações mais atuais do servidor a cada período. Portanto, a cada 0.6 segundos, a tela requisita as frenagens mais atuais juntamente com as trocas de marcha, e separa o resultado dessas requisições em duas seções. Na Figura 24, pode-se ver um exemplo desta tela em funcionamento, onde as informações de frenagens são inseridas na caixa à esquerda, enquanto que as trocas de marcha estão na caixa à direita:

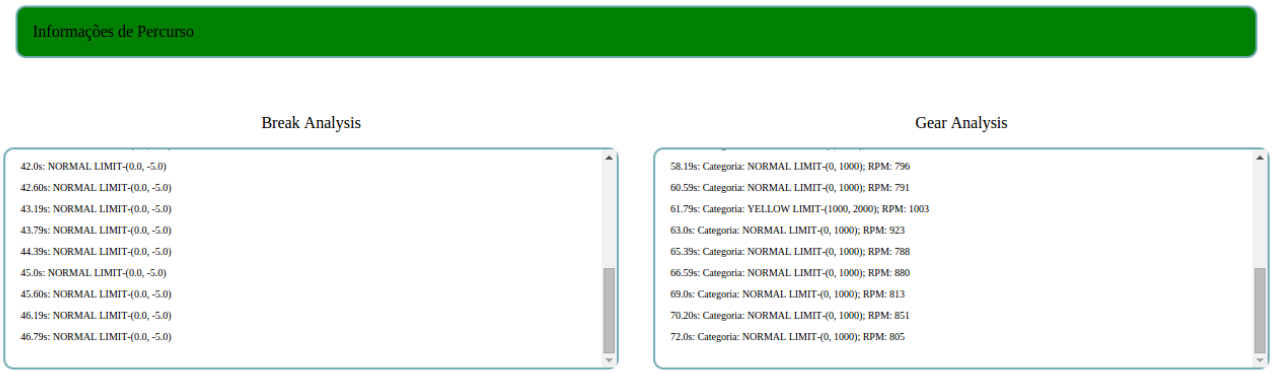


Figura 24. Tela do Acompanhamento em Tempo Real.

5.4.2 Tela de Relatório de Percurso

A tela de Relatório de Percurso é a tela final do sistema, onde são mostrados os dados estatísticos do percurso, os mesmos que foram exemplificados na Figura 19, juntamente com a classificação do motorista dada pelo Algoritmo de Aprendizagem de Máquina. Segue na Figura 25 um exemplo dessa tela:

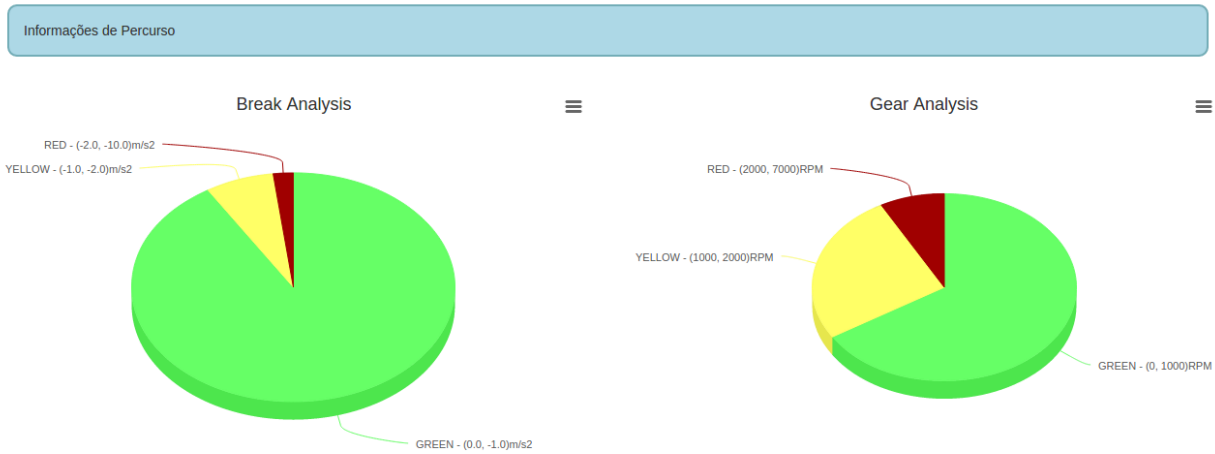


Figura 25. Tela final com os dados estatísticos do percurso.

5.4.3 Tela de Classificação do Percurso

A tela de Classificação do Percurso também faz parte da tela final do sistema e é responsável por mostrar a classificação do percurso de acordo com o algoritmo de Aprendizagem de Máquina KNN explicado em 4.6. Nessa tela, é mostrado o resultado final da classificação detalhada em 4.6.2. Segue na Figura 26 um exemplo da Tela de Classificação:

Figura 26. Tela da classificação do motorista.

Na parte superior da tela é possível ver a classificação do motorista medida em estrelas, sendo uma estrela para o motorista menos cauteloso e cinco para o motorista com uma maneira mais adequada de direção.

Capítulo 6

Trabalhos Futuros

Este capítulo tem como objetivo explicitar possíveis trabalhos a serem desenvolvidos no futuro com o intuito de incrementar este projeto.

6.1 Unificação dos Sistemas Embarcados

A unificação dos Sistemas Embarcados consistiria em unir as etapas de Captura de Dados no Barramento CAN (Capítulo 3) e de Processamento de Dados (Capítulo 4) em uma plataforma única, como já mencionado neste documento.

Inicialmente, havia sido projetado que a plataforma Intel Edison seria responsável por essas duas etapas. No entanto, não foi possível realizar a a integração entre Intel Edison e CAN-Shield Seeed por esta última não possuir qualquer biblioteca já implementada na liguagem Python para a Intel Edison. A implementação desta biblioteca surgiu como um possível trabalho a ser feito como Trabalho de Conclusão de Curso, no entanto, achou-se mais interessante um sistema que de fato gerasse novas informações ao utilizar os dados que trafegam em uma rede intraveicular.

6.2 Detecção de Outros Eventos de Direção

A proposta desse trabalho seria implementar outras detecções de eventos além de frenagens e trocas de marcha utilizando as mensagens CAN disponíveis. A detecção de outros eventos de direção entregaria mais informações estatísticas sobre a forma com a qual o motorista dirige, além de fornecer mais dados ao processo de classificação com IA.

Inicialmente, pensou-se em vários outros eventos de direção que poderiam ser detectados com o objetivo de inferir quão seguro é o comportamento do motorista diante do volante. Além de frenagens e troca de marcha, estavam:

- **Lane-Drifting:** Tem como objetivo detectar o comportamento do motorista ao mudar de faixa ou ao realizar uma conversão. É comum observar que alguns motoristas possuem o hábito de não utilizar a seta para alertar sobre a mudança de direção do veículo, utilizando-a tardiamente, ou até mesmo negligenciando seu uso. A implementação deste caso dependeria de informações a respeito da posição das rodas ou volante, além de informações a respeito do acionamento das setas. Assim, é possível calcular uma relação de tempo entre elas,

concluindo quão antes o motorista costuma alertar sobre a mudança de direção do veículo.

- **Velocidade em curvas:** Tem como objetivo analisar quão veloz o motorista costuma realizar curvas, detectando quando estes eventos acontecem em altas velocidades. A implementação deste caso também dependeria da informação a respeito da posição das rodas ou volante, além da velocidade instantânea que já é recuperada e mencionada neste trabalho.

Estes eventos não puderam ser detectados durante este projeto pela indisponibilidade das informações relativas à posição das rodas, volante e setas. Como pode ser visto em [4], apenas os IDs de algumas mensagens CAN são disponibilizadas por padrão. Cada montadora possui seu conjunto de IDs de mensagens CAN, não disponibilizando-o gratuitamente à comunidade. Portanto, pra implementar esses casos mencionados acima, seria necessário ter acesso a esses conjuntos proprietários de IDs de mensagens CAN.

6.3 Fornecimento de Dados a Empresas

Este trabalho também poderia ser utilizado comercialmente, fornecendo dados a respeito do perfil de condutores para empresas como seguradoras ou empresas de logística. Com uma classificação a respeito de trechos conduzidos por um motorista, seria possível traçar o perfil do mesmo, identificando os principais defeitos em seu estilo de condução, além de assegurar que ele seja cobrado de maneira mais condizente à ameaça que oferece ao veículo.

6.4 Refinamento da Classificação de Frenagens e Trocas de Marcha

Um próximo passo para este projeto poderia envolver o trabalho em conjunto com profissionais de Engenharia Mecânica, cuja expertise poderia trazer mais informações a respeito de como separar os diferentes tipos de frenagens e trocas de marcha. Além disso, estes profissionais poderiam oferecer mais idéias a respeito de quais outros fatores também levam um motorista a ameaçar a integridade mecânica do veículo que dirigem.

Capítulo 7

Conclusão

Após a realização deste trabalho, concluiu-se que sistemas capazes de analisar os dados presentes nas redes intraveiculares e processá-los podem vir a ser uma oportunidade de negócio, uma vez que estes dados podem conter bastante informação a respeito do motorista e do carro, os quais podem ser de interesse de grandes empresas do ramo de logistíca e seguros. Além disso, considerando que dados importantes do motorista foram gerados a partir de um conjunto limitado de mensagens CAN, a quantidade de dados a serem gerados pode aumentar consideravelmente quando em posse das tabelas proprietárias de mensagem CAN, aumentando o potencial de projetos deste tipo.

Capítulo 8

Anexos

Este capítulo tem como objetivo disponibilizar os anexos referentes aos códigos utilizados no sistema. Todo o código presente neste capítulo está no formato de pseudo-código por motivos de complexidade e tamanho.

8.1 Pseudo-Código da Captura de Dados

Nesta seção está o pseudo-código utilizado para a captura dos dados no barramento CAN:

```
//Initializa o monitor serial e a CAN Shield.
initialize_peripherals()

// Checa os valores das variaves no CAN Bus e imprime no Monitor Serial.
while there_is_power(){

    // Checa o RPM.
    check_rpm();
    // Imprime o RPM.
    print rpm;

    // Checa a velocidade.
    check_speed();
    // Imprime a velocidade.
    print speed;

    // Checa a pressão no pedal do acelerador.
    check_accelerator_pedal();
    // Imprime a pressão no pedal do acelerador.
    print accelerator_pedal;

    delay(600);
}
```

8.2 Pseudo-Código do Processamento de Dados

Nesta seção está o pseudo-código utilizado para a processar os dados:

```
PERIOD = 0.6
LOG_PATH = "Caminho com o arquivo de log"

# Cria um Dict com as entradas de SPEED, RPM e PEDAL_ACCELERATOR presentes
no arquivo.
# Este dict é criado após a leitura do log com os dados da Captura.
""" log_dict = {
    'SPEED': [list_of_speed_values],
    'RPM': [list_of_rpm_values],
    'PEDAL_ACCELERATOR': [list_of_pedal_accelerator_values]
}
"""
log_dict = set_log_dict(LOG_PATH)

# Salva o valor de iterações do sistema.
route_length = smaller_list_length_in_dict(log_dict)

# Cria as entradas RPM_DERIVATIVE e ACCELERATION no dict de entradas.
log_dict['RPM_DERIVATIVE'] = []
log_dict['ACCELERATION'] = []

# Cria as listas com os eventos de Frenagem e Troca de Marcha.
break_events = []
shifting_gear_events = []

for item in range(route_length):

    # Inclui a leitura momentânea dos dados em um Dict.
    current_data = {
        'SPEED': log_dict['SPEED'][item],
        'RPM': log_dict['RPM'][item],
        'PEDAL_ACCELERATOR': log_dict['PEDAL_ACCELERATOR'][item]
    }

    # Descobre o valor da derivada do RPM naquele momento.
    current_data['RPM_DERIVATIVE'] = proccess_rpm_derivative(current_data,
item, log_dict)
    # Salva o valor da derivada do RPM no histórico.
    log_dict['RPM_DERIVATIVE'].append(current_data['RPM_DERIVATIVE'])

    # Descobre o valor da aceleração naquele momento.
    current_data['ACCELERATION'] = (proccess_acceleration(current_data,
item, log_dict))
    # Salva o valor da aceleração no histórico.
    log_dict['ACCELERATION'].append(current_data['ACCELERATION'])
```

```
    # Detecta o evento de Frenagem naquele momento.
    break_event = detect_break_event(current_data, log_dict, item)

    # Detecta o evento de Troca de Marchas naquele momento.
    shifting_gear_event = detect_shifting_gear_event(current_data,
log_dict, item)

    # Se houver algum evento de frenagem, ele é enviado para o Servidor Web
    # e salvo na lista de eventos de frenagem.
    if break_event is not None:
        send_data_to_web_server(break_event)
        break_events.append(break_event)

    # Se houver algum evento de troca de marcha, ele é enviado para o
    # Servidor Web e salvo na lista de eventos de troca de marcha.
    if shifting_gear_event is not None:
        send_data_to_web_server(shifting_gear_event)
        shifting_gear_events.append(shifting_gear_event)

# Ao término do percurso, os dados são analisados por Inteligência
# Artificial para classificar o motorista.
driver_analysis = analyse_driver_data
                  (break_events=break_events,
                   shifing_gear_events=shifting_gear_events)

# Os dados com a análise de IA são enviados para o servidor Web para
# mostrar a classificação do motorista.
send_data_to_web_server(driver_analysis)
```

Referências

[1] MITCHELL, Tom. **Machine Learning**. McGraw-Hill, 1997.

[2] DUDA, Richard O.; HART, Peter E.; STORK, David G. **Pattern Classification**. Second Edition. Wiley-Interscience, 2000.

[3] CORRIGAN, Steve. **Introduction to the Controller Area Network (CAN).** Texas Instruments, 2008.

[4] Wikipedia, (2016). **OBD-II PIDs**. [Online] Disponível em: https://en.wikipedia.org/wiki/OBD-II_PIDs [Acessado 17 Jan. 2016].

[5] FIELDING, GETTYS, MOGUL; **Hypertext Transfer Protocol -- HTTP/1.1.** [Online] Disponível em: https://www.w3.org/Protocols/rfc2616/rfc2616.txt [Acessado 13 Jan. 2016].

[6] Wikipedia, (2016). **Packet Analyzer**. [Online] Disponível em: https://en.wikipedia.org/wiki/Packet_analyzer [Acessado 13 Jan. 2016].

[7] Arduino, (2016). **Arduino Uno**. [Online] Disponível em: https://www.arduino.cc/en/Main/ArduinoBoardUno [Acessado 14 Jan. 2016].

[8] Seeed, (2016). **CAN-Bus Shield**. [Online] Disponível em: http://www.seeedstudio.com/wiki/CAN-BUS_Shield#Getting_Started [Acessado 14 Jan. 2016].

[9] Intel, (2016). **Intel Edison**. [Online] Disponível em: http://download.intel.com/support/edison/sb/edison_pb_331179002.pdf [Acessado 14 Jan. 2016].

[10] Django Project, (2016). **Django**. [Online] Disponível em: https://www.djangoproject.com/ [Acessado 17 Jan. 2016].

[11] VALASEK, MILLER, (2014). **Adventures in Automotive Networks and Control Units**. [Online] Disponível em: http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf [Acessado 25 Jan. 2016].

[12] IPEA, (2014). **Acidentes de trânsito nas rodovias federais brasileiras**. [Online] Disponível em: http://www.ipea.gov.br/portal/images/stories/PDFs/relatoriopesquisa/150922_relatorio_acidentes_transito.pdf [Acessado 09 Abr. 2016].