UNIVERSIDADE FEDERAL DO PARANÁ

PEDRO MANTOVANI ANTUNES

MAPEAMENTO DA QUALIDADE DE PAVIMENTOS POR SENSORES, SISTEMAS EMBARCADOS E INTELIGÊNCIA ARTIFICIAL

CURITIBA

2018

PEDRO MANTOVANI ANTUNES

MAPEAMENTO DA QUALIDADE DE PAVIMENTOS POR SENSORES, SISTEMAS EMBARCADOS E INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados da Universidade Federal do Paraná como requisito à obtenção do título de Engenheiro Eletricista
Orientador: Prof. Dr. Carlos Marcelo Pedroso

CURITIBA
2018

TERMO DE APROVAÇÃO

PEDRO MANTOVANI ANTUNES

MAPEAMENTO DA QUALIDADE DE PAVIMENTOS POR SENSORES, SISTEMAS EMBARCADOS E INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados da Universidade Federal do Paraná como requisito à obtenção do título de Engenheiro Eletricista, pela seguinte banca avaliadora:

_____

Prof. Dr. Carlos Marcelo Pedroso
Orientador – Departamento de Engenharia Elétrica, UFPR

_____

Prof. M.Sc. Bruno Pohlot Ricobom
Departamento de Engenharia Elétrica, UFPR

_____

Prof. Dr. Wilson Arnaldo Artuzi Junior
Departamento de Engenharia Elétrica, UFPR

Curitiba, 04 de dezembro de 2018

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Sandra e Josmar, que sempre foram meus exemplos de vida, à minha irmã Janaina e também a toda minha família, pelo amor, apoio incondicional e paciência durante toda esta jornada, que foram fundamentais para me tornar uma pessoa melhor e fazer deste momento uma realidade.

Aos meus amigos, por todos os momentos e parcerias de longa data, e por sempre me manterem motivado nos meus objetivos de vida.

Aos professores inspiradores que tive o prazer de conhecer, por todo o empenho, dedicação e conhecimentos compartilhados, e mais especificamente ao meu professor orientador Carlos Marcelo Pedroso, por todo o auxílio durante a elaboração deste trabalho.

RESUMO

A má preservação de pavimentos em vias públicas é um problema que vem se agravando com a expansão da mobilidade urbana. Apesar disso, ainda há poucas informações sobre as condições dos pavimentos nas cidades e rodovias brasileiras, e estas poucas informações são obtidas manualmente. Este trabalho consiste no projeto e desenvolvimento de um sistema a ser embarcado em automóveis que, através de sensores e algoritmos de inteligência artificial, seja capaz de avaliar a atual condição do pavimento. Para tal, foram empregados acelerômetros, giroscópios e GPS (*Global Positioning System*) operando conjuntamente com uma SVM (*Support Vector Machines*). Para o treinamento e validação do sistema proposto, foi utilizado um conjunto de 11 horas de dados coletados, separando as estradas em 2 e 3 níveis de qualidade. O modelo que apresentou o melhor desempenho em ambos os cenários foi com a aplicação do *kernel* RBF (*Radius Basis Function*). Foi avaliada também uma metodologia para tornar desnecessária uma orientação específica dos sensores. Mesmo neste cenário mais adverso de uso, o classificador apresentou resultados suficientemente bons para implantação em um sistema de classificação em tempo real, o que indica que o método proposto pode ser implementado com sucesso em dispositivos como *smartphones* com disponibilidade de *hardware* apropriado. Finalmente, foi desenvolvida uma plataforma na *Internet* para armazenamento das avaliações realizadas, que são enviadas do sistema embarcado através de um módulo GSM. As coordenadas geográficas são enviadas juntamente com a avaliação do classificador, o que possibilita a fácil visualização das informações de qualidade de pavimentos em uma determinada região.

Palavras-chave: inteligência artificial, SVM, acelerômetros, IoT, GPS, rodovias, pavimentos.

ABSTRACT

The preservation of pavements in public roads is an issue that has been aggravating with the expansion of urban mobility. Nevertheless, there is still few information about pavement condition on Brazilian cities and roads, and these few information were all gathered manually. This work consists of the design and development of a car-embedded system that, through sensors and artificial intelligence algorithms, is able to assess the current pavement condition. To do so, accelerometers, gyroscopes and GPS (Global Positioning System) were deployed working conjointly with a SVM (Support Vector Machines). For the training of the SVM, a dataset of more than 11 hours of sampled data was used. The samples were manually classified into 2 and 3 quality levels. Better performance was achieved by using a model with a RBF (Radial Basis Function) kernel. A methodology to prevent the need for a specific orientation of the sensors was also assessed. Even in this more several conditions of use, the classifier showed sufficiently good results for deployment of a real-time classifier system. Finally, an online platform was developed for storing the detections, which are sent from the embedded system through a GSM module. The geographical coordinates are sent together with the classification, which enables an user-friendly visualization of the road quality information.

Keywords*: artificial intelligence, SVM, accelerometers, IoT, GPS, roads, pavements.*

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A condição satisfatória de estradas e rodovias é fundamental para uma condução segura e tranquila. Os buracos em autovias podem originar acidentes fatais, diminuir a velocidade média da estrada, além de potencialmente causar danos à estrutura dos automóveis. Este fator é ainda mais agravante no Brasil, onde os motoristas gastam em média 26% a mais na manutenção de veículos por conta das condições de pavimentação em comparação à outros países (CNT, 2016). Segundo a Confederação Nacional do Transporte, cerca de 48% das rodovias do país possuem pavimento avaliado como regular, ruim ou péssimo (CNT, 2016). Ainda assim, o transporte rodoviário é o mais utilizado no Brasil, representando 60% de todos os transportes, enquanto o ferroviário possui um índice de 20%, hidroviário 13% e aéreo 4% (ESCOLARES, 2015).

Com um índice de utilização tão elevado das rodovias, a manutenção de condições adequadas do pavimento para tráfego de veículos é essencial para a segurança dos condutores e pedestres, uma vez que defeitos nas vias estão entre as principais causas de acidente de trânsito no mundo. Ainda, segundo a Organização Mundial da Saúde (OMS), o Brasil está em quinto lugar no ranking de mortes por acidente de trânsito (METRO, 2017).

Agregando esta análise com o fato de que as rodovias não pavimentadas representam cerca de 90% da malha rodoviária no país, são apresentados argumentos suficientes para entender a necessidade de um mapeamento estratégico da qualidade e de irregularidades das vias públicas no Brasil (CNT, 2018). Atualmente, isto é realizado de forma manual e apenas no âmbito das rodovias federais pelo CNT (Confederação Nacional dos Transportes). Não obstante, devido à larga malha rodoviária do Brasil (1,5 milhão de km), apenas 7% dessa malha rodoviária é avaliada (CNT, 2018). Uma oportunidade de automatizar este processo e realizá-lo de forma eficiente poderá diminuir acentuadamente os custos, aumentar muito a gama de vias avaliadas e diminuir o tempo de avaliação.

Este mapeamento seria benéfico para muitas partes. Para os cidadãos que convivem com vias em condições ruins, a transparência das avaliações de qualidade de pavimento pode servir como forma de pressionar os órgãos públicos a realizar obras de reparo. Para os órgãos públicos, isto pode auxiliar a planejar os locais e

prioridades destas obras. Além disso, danos a veículos em razão de falhas na pista é de responsabilidade da administração pública (PEIXOTO; SILVA, 2011).

Com os dispositivos e ferramentas hoje disponíveis pela eletrônica e computação, é possível desenvolver uma maneira automática de mapeamento, com mínima a nenhuma intervenção manual no processo. Ademais, os resultados gerados por estas avaliações podem ser disponíveis *online* para consulta pública. Assim, o escopo deste projeto será desenvolver e implementar um sistema integrado utilizando sensores e microcontroladores que comuniquem-se com a *Internet* para o cumprimento de tal tarefa.

1.1 OBJETIVOS

O tamanho da malha rodoviária juntamente à má condição das estradas brasileiras mostra uma potencial uma demanda para um sistema capaz de mapear a qualidade dos pavimentos nas vias brasileiras. Prevê-se aqui um *hardware* que poderá ser embarcado em um veículo automotor, com a capacidade de avaliar a qualidade de um pavimento de qualquer tipo automaticamente em tempo real, conforme a variação nas medições de sensores durante a condução deste veículo. Os objetivos a seguir expõem o escopo previsto até o final deste projeto.

1.1.1 Objetivo Geral

Com este trabalho propõe-se o desenvolvimento de um sistema embarcado que seja capaz de avaliar a qualidade do pavimento em vias públicas através de algoritmos de inteligência artificial, utilizando sensores como acelerômetros, giroscópios e GPS (*Global Positioning System*). Além disso, os resultados obtidos devem ser transmitidos e armazenados em um servidor disponível na *Internet*.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho foram definidos como os requisitos necessários para o funcionamento de cada etapa do projeto, que integrados consistem no objetivo geral e final. Assim, faz-se necessário o desenvolvimento e funcionamento dos seguintes itens:

- validar o funcionamento e aquisição de dados dos sensores utilizados em um sistema embarcado;
- definir um plano de coleta de dados e avaliação de qualidade de pavimento para utilização no treinamento do algoritmo de avaliação;
- verificar a eficiência da aplicação de um algoritmo de inteligência artificial para a avaliação da qualidade de vias pavimentadas;
- transmitir os resultados das avaliações para um servidor pela *Internet*.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os estudos realizados durante o planejamento e execução deste projeto. Informações importantes sobre os sensores, protocolos de comunicação e algoritmos utilizados são expostos aqui, assim como projetos relacionados ao escopo deste trabalho.

2.1 TRABALHOS RELACIONADOS

Obras anteriores já propuseram soluções parecidas para objetivos também similares. Contudo, existem diversas maneiras de se abordar o problema de classificação da qualidade de vias, seja utilizando um algoritmo diferente, outros tipos de sensores, dados de outra natureza, ou até mesmo abordando escopos diferentes.

Em 2014, um grupo de engenharia civil da Universidade de Calábria propôs uma solução utilizando Android e inteligência artificial para detecção de buracos nos pavimentos (VITTORIO et al., 2014). Seus objetivos foram levemente diferentes deste trabalho, pois seu foco foi somente na detecção de buracos. Para tal, utilizaram-se de acelerômetros para detectar eventos de alto impulso no eixo vertical deste sensor. Contudo, o celular não foi responsável por este processamento, e contou com o envio das leituras dos sensores em tempo real para um servidor a uma taxa de uma leitura por segundo. A análise dos sinais ficou então por conta do servidor, o que possivelmente poderia tornar-se um gargalo em larga escala e um problema para lugares sem conexão. Algoritmos para reorientação dos eixos e filtragem de baixa frequência também foram utilizados. Seus testes consistiram na avaliação do algoritmo em duas estradas, atingindo uma eficiência de 80% de verdadeiros positivos e 20% de falsos positivos.

Outra técnica consiste na análise de imagens para detecção de buracos. Este método foi explorado por Koch e Brilakis em 2011. Sua abordagem utilizou técnicas de processamento de imagem para estimar o formato e tamanho de buracos no asfalto. Seu conjunto de dados compreendeu 120 imagens de asfalto em diversas condições, rendendo uma acurácia final de 86% (KOCH; BRILAKIS, 2011). Por utilizar imagens de câmeras comuns, a qualidade de detecção é diretamente influenciada pela qualidade

da mesma e pelas condições de luminosidade, além de ser uma tarefa mais custosa computacionalmente do que a análise de dados de acelerômetros. Contudo, por não utilizar-se deste sensor, a detecção não varia conforme a suspensão e amortecimento do veículo.

Na Universidade da Letônia, outro grupo também estudou a detecção de buracos em pavimentos com a utilização da plataforma Android. Foram analisados quatro métodos para detecções pontuais de buracos através de acelerômetros: limite máximo de aceleração do eixo vertical, limite diferencial entre medidas consecutivas, desvio padrão do eixo vertical, e cruzamento de medidas por zero. Realizaram 10 voltas em um trajeto de 4,4 km com buracos mapeados previamente. O melhor resultado apresentou uma precisão de 90% com o algoritmo de detecção do limite diferencial (MEDNIS et al., 2011).

Seguindo o objetivo para detecção de buracos ou irregularidades pontuais, um laboratório do MIT em 2008 implantou um sistema embarcado em táxis da região de Boston com acelerômetros. Neste trabalho, utilizaram filtros passa-alta e de velocidade mínima para eliminar potenciais problemas de falso-positivos, que foi reduzido para 0,2% (ERIKSSON et al., 2008). Contudo, os autores obtiveram dificuldades para analisar o valor de falsos-negativos, pois observaram que os motoristas evitam intencionalmente os buracos nas rodovias, desviando o carro. Por este motivo, a obra aqui apresentada utiliza uma abordagem de aprendizado supervisionado para detectar o comportamento do motorista através de medidas do acelerômetro e giroscópio para classificar a qualidade da pista.

2.2 SENSORES

De forma a implementar a avaliação automática de qualidade do pavimento, é necessário coletar medições de algumas grandezas físicas no momento da condução de um veículo, processo este que é realizado por sensores. Estes são dispositivos eletrônicos que permitem a conversão de uma grandeza física ou energia em sinal elétrico (BALBINOT; BRUSAMARELLO, 2011). Neste projeto, é feito o uso de acelerômetros e giroscópios para medir quantidades de aceleração e velocidade angular nos três eixos, e assim estimar métricas de qualidade do pavimento.

2.2.1 Acelerômetro e Giroscópio

Acelerômetros são sensores projetados para medir a aceleração própria de um objeto. A aceleração própria de um objeto, diferentemente do sentido convencional da palavra, é a aceleração relativa a um referencial em queda livre. Desta forma, um objeto em repouso em cima de uma mesa possui uma aceleração própria de $9,8m/s^2$ para cima, enquanto um objeto em queda livre ou orbitando a Terra possui uma aceleração própria de $0m/s^2$ (RINDLER, 2013). Como muitas das medidas realizadas pelos acelerômetros são relativas à aceleração da Terra, a unidade comum para estas medidas é a força G, onde $1g = 9,8m/s^2$.

Os acelerômetros podem ser construídos através da fundamentação de diversos princípios físicos. Entre eles, vale citar os piezoelétricos, capacitivos, de efeito Hall e óticos (WOODFORD, 2017). Contudo, entre tantos tipos disponíveis, o mais comum comercialmente é o capacitivo MEMS (*Microelectromechanical systems*).

O princípio de funcionamento dos acelerômetros capacitivos MEMS baseia-se no deslocamento relativo entre placas metálicas dentro de um invólucro. Sabe-se que a capacitância de duas placas paralelas é dada pela Fórmula 2.1.

$$C = \epsilon \frac{A}{d} \tag{2.1}$$

Onde ϵ é a permissividade do material isolante, A é a área das placas paralelas e d é a distância entre elas.

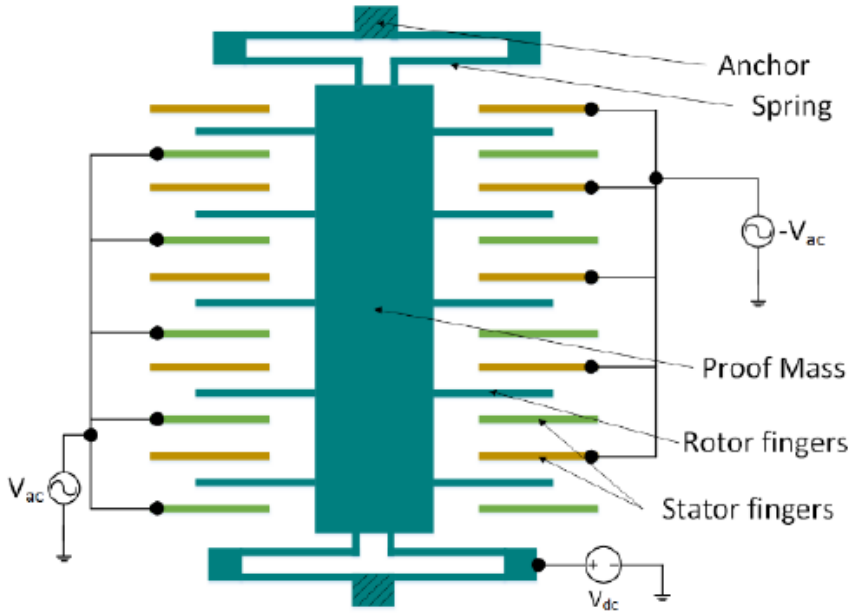
Desta forma, o deslocamento entre estas placas pode ser medido pela diferença no valor da capacitância delas (ANDREJASIC, 2008). A capacitância, por ser uma grandeza elétrica, é facilmente transformada para outros tipos de sinais elétricos. Quando a distância entre as duas placas metálicas aumenta, a capacitância diminui, e vice-versa.

Assim, um acelerômetro capacitivo pode ser montado como na Figura 2.1. Neste diagrama, são colocadas diversas placas metálicas móveis em um bloco acopladas à duas molas e amortecedores. Placas fixas são então intercaladas com as placas móveis, de forma que qualquer aceleração própria no material cause uma diferença na distância entre elas, e consequentemente em suas capacitâncias.

Os giroscópios funcionam com uma estrutura muito similar aos acelerômetros, e por isso foram classificados dentro desta seção. Estes são sensores utilizados para a

medição de orientação e velocidade angular de um objeto. Os giroscópios eletrônicos comerciais mais comuns são do tipo MEMS e funcionam através de estruturas vibratórias sujeitas à força Coriolis. Ao sofrer alguma mudança na velocidade angular, essas estruturas sofrem um deslocamento proporcional. Semelhantemente aos acelerômetros MEMS, o deslocamento é calculado pela diferença de capacitância entre placas carregadas (APOSTOLYUK, 2006).

FIGURA 2.1 – Esquema eletromecânico do acelerômetro capacitivo MEMS



Fonte: (SINHA et al., 2014)

Os componentes mecânicos como as molas e amortecedores podem introduzir erros de leitura em certas faixas de frequência, principalmente na ressonância. Portanto, o uso deste tipo de acelerômetros é aconselhado para frequências menores que a harmônica, na faixa de dezenas até centenas de Hertz (NISTICO, 2013).

2.2.2 *Global Positioning System* (GPS)

Apesar de não ser tecnicamente um sensor, os receptores GPS conseguem estimar medidas importantes como sua própria localização geográfica, velocidade atual e altitude, e por isso foram considerados dentro desta seção do trabalho.

O sistema de GPS funciona através do posicionamento de satélites em órbita da Terra. O projeto inicial da infraestrutura GPS previa um total de 24 satélites, porém em 2016 havia um total de 32 satélites em operação, para maior cobertura e disponibilidade

de sinal (NGA, 2016). Estes satélites possuem um relógio atômico internamente, de forma a medir o tempo da forma mais precisa possível. O receptor também possui um relógio interno, apesar de menos preciso.

Cada um dos satélites em órbita envia constantemente sinais eletromagnéticos modulados à Terra, contendo sua posição na órbita e o horário de transmissão do sinal. O receptor, que também contém um relógio interno, consegue medir sua distância ao satélite através do tempo de viagem do sinal, calculado a partir da Fórmula 2.2.

$$|r(x, y, z)| = (T_R - T_T) \cdot c \tag{2.2}$$

Onde $|r(x, y, z)|$ é o módulo da distância percorrida pelo sinal, T_R é o tempo na recepção do sinal medido pelo próprio receptor, T_T é o tempo da transmissão enviado pelo satélite, e c é a velocidade da luz. Desta forma, a posição do receptor é estimada utilizando a Fórmula 2.3.

$$p_r(x, y, z) = p_s(x, y, z) + |r(x, y, z)| \tag{2.3}$$

Onde $p_r(x, y, z)$ são as coordenadas cartesianas do receptor, e $p_s(x, y, z)$ são as coordenadas do satélite, enviadas pelo mesmo. Contudo, considerando as variáveis x, y e z, este é um sistema indeterminado para os dados de apenas um satélite. Assim, para realizar a geolocalização, o receptor precisa então receber o sinal de pelo menos três satélites simultaneamente. Obtendo estes sinais, o receptor pode determinar a sua posição exata, pela técnica popularmente conhecida como triangulação. No entanto, para corrigir os efeitos de *drift* do relógio do receptor, o sinal de um quarto satélite pode tornar-se necessário. Caso mais satélites estejam disponíveis, estes podem ser utilizados para aumentar ainda mais a precisão da posição calculada.

2.3 PROTOCOLOS E BARRAMENTOS DE COMUNICAÇÃO

O projeto proposto aqui depende extensamente da correta comunicação entre os diferentes módulos integrantes do sistema. Consequentemente, uma fundamentação sobre os barramentos e os protocolos de comunicação utilizados é relevante. Aqui será discorrido sobre os padrões de comunicação I2C (*Inter-Integrated Circuit*), UART (*Universal Asynchronous Receiver-Transmitter*) e a especificação NMEA-0183.

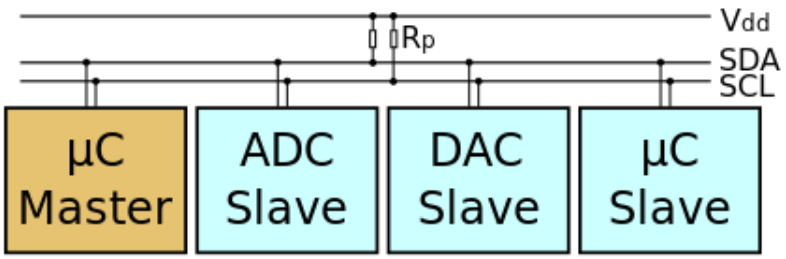
2.3.1 Comunicação I2C

O I2C é um padrão utilizado para comunicação entre periféricos com um número reduzido de fios. Foi inventado pela *Phillips Semiconductor* na década de 1980, e até hoje é amplamente utilizado na indústria, geralmente para comunicação entre módulos de baixa velocidade presentes na mesma placa de circuito impresso (IRAZABAL; BLOZIS, 2003) (KALINSKY; KALINSKY, 2001).

O funcionamento do I2C é respaldado por uma comunicação serial do tipo *master/slave* de apenas dois pinos: SDA e SCL. O SDA é uma sigla para *Serial Data Line*, e como o nome sugere, é por onde todos os dados são transmitidos, de forma bidirecional. Consequentemente, por ser uma linha compartilhada, o I2C é uma comunicação *half-duplex*, onde apenas um dispositivo pode transmitir por vez.

SCL, por sua vez, é uma sigla para *Serial Clock Line*, e é a linha por onde o pulso de relógio gerado pelo dispositivo *master* é transmitido. No projeto original do I2C, a velocidade máxima prevista era de 100kbps. Contudo, a revisão mais recente do protocolo até a data de publicação deste trabalho prevê velocidades de até 3,4Mbps (SEMICONDUCTORS, 2014). As linhas do I2C geralmente funcionam em tensões de 3,3V ou 5V com resistores de *pull-up*, porém outros valores também são permitidos. Por possuir um dispositivo *master* e possivelmente vários *slaves*, um circuito I2C pode ser definido como na Figura 2.2. Percebe-se os resistores de *pull-up* em ambas as linhas SDA e SCL, e o compartilhamento destas linhas entre os múltiplos dispositivos conectados.

FIGURA 2.2 – Exemplo de comunicação I2C



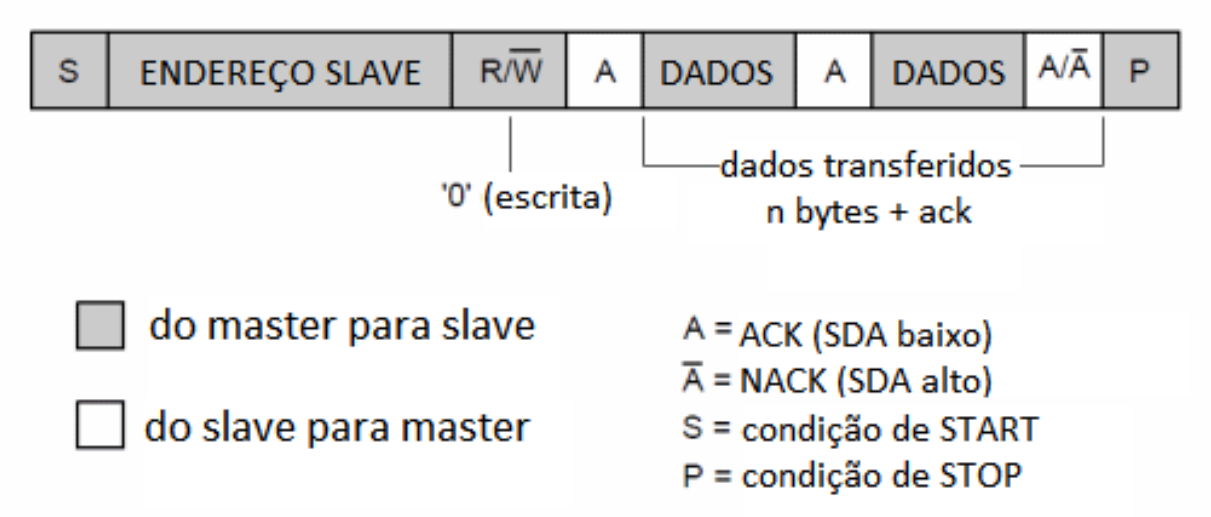
Fonte: (BURNETT, 2018)

De modo a direcionar os dados para um único módulo entre todos os conectados, o I2C define endereços de 7 bits para os dispositivos. Desta forma, para que

a comunicação ocorra com sucesso conforme a topologia da Figura 2.2, todos os módulos devem possuir endereços diferentes.

Estando todos os endereços devidamente definidos, a transmissão de dados para escrita no *slave* ocorre na forma da Figura 2.3. Inicialmente, o *master* envia um bit sinalizando o início da transmissão (*START*), na forma de uma transição alto-baixo no SDA enquanto o SCL estiver em nível alto. A seguir, o endereço do *slave* é transmitido, começando sempre pelo bit mais significativo. Depois, um bit é enviado indicando se o *master* vai ler ou escrever na linha. O bit 0 indica escrita.

FIGURA 2.3 – Diagrama temporal I2C



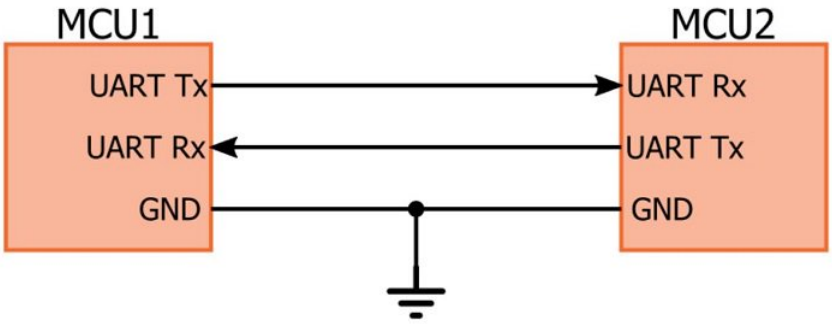
Fonte: (INFO, 2018)

No exemplo da Figura 2.3, a escrita foi sinalizada, e por isso o *master* continua escrevendo no SDA, enquanto o *slave* continua ouvindo. Contudo, antes de realizar a transmissão dos dados, o dispositivo ouvindo deve enviar um ACK (bit 0) para confirmar o recebimento. Confirmado o ACK, os dados começam a ser transmitidos no formato de 8 bits e mais um ACK ao final. Esta transmissão continua indefinidamente até a condição de *STOP* ser sinalizada através de uma transição baixa-alta enquanto o SCL estiver em alto. Neste momento, a transmissão é encerrada e a linha I2C é liberada novamente para outras transmissões. O processo para leitura do *slave* ocorre de forma análoga.

2.3.2 Universal Asynchronous Receiver-Transmitter

A UART (*Universal Asynchronous Receiver-Transmitter*) é um dispositivo de comunicação serial ponta-a-ponta cuja transmissão que, ao contrário da I2C, ocorre de forma assíncrona. Isto significa que não há um sinal de *clock* transmitido junto à linha. Desta forma, toda a sincronização do sinal ocorre por *software*, consequentemente diminuindo a velocidade máxima permitida pelo padrão (ROUSE, 2012). Apesar disso, a velocidade da UART pode ser alterada a qualquer momento, desde que ambos os dispositivos estejam operando na mesma velocidade. Assim, alguns padrões de velocidade são definidos, sendo os mais comuns: 4800bps, 9600bps, 19200bps e 115200bps. Geralmente a comunicação UART depende de três conexões: RX e TX, onde os dados serão respectivamente recebidos e transmitidos, além do terra. Isto possibilita que a comunicação seja *full-duplex*, com ambas as pontas recebendo e transmitindo dados simultaneamente (KEIM, 2016). Assim, uma conexão entre dois dispositivos UART são realizados como na Figura 2.4. É importante notar que o TX de uma ponta é ligado no RX da outra, e vice-versa.

FIGURA 2.4 – Conexão para UART



Fonte: (KEIM, 2016)

Em circuitos UART, a comunicação ocorre da seguinte forma: primeiramente um bit de início (*start*) é enviado forçando o nível lógico da linha para 0. Imediatamente após este bit, os dados começam a ser enviados, geralmente iniciando-se pelo bit menos significativo. O número de bits de dados enviados por transmissão depende da codificação utilizada, porém é usual o valor de 8 bits. Em alguns sistemas, um bit de paridade também é enviado após os dados para verificação da integridade da comunicação. Finalmente, um ou dois bits de parada (*stop*) são enviados em nível lógico alto, assegurando o receptor do final da transmissão dos dados (DURDA, 1996).

2.3.3 Especificação NMEA-0183

A *National Marine Electronics Association* (NMEA) desenvolveu um padrão para dispositivos eletrônicos marinhos. Esse padrão inclui o protocolo de transmissão de dados entre dispositivos marinhos e/ou computadores, e também define as características do sinal elétrico (INFORMATION, 2015). Receptores de GPS utilizam o formato de dados especificado pelo protocolo NMEA-0183, onde o tipo da mensagem é especificada pela primeira palavra, e os dados desta mensagem são separados por vírgulas. Assim, um exemplo de uma mensagem pode ser definido como:

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Todas as mensagens nesse padrão são iniciadas pelo caractere cifrão ($), e cada seção da mensagem é separada por uma vírgula (BETKE, 2000). Quebrando a mensagem em partes, ela é caracterizada de acordo com a Tabela 2.1.

TABELA 2.1 – Campos da mensagem $GPRMC

Sequência	Valor	Descrição
0	$GPRMC	Identificador da mensagem
1	123519	Tempo UTC em segundos
2	A	Status: A=Ativo, V=Inválido
3	4807.038	Latitude no formato DDMM.MMM (D=Graus, M=min.)
4	N	Hemisfério Norte (N=Norte, S=Sul)
5	01131.000	Longitude no formato DDDMM.MMM (D=Graus, M=min.)
6	E	Hemisfério Oriental (W=Ocidental, E=Oriental)
7	022.4	Velocidade de chão em nós por segundo
8	084.4	Ângulo da direção do movimento em graus
9	230394	Data no formato DDMMYY (D=Dia, M=Mês, Y=Ano)
10	003.1	Declinação magnética em graus
11	W	Sentido da declinação magnética (W=Oeste, E=Leste)
12	*6A	*Checksum* da mensagem

Fonte: (BADDELEY, 2001)

A mensagem GPRMC é uma forma básica de mensagem GPS no padrão NMEA. Ela representa os dados mínimos recomendados de GPS/TRANSIT. Existem

diversas outras mensagens previstas na especificação NMEA-0183, que podem conter outros tipos de informação (WORLD, 2015). A implementação ou não de cada uma destas mensagens fica a critério do fabricante do receptor GPS.

2.4 ALGORITMOS DE CLASSIFICAÇÃO

Uma das áreas mais importantes do estudo da inteligência artificial é o desenvolvimento de algoritmos que consigam analisar uma observação e extrair alguma informação relevante dela. Uma dos tipos destes algoritmos é o aprendizado supervisionado. Este consiste no processo de realizar o treinamento (ou aprendizado) do sistema a partir de um conjunto que possui classificação previamente conhecida, e depois aplicar o modelo treinado em casos desconhecidos. Esta estimação do valor de saída pode ser de dois tipos: de valor contínuo, ou entre um número discreto de classes. O primeiro caso é chamado de regressão, e o segundo de classificação.

No caso deste projeto, o objetivo é conseguir avaliar a qualidade de pavimento conforme as leituras de sensores entre um número finito de níveis de qualidade, como bom, regular ou ruim, e portanto serão utilizados classificadores.

2.4.1 Classificador Linear

O classificador linear é um método para classificar entradas entre duas ou mais classes de saída. Seu funcionamento baseia-se na extração de uma ou mais *features* da observação de entrada e processá-las para realizar a classificação. O processamento consiste em achar uma função numérica linear que separa as classes com a maior precisão. Por envolver a procura de uma função matemática, estas *features* devem ser apresentadas no formato de um valor numérico. Assim, o classificador se manifestará através da Fórmula 2.4 (GROSSE, 2017).

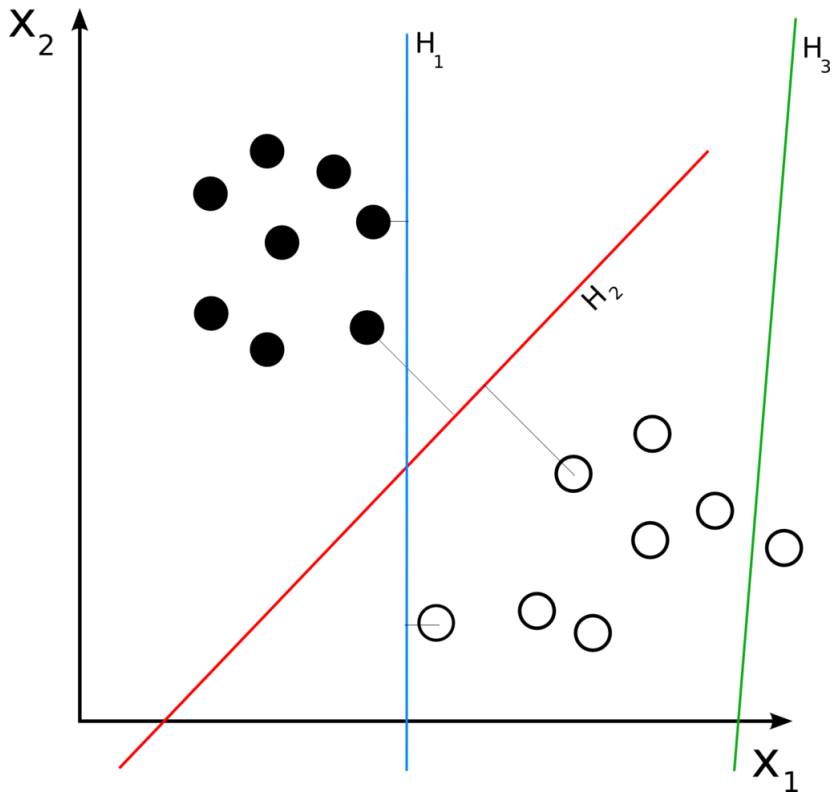
$$f(\vec{x}, \vec{w}, b) = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b = \vec{w_T} \cdot \vec{x_T} + b \tag{2.4}$$

Nesta equação, os valores de $\vec{w}$ são os coeficientes da reta chamados de pesos, que definirão a inclinação no hiperplano. O vetor $\vec{x}$, por sua vez, são as entradas (*features*) deste classificador. O valor de b representa o viés, uma espécie de deslocamento do valor. Finalmente, o classificador apresenta-se como uma função dessas três variáveis: $f(\vec{x}, \vec{w}, b)$. Caso o seu valor para uma entrada $\vec{x}$ seja maior do que zero,

ele pertencerá a uma classe, e caso seja menor do que zero, ele pertencerá à outra. A Figura 2.5 representa muito bem as possibilidades de um classificador linear.

Nela, os dados de treino são representados pelos círculos pretos e brancos, representando categorias diferentes. Os eixos x_1 e x_2 representam as *features* desse classificador, e as retas H_1, H_2 e H_3 constituem possíveis classificadores. Finalmente, as inclinações das retas representam os pesos de cada *feature* ($\vec{w}$). O viés b não é tão claro na imagem, mas representa o valor da reta no eixo x_2 quando o eixo x_1 é zero.

FIGURA 2.5 – Exemplo de classificador linear



Fonte: (WEINBERG, 2018)

Analisando a Figura 2.5, é visível a diferença de qualidade entre os classificadores. H_3 está claramente mal elaborado, e não consegue separar com sucesso os círculos pretos dos brancos. O classificador H_1, por sua vez, consegue classificar com 100% de acerto as entradas dos dados de treino, porém com uma margem tão pequena que possivelmente sua taxa de acerto cairia muito quando colocado em prática. Finalmente, H_2 representa o classificador ideal para estes dados de treino, pois classifica todos os dados sem erro, e maximiza sua margem para erros equalizando as distâncias entre os círculos de classes diferentes mais próximos do limiar.

Assim sendo, a função do algoritmo de um classificador linear é de encontrar os valores do vetor de pesos $\vec{w}$ e do viés b para minimizar os erros de classificação e maximizar a margem de um conjunto de dados de treino. Justamente por depender de dados de treino já classificados, este é um tipo de aprendizado de máquina denominado de supervisionado.

2.4.2 *Support Vector Machines*

O classificador linear, apesar de possuir fácil implementação computacional e de rápida execução, possui um problema que é muito recorrente em casos práticos: a separabilidade linear. Esta é uma característica que define se um conjunto de dados de duas classes consegue ser totalmente separado por um hiperplano traçado entre elas.

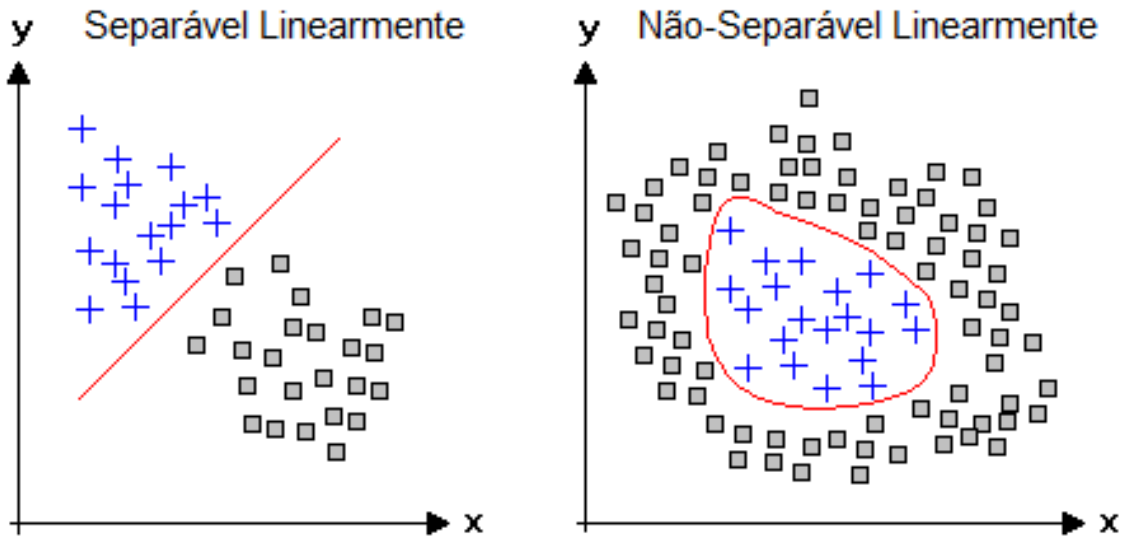
A Figura 2.6 ilustra esta situação. No conjunto de dados da esquerda, as duas classes, cruzes e quadrados, podem ser completamente separadas através de um hiperplano; neste caso uma reta, pois apenas duas dimensões são consideradas. Aqui, o classificador linear funciona com sucesso.

Contudo, no conjunto de dados da direita, não há nenhuma reta que possa ser traçada que consiga separar as duas classes com sucesso. Se um classificador linear fosse aplicado à estes dados, a taxa de erros seria altíssima, qualquer que fossem os parâmetros utilizados neste classificador. A melhor solução neste caso seria encontrar classificadores não lineares, como o proposto na figura, que consigam separar estas classes com sucesso (MANNING; RAGHAVAN; SCHÜTZE, 2009).

Assim, a melhor solução para este tipo de conjunto de dados é a transformação das *features* em um outro domínio que seja linearmente separável, através da aplicação de funções matemáticas. A generalização do classificador linear para outras funções matemáticas é a caracterização das *Support Vector Machines*, mais conhecidas como SVM.

A definição de uma função matemática única e dependente do conjunto de dados que os tornaria linearmente separáveis pode ser computacionalmente muito complexa. Nestes casos, portanto, utiliza-se o conceito do *kernel trick*, uma função de similaridade que emprega o produto interno entre as *features* de dados de teste e os dados de treino. Assim, a classificação de uma entrada utilizando uma função *kernel* é

FIGURA 2.6 – Problema da separabilidade linear



Fonte: (O Autor, 2018)

dada pela Fórmula 2.5.

$$\hat{y} = sgn \sum_{i=1}^{N} w_i \cdot y_i \cdot K(x_i, x') \tag{2.5}$$

Onde x_i é uma entrada de treino (já classificada), y_i é a classificação desta entrada (assumindo valores -1 ou +1), w_i é o peso da *feature*, N é o número de dados de treino, x' são as *features* de uma entrada não classificada, e por fim $K(x_i, x')$ é a função *kernel*.

Os tipos de *kernel* mais utilizados nos estudos de inteligência artificial são os lineares, polinomiais, RBF (*Radial Based Function*) e sigmoide. O *kernel* polinomial, por exemplo, apresenta-se na forma da Fórmula 2.6:

$$K(x, x') = (x^T x' + c)^d \tag{2.6}$$

Nela, percebe-se o produto interno através da transposta do vetor de entrada classificada e do vetor de entrada a ser classificado. O termo d representa o grau do polinômio utilizado, enquanto c é uma constante indicativa do viés para os termos de baixa ordem da equação. Neste trabalho, apenas o polinômio de grau 2 será estudado.

Outro *kernel* muito comum é o RBF, cuja Fórmula 2.7 o descreve:

$$K(x, x') = exp\Big(- \frac{||x - x'||^2}{2\sigma}\Big) \tag{2.7}$$

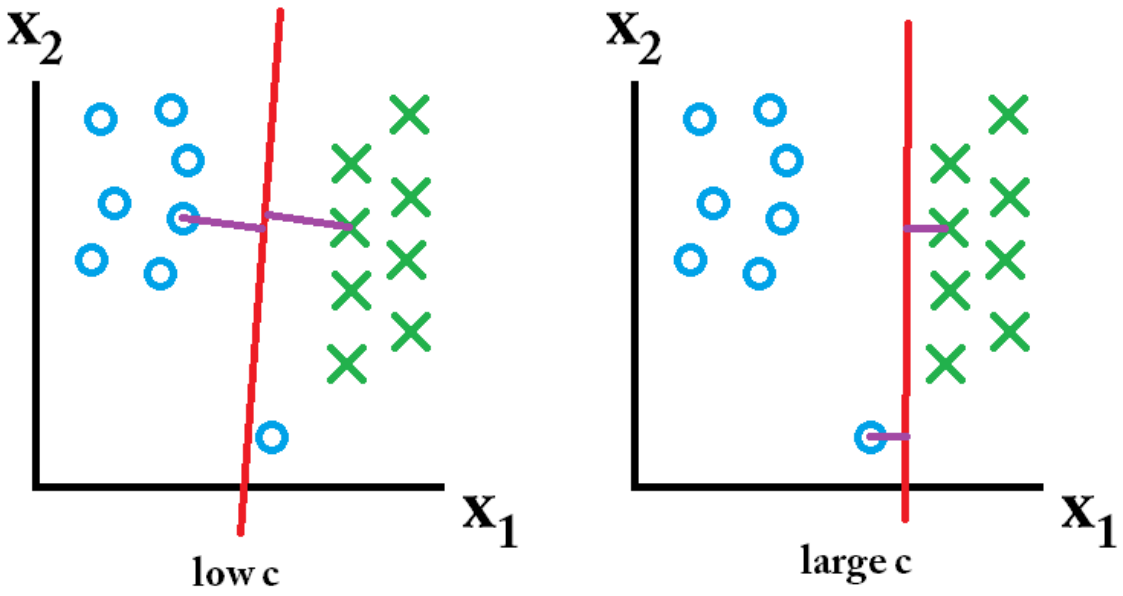
O termo $||x - x'||$ computa a distância euclidiana entre a entrada classificada e a entrada a ser classificada. Esta distância é então elevada ao quadrado e dividida por uma constante 2σ. Esta constante retrata o valor de escala desta distância, e muitas vezes é substituída por $\gamma = 1/2\sigma$. O *kernel* então é extraído pela exponencial destes cálculos.

2.4.3 Hiper-parâmetros para treinamento da SVM

Para realizar o treinamento do modelo para algoritmos de SVM, o analista deve selecionar alguns parâmetros internos, denominados hiper-parâmetros, que influenciarão diretamente na qualidade e precisão do modelo encontrado.

Comum a todos os *kernels* é o parâmetro C. Ele define um *trade-off* entre o esforço para a correta classificação de todos os dados de treino, e um limite de decisão com maior margem (SCIKIT-LEARN, 2018). Para melhor entendimento deste *trade-off*, considere o exemplo da Figura 2.7.

FIGURA 2.7 – Possibilidades para o parâmetro C



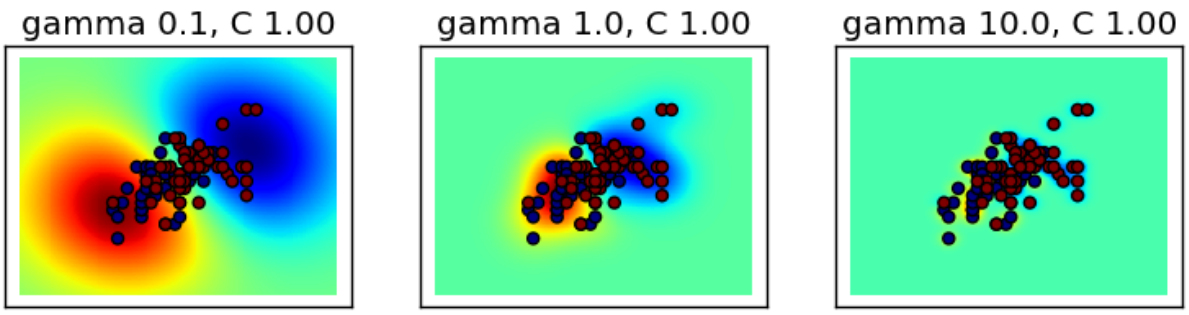
Fonte: (CASPERSEN, 2018)

No caso da gráfico da esquerda, um valor baixo de C foi escolhido. Assim, o algoritmo de treinamento dá preferência a uma margem de decisão mais elevada, e considera o círculo classificado erroneamente como um *outlier*. Por outro lado, caso um valor elevado de C seja escolhido, como no caso da direita, o algoritmo abre mão de uma margem de decisão maior para classificar corretamente todos os dados de treino.

Contudo, esta correta classificação de todos os dados de treino não significa que o mesmo será verdade com os dados de teste. Para avaliar qual valor de C possuirá melhor desempenho na prática, deve-se realizar uma varredura entre diversos valores deste parâmetro e avaliar o desempenho do modelo através de algum tipo de indicador de performance, sempre verificando contra um outro *dataset* separado para teste.

Outro parâmetro utilizado nos *kernels* RBF e polinomial é o γ. Como pode ser observado na Fórmula 2.7, um γ com valor baixo pode ser traduzido como um alto valor de variância da gaussiana projetada pelo *kernel* para cada vetor de suporte. Isto significa que um modelo com um γ muito baixo possui alta variância, e o raio de influência de cada vetor de suporte é maior. Desta forma, caso este raio seja muito grande, ele não conseguirá captar as peculiaridades de cada modelo e não conseguirá classificar corretamente. Por outro lado, caso o γ seja muito alto, o raio de influência de cada vetor de suporte é muito baixo, potencialmente levando à um *overfitting* do modelo. A Figura 2.8 ilustra muito bem esta situação: quando γ assume valores muito elevados, pode-se observar que não generalização no modelo, ocorrendo o *overfitting*. Por outro lado, caso ele seja muito baixo, não consegue absorver os detalhes de cada dado e também não classifica corretamente.

FIGURA 2.8 – Possibilidades para o parâmetro γ



Adaptado de: (SCIKIT-LEARN, 2018)

Para a escolha do valor mais adequado de γ deve-se proceder da mesma

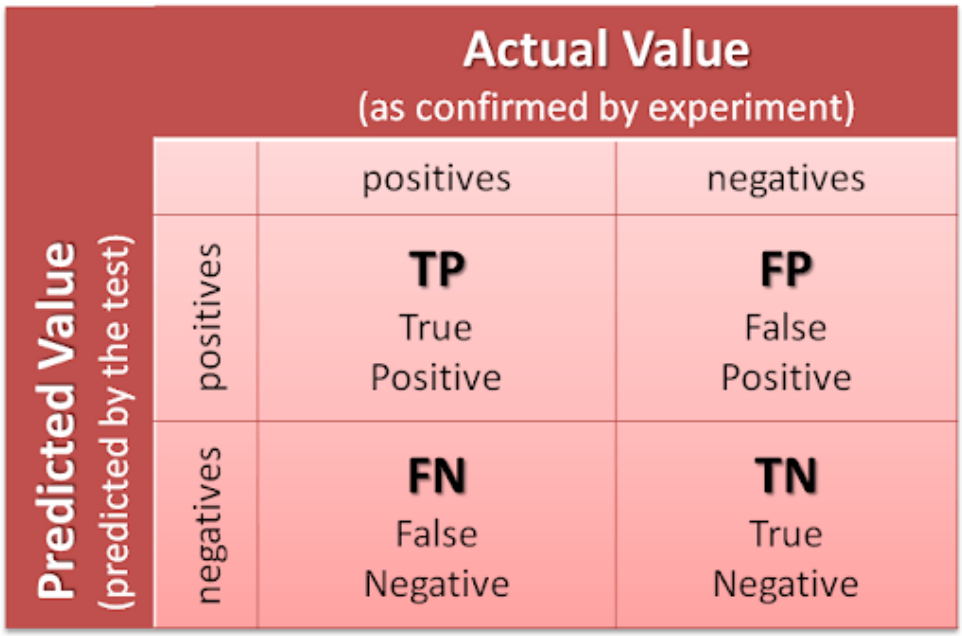
maneira que com o parâmetro C, ou seja, analisando individualmente o desempenho de cada modelo para diversos valores diferentes.

Apesar de não representado na Equação 2.6, um termo γ também pode ser colocado multiplicando sobre o produto interno calculado, e desta forma, este parâmetro também é relevante para o *kernel* polinomial, apesar de operar de forma levemente diferente. Este *kernel* também possui o parâmetro do termo independente da equação polinomial, porém este termo será considerado como zero neste trabalho, para simplificação e por não apresentar grande relevância nos resultados.

2.4.4 Avaliação de desempenho

A fim de analisar o desempenho de cada classificador, pode se fazer uso da matriz de confusão e seus indicadores. As colunas desta matriz representam o resultado esperado (valores atuais), enquanto as linhas representam o que foi previsto pelo algoritmo. Assim, as dimensões desta matriz possuem o mesmo valor do número de classes presentes no modelo. A Figura 2.9 ilustra um exemplo dessa matriz de confusão para 2 classes.

FIGURA 2.9 – Exemplo de matriz de confusão



Fonte: (IDRIS, 2018)

Através desta matriz, é possível entender as classificações corretas e errôneas

do modelo. As relações na matriz de confusão podem ser representadas por quatro possibilidades diferentes:

- verdadeiro positivo (VP): este é um caso em que a previsão foi verdadeira e o valor atual também foi verdadeiro;
- verdadeiro negativo (VN): a previsão foi negativa e o valor atual também;
- falso positivo (FP): a previsão foi positiva mas o valor atual era negativo. Este caso é classificado como erro do Tipo 1;
- falso negativo (FN): a previsão foi negativa mas o valor atual era positivo. Este caso é classificado como erro do Tipo 2.

Encontrados os valores desta matriz, é possível extrair alguns importantes indicadores de qualidade e desempenho deste modelo. O primeiro indicador é a precisão (π), que pode ser calculado como:

$$\pi = \frac{VP}{VP + FP} \tag{2.8}$$

Desta forma, o indicador de precisão mostra a proporção de dados avaliados como positivos que eram realmente positivos. Outra informação importante que pode ser extraída desta matriz é o indicador denominado de recall (ρ). Este número avalia a proporção das previsões positivas entre os dados reais positivos. Matematicamente, pode ser obtido por:

$$\rho = \frac{VP}{VP + FN} \tag{2.9}$$

Assim, caso o modelo classifique todos os casos como positivos, o recall dele será 100%. Como ambas as medidas de precisão e recall são importantes, porém muitas vezes insuficientes por si só, elas geralmente são combinadas em uma média harmônica para formar um novo indicador, o F1 Score (F). Este, então, é calculado através de:

$$F = \frac{2\pi\rho}{\pi + \rho} \tag{2.10}$$

onde π é a precisão e ρ é o recall.

O F1 Score funciona muito bem como um indicador geral de desempenho do modelo encontrado. Desta forma, ele foi utilizado como indicador a ser maximizado na varredura dos parâmetros internos de cada classificador (C e γ). Assim, o modelo ideal para cada classificador foi aquele cujos parâmetros internos maximizou seu F1 Score.

Contudo, esta definição do F1 Score assumiu uma matriz de confusão de ordem 2 por 2, o que representa um caso restrito de um classificador binário. Entretanto, caso seja necessário avaliar um modelo que possua mais de duas classes, o F1 Score pode ser derivado de duas formas diferentes.

A primeira maneira é denominada de micro-média. Nela, os valores de precisão e recall são calculados globalmente sobre todas as classes de decisão (ÖZGÜR; ÖZGÜR; GÜNGÖR, 2005). O valor do F1 Score é determinado posteriormente apenas utilizando estes valores encontrados de precisão e recall globais, que são definidos pela Equação 2.11:

$$\pi = \frac{\sum_{i=1}^{N} VP_i}{\sum_{i=1}^{N} VP_i + FP_i} \qquad \rho = \frac{\sum_{i=1}^{N} VP_i}{\sum_{i=1}^{N} VP_i + FN_i} \tag{2.11}$$

onde N é o número de classes. Como a precisão e o recall na micro-média são calculados somando todas as classes juntas, este método não é recomendado caso haja um desbalanço muito grande entre as frequências das classes, uma vez que isso colocaria um peso maior nas classes de maior frequência.

O outro método é a macro-média. Neste caso, cada classe calcula seus próprios índices de precisão, recall e F1 Score individualmente (ÖZGÜR; ÖZGÜR; GÜNGÖR, 2005). Posteriormente, o F1 Score do modelo é a média de todos os F1 Score de cada classe, como na Equação 2.12:

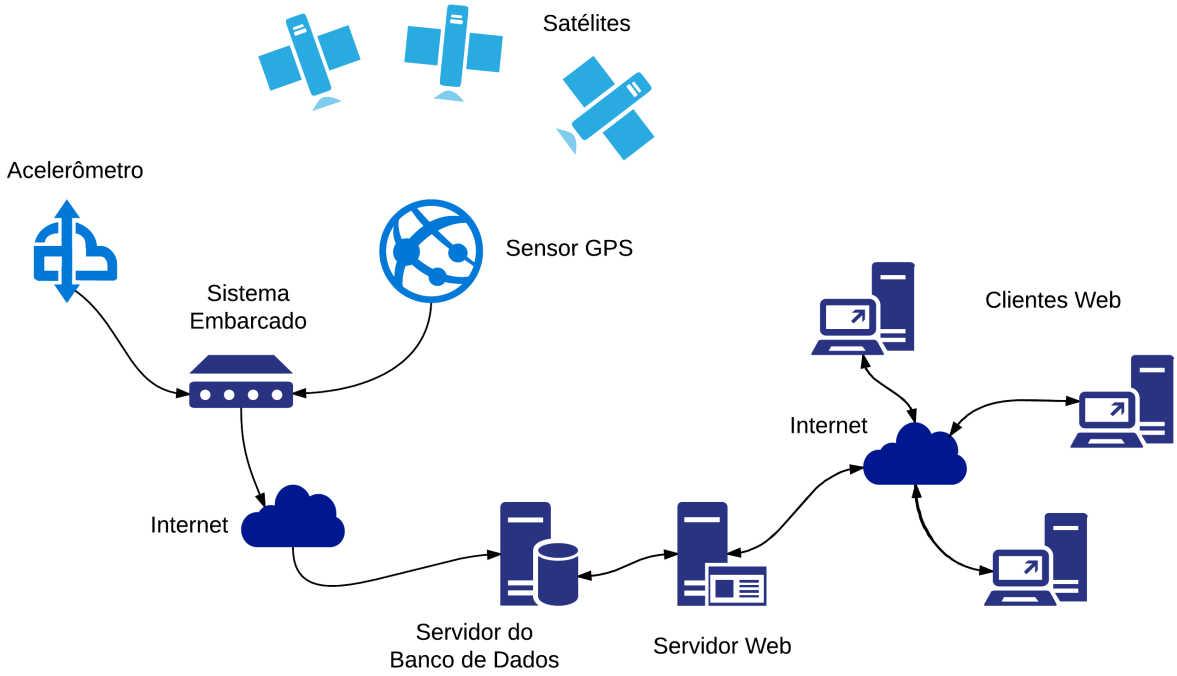
$$F(macro) = \frac{\sum_{i=1}^{N} F_i}{N} \tag{2.12}$$

Como a macro-média realiza a média apenas após o cálculo da precisão e do recall, ela atribui pesos iguais para cada classe, não sofrendo com o problema do desbalanceamento observado na micro-média.

3 SISTEMA PARA DETECÇÃO DE QUALIDADE DE VIAS

A fim de alcançar todas as funcionalidades propostas neste trabalho, três frentes precisam ser desenvolvidas: o interfaceamento e leitura dos sensores; a interpretação dos dados e consequente elaboração do modelo do classificador; e a comunicação e envio do resultado para um servidor Web. O diagrama de todos os componentes do sistema pode ser verificado na Figura 3.1. Nesta Figura, também fica mais clara a função de cada elemento para o projeto, assim como a relação entre eles.

FIGURA 3.1 – Diagrama do sistema de mapeamento



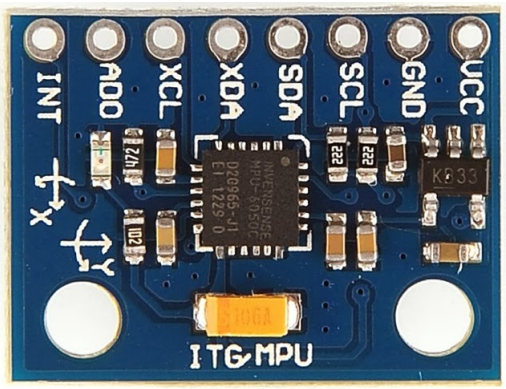
Fonte: (O Autor, 2018)

3.1 MÓDULO ACELERÔMETRO E GIROSCÓPIO

Para o desenvolvimento deste projeto foi escolhido o modelo MPU-6050 como módulo acelerômetro e giroscópio. Este é um circuito integrado desenvolvido pela *InvenSense*, que combina um acelerômetro MEMS capacitivo de 3 eixos e um giroscópio de 3 eixos em um mesmo CI (Circuito Integrado) de baixo consumo, e comunica-se através do protocolo I2C (INVENSENSE, 2017). Os acelerômetros possuem uma escala configurável de $\pm 2g$, $\pm 4g$, $\pm 8g$ ou $\pm 16g$. Similarmente, os giroscópios também

podem ser configurados com uma escala de $\pm 250°/s$, $\pm 500°/s$, $\pm 1000°/s$ ou $\pm 2000°/s$. A Figura 3.2 mostra a placa utilizada com o MPU-6050. Como pode ser observado, a placa possui 8 pinos, sendo eles: INT - Interrupção; AD0 - Endereço; XDA e XCL - Comunicação I2C auxiliar; SCL e SDA - Comunicação I2C principal; VCC e GND - Alimentação. Destes, serão utilizados apenas os pinos de alimentação e comunicação I2C principal.

FIGURA 3.2 – Placa do módulo MPU-6050



Fonte: (14CORE, 2018)

Os registradores internos de medição do MPU-6050, são divididos em 2 de 8 bits, pois a medida é realizada em 16 bits. O registrador com o menor endereço possui o byte de maior significância, e o registrador seguinte possui o byte de menor significância. Os números são armazenados no formato de complemento de dois. Desta forma, para obter o valor correto da leitura no *software*, deve-se concatenar os valores de ambos os registradores, e converter o valor do complemento de dois com o uso da Fórmula 3.1.

$$x_S = -(2^N - x_C) \tag{3.1}$$

Onde x_S é o valor sinalizado após a conversão, N é o número de bits do registrador (16 neste caso) e x_C é o valor do registrador em complemento de dois. Esta fórmula é utilizada apenas se o bit mais significativo do número estiver em 1, o que indica que o número é negativo. Caso contrário, pode-se ignorar a fórmula e utilizar a conversão convencional de número binário.

Para garantir que as leituras dos acelerômetros e giroscópios estejam corretas, o módulo foi submetido aos seguintes testes: O circuito foi contido em um ambiente controlado, onde foram medidos os seus valores de aceleração em queda livre e nos

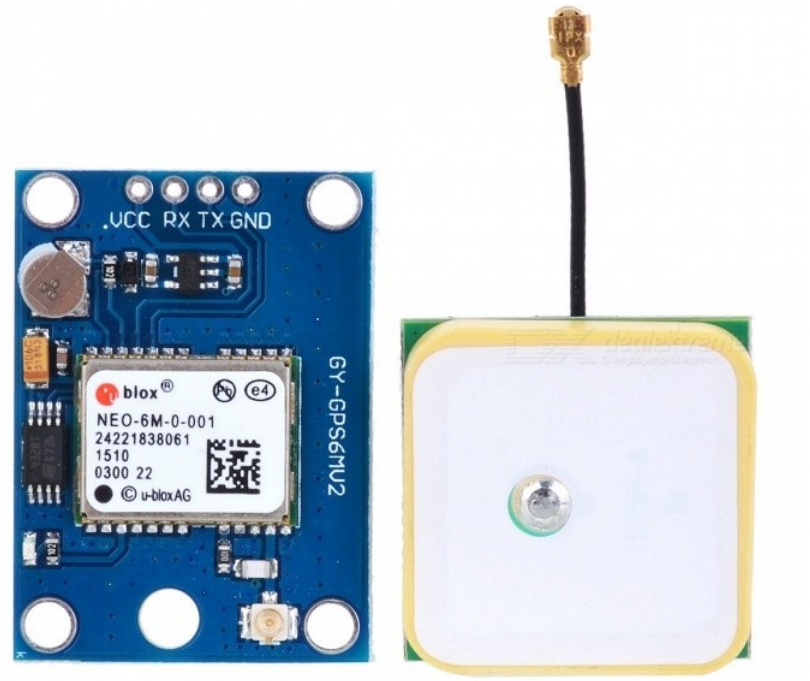
momentos seguintes após o choque com o chão. Para testar a velocidade angular, o circuito foi manualmente rotacionado em volta de seus três eixos, a fim de verificar a correta identificação e medição destes.

3.2 MÓDULO GPS

O módulo de GPS utilizado no sistema embarcado foi o Ublox GY-NEO6MV2. Este módulo comunica-se através de uma interface UART, possui uma memória não-volátil interna para armazenamento de configurações e inclui uma antena de cerâmica (UBLOX, 2011). A Figura 3.3 apresenta o módulo em questão, onde podem ser observados o circuito e a antena utilizados. Assim como na placa do MPU-6050, os pinos disponíveis são apresentados na imagem. São eles: VCC e GND para alimentação; RX e TX para comunicação serial com o sistema embarcado.

Para garantir o funcionamento adequado do módulo GPS, medidas de latitude, longitude e velocidade foram realizadas através de uma referência conhecida, e analisando sua precisão posteriormente.

FIGURA 3.3 – Placa do módulo GPS GY-NEO6MV2



Fonte: (DEALEXTREME, 2018)

3.3 MÓDULO GSM

Após a definição do modelo de inteligência artificial, será implementada a classificação em tempo real dos pavimentos. Para que esta classificação em tempo real tenha algum propósito, seus resultados devem ser mandados para um servidor central localizado na *Internet* que irá armazenar estes dados em definitivo. Como o sistema será embarcado em automóveis, a comunicação foi implementada através de um módulo GSM, que enviará os dados pela *Internet* através do protocolo HTTP (*Hypertext Transfer Protocol*).

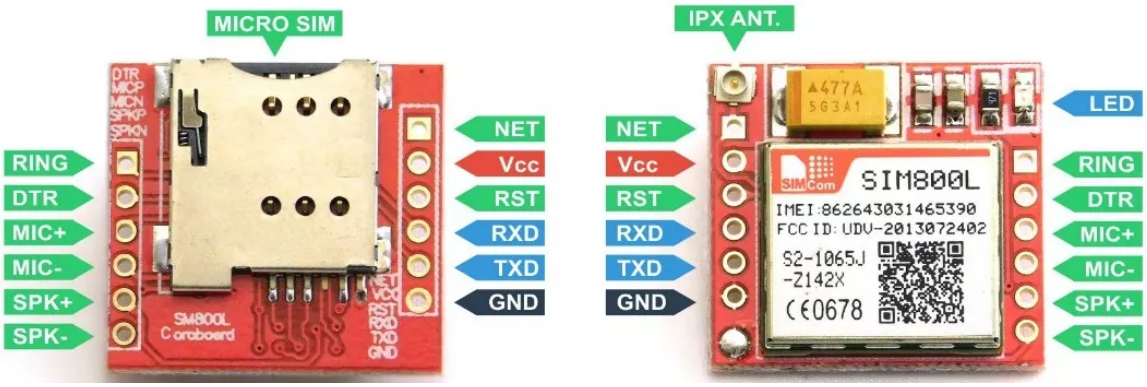
Para tal, foi utilizado o módulo GSM SIM800L, fabricado pela SIMCom. Este é um módulo GPRS/GSM quadriband que funciona nas frequências 850MHz, 900MHz, 1800MHz e 1900MHz. Este módulo possui uma tensão de entrada de 3,4 a 4,4V, e pode consumir até 2A em picos de corrente (SIMCOM, 2013).

Para comunicação com o modem, e consequentemente o envio de mensagens, realização de ligações e conexão com a *Internet*, comandos podem ser enviados para o SIM800L utilizando uma interface UART, conforme descrito na Seção 2.3.2. Os comandos aceitos pelo módulo são definidos por um conjunto de comandos AT estendido, conforme especificado pela própria SIMCom. Estas instruções são em sua maior parte padrão na indústria, porém a SIMCom adicionou algumas funcionalidades próprias em comandos customizados.

Diversas distribuições de módulos com o SIM800L estão disponíveis no mercado. A utilizada neste trabalho está apresentada na Figura 3.4. Nesta figura, é possível notar os pinos disponibilizados pela placa, como conexões para microfone (MIC+/MIC-), alto-falantes (SPK+/SPK-), serial (RXD/TXD) e antena (IPX ANT.). Desses, apenas a conexão com a serial será conectada, além da alimentação e a antena. Ademais, um cartão SIM (*Subscriber Identity Module*) válido por uma operadora nacional deve ser adquirido e utilizado para correta operação de todas as funcionalidades do módulo.

Para garantir o correto funcionamento deste módulo, alguns comandos AT serão testados, como a validação da comunicação UART, o envio de mensagens de texto, e comunicação pela *Internet*. Para efeitos de documentação na seção de resultados deste trabalho, serão mostrados apenas os testes relacionados à comunicação TCP/IP, pois é a única relevante ao escopo do sistema proposto.

FIGURA 3.4 – Módulo SIM800L Utilizado



Fonte: (O Autor, 2018)

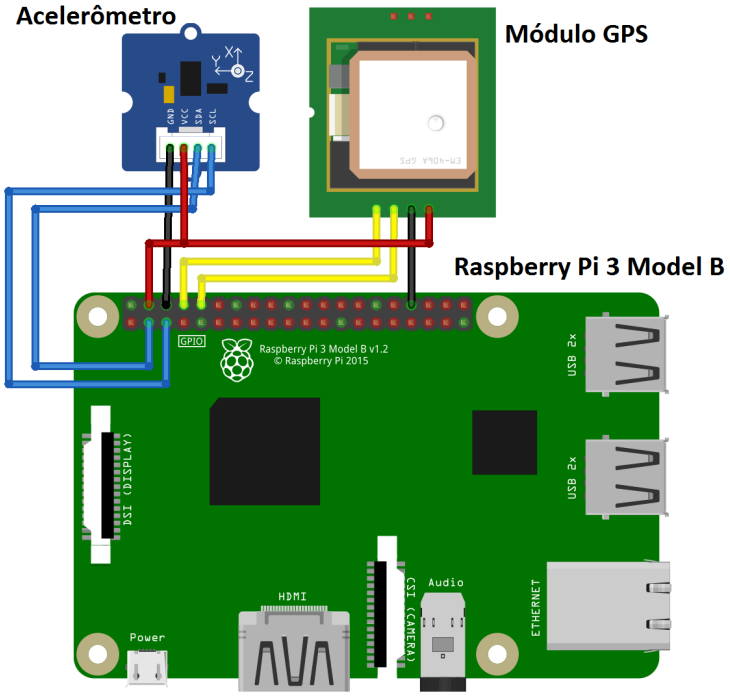
3.4 RASPBERRY PI

Para a escolha do sistema embarcado, foram levados em conta a facilidade de prototipagem, custo, agilidade de desenvolvimento do *software* embarcado, suporte aos protocolos de comunicação dos sensores e compartibilidade de biblitocas de inteligência artificial. Avaliando todos estes quesitos, o Raspberry Pi mostrou-se a plataforma mais adequada para a tarefa. O Raspberry Pi 3 Model B+ é um microcomputador baseado na CPU (*Central Processing Unit*) Broadcom BCM2837 Quad-Core 1,2GHz de 64 bits. Possui 1GB de RAM (*Random Access Memory*), 40 pinos de Entrada/Saída, 4 portas USB (*Universal Serial Bus*), porta HDMI (*High-Definition Multimedia Interface*), Ethernet e Wi-Fi. Além disso, ainda dispõe de pinos configuráveis para fornecer suporte à comunicação I2C e UART (FOUNDATION, 2017).

Para fins deste trabalho, os pinos mais importantes são os da comunicação I2C e Serial, que podem ser encontrados logo no início do *header*. Agrupando todos os componentes de *hardware*, é possível montar o esquemático do circuito, como ilustrado na Figura 3.5.

Uma das dificuldades encontradas com a utilização do Raspberry Pi foi a disponibilidade de apenas uma porta serial no sistema. Como ambos os módulos GSM e GPS comunicam-se através da UART, foi necessário desenvolver um método de multiplexação para que esta única porta serial seja compartilhada por ambos os dispositivos. Assim, um CI foi adicionado para fazer essa multiplexação, e a sincronização desta comutação ficou por conta do *software*.

FIGURA 3.5 – Esquemático do Circuito



Fonte: (O Autor, 2018)

3.5 MÉTODO PARA COLETA DE DADOS

Conforme explicado na Seção 2.4.1, os algoritmos de aprendizado supervisionado necessitam de dados com classificações manuais para treinamento do modelo. Assim, para esta classificação, duas metodologias de separação de classes foram utilizadas. A primeira delas divide o pavimento em três níveis de qualidade, seguindo o critério da Tabela 3.1. A segunda metodologia de separação une as classes ’Regular’ e ’Ruim’ em uma só, e portanto separa o pavimento em apenas dois níveis de qualidade.

Para a avaliação manual, foram colocados três botões em uma *protoboard* no console de um carro, juntamente ao protótipo do sistema. Cada um desses botões representa o nível de qualidade atual do pavimento. Conforme o veículo foi sendo conduzido por estradas em condições diversas, o condutor ia sinalizando esta condição manualmente. Um *software* em *background* no Raspberry Pi foi responsável por registrar os dados dos sensores a cada 10ms e armazená-los juntamente com a condição do pavimento em um arquivo no formato CSV (*Comma Separated Values*), de forma que possam ser processados posteriormente.

Antes da introdução dos dados no treinamento da SVM, um algoritmo de pré-

TABELA 3.1 – Níveis de qualidade do pavimento

Valor	Nome	Significado
3	Boa	Nenhuma ou mínima irregularidade no pavimento que não afetam a experiência da condução.
2	Regular	Poucas irregularidades e pequenos buracos no pavimento que podem ligeiramente afetar a qualidade da condução.
1	Ruim	Diversos buracos e problemas no pavimento que afetam gravemente a qualidade da condução, podendo até obrigar a diminuição de velocidade do automóvel.

Fonte: (O Autor, 2018)

processamento foi executado para descartar medidas não desejadas, ou que pudessem causar alguma inconsistência no classificador. Além disso, ele também foi utilizado para agrupar as medições de um determinado valor contínuo de tempo. Assim, os seguinte método de pós-processamento foi empregado:

- eliminar medidas com velocidade abaixo de 15 km/h;
- eliminar medidas que possuam um sinal de GPS inválido;
- agrupar os últimos 10 segundos de medições contínuas, calculando a média e variância dos valores de aceleração, velocidade, e velocidade angular;
- normalizar os dados para que o conjunto de dados de cada *feature* possua média 0 e variância 1.

3.6 MÉTODO PARA AVALIAÇÃO DA SVM

Após o pré-processamento dos dados, finalmente é possível aplicar os algoritmos de SVM para adquirir o modelo final e avaliar o seu desempenho. Para tanto, os dados pré-processados foram separados pela metade em dois conjuntos: os dados de treino e os dados de teste. Os primeiros foram usados para adquirir o modelo, enquanto o segundo foi consumido para a avaliação de desempenho do mesmo.

Foram analisados três tipos diferentes de *kernel*: o linear, o polinomial com grau 2, e o RBF. Também foi realizada uma varredura dos hiper-parâmetros envolvidos, de forma a obter o melhor desempenho. Estas análises foram realizadas duas vezes,

uma com as três classes de qualidade (bom, regular e ruim), e outra para duas classes (bom e ruim).

Por fim, o *kernel* com melhor desempenho também foi analisado com um outro conjunto de *features*, um em que não haja necessidade de orientação específica dos eixos dos acelerômetros e giroscópios, através do cálculo do módulo dos três eixos.

A fim de definir os valores dos parâmetros C e γ, foi escolhido o índice F1 Score aplicado através da macro-média para maximização. A macro-média foi utilizada pois espera-se que haja um número muito maior de vias com qualidade boa do que ruim.

Após a obtenção do hiper-parâmetros, a matriz de confusão será construída para cada modelo e *kernel* diferente, de modo a poder observar as nuances de cada classificador. Neste trabalho, dadas as metodologias 1 e 2, verifica-se duas possíveis matrizes de confusão: uma com dimensões 2x2 e outra 3x3.

3.7 CLASSIFICAÇÃO EM TEMPO REAL

Uma vez capturado o modelo mais performático da SVM, é necessário desenvolver o código-fonte do *software* que será embarcado no Raspberry Pi para avaliação em tempo real da qualidade do pavimento.

No diagrama da Figura 3.6, verifica-se o fluxo lógico deste código. Inicialmente, são inicializadas as comunicações I2C e Serial. Aguarda-se então a obtenção do sinal GPS. Estabelecidas todas estas conexões, o *software* começa a armazenar todas as leituras do acelerômetro e GPS, desde que acima da velocidade limite de 15km/h. Esta velocidade mínima é um limite para não realizar leituras sem relevância com o veículo quase parado. Toda vez que uma velocidade abaixo de 15 km/h for detectada, todas as leituras no *buffer* são descartadas.

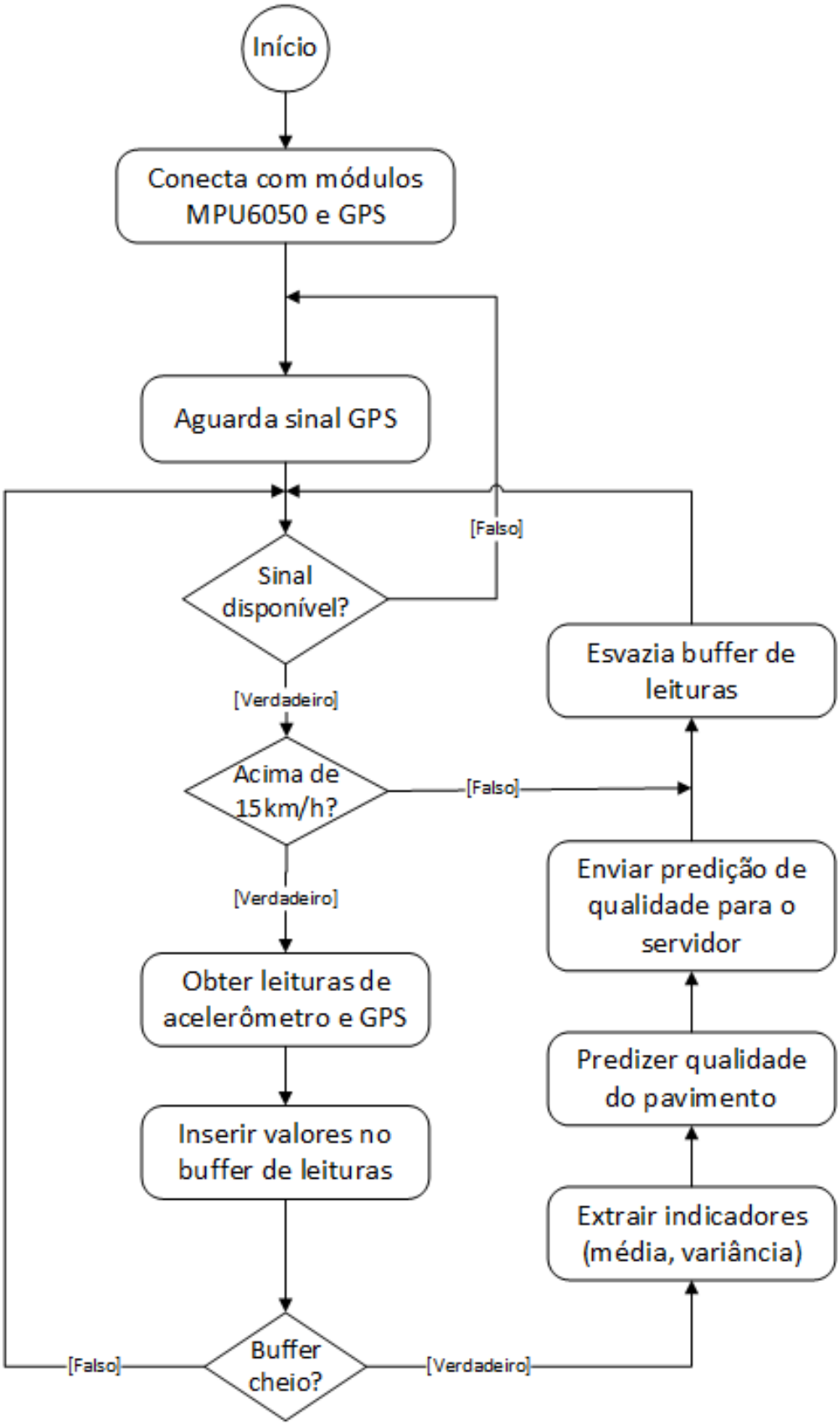
Estando então acima da velocidade mínima, o sistema começa a acumular todas as leituras do acelerômetro, giroscópio e GPS em um *buffer* durante um tempo de 10 segundos, completando este *buffer*. Neste momento, todos os indicadores de entrada da SVM são extraídos: a média e a variância dos três eixos do acelerômetro, giroscópio e da velocidade do automóvel. Este algoritmo de classificação, já treinado, é acionado para classificar a qualidade do pavimento.

Feita a classificação, o *software* embarcado abre uma conexão TCP (*Trans-*

mission Control Protocol) com o servidor através do módulo GSM para enviar esta informação, juntamente com as coordenadas GPS da detecção. Depois disso, o *buffer* é novamente esvaziado e o processo reinicia novamente. Enquanto houver sinal de GPS e velocidade acima do limite mínimo, o sistema continuará a adquirir dados e enviar ao servidor indefinidamente.

O código do fluxograma da Figura 3.6 será implementado no Raspberry Pi utilizando a linguagem Python, com algumas bibliotecas nativas e auxiliares para facilitar o desenvolvimento, como a *threading*, *SMBus*, *Serial* e *sys*.

FIGURA 3.6 – Fluxograma do *Software* Embarcado



Fonte: (Autor, 2018)

4 RESULTADOS

De forma a modularizar o processo de desenvolvimento e teste, os dados apresentados no início deste capítulo de resultados foram obtidos com os sensores e módulos operando independentemente. Posteriormente à comprovação do funcionamento adequado de todas as partes necessárias, serão apresentados os resultados referentes à SVM e a integração destes módulos.

4.1 GPS

O módulo GPS empregado neste trabalho utiliza-se da comunicação serial através da interface *UART* para transmitir seus dados. Assim, o primeiro teste realizado foi escutar a porta serial do Raspberry Pi a fim de identificar as mensagens NMEA enviadas pelo módulo. Estes dados podem ser vistos na Figura 4.1. A cada segundo, um bloco contendo todas as mensagens como mostrada nesta Figura é repetido.

FIGURA 4.1 – Dados não-tratados recebidos do GPS

```
$GPRMC,015255.00,A,2523.43864,S,04913.21103,W,0.553,,101017,,,A*79
$GPVTG,,T,,M,0.553,N,1.025,K,A*26
$GPGGA,015255.00,2523.43864,S,04913.21103,W,1,08,1.29,914.5,M,-2.5,M,,*41
$GPGSA,A,3,16,18,10,26,21,15,29,20,,,,,2.58,1.29,2.23*01
$GPGSV,3,1,12,10,59,315,34,13,03,135,,15,24,112,23,16,34,237,18*7D
$GPGSV,3,2,12,18,86,198,15,20,20,138,11,21,50,157,27,25,02,015,*73
$GPGSV,3,3,12,26,47,276,23,27,17,220,17,29,30,064,22,32,07,356,*76
$GPGLL,2523.43864,S,04913.21103,W,015255.00,A,A*65
```

Fonte: (Autor, 2018)

Para conseguir interpretar os dados de maneira correta, foi criada uma classe em Python para a leitura e formatação desses dados, abstraindo diversas peculiaridades do protocolo. Esta classe foi chamada simplesmente de GPS e está disponível no Apêndice A.

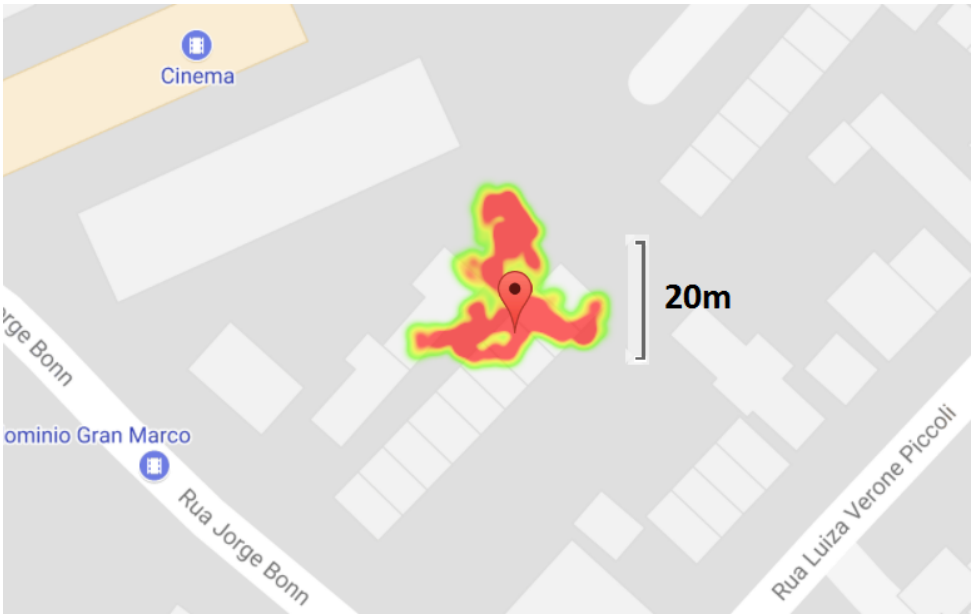
A classe GPS herda da classe *Thread*. Isto foi feito assim pois a leitura dos dados é realizada pela porta serial, que precisa ser constantemente lida para obtenção dos dados atualizados. Assim, utilizar uma nova *thread* provê uma interface onde os dados são atualizados automaticamente e frequentemente.

Um problema encontrado na integração dos diferentes módulos foi que ambos o GPS e o GSM utilizam-se da comunicação serial, porém o Raspberry Pi possui somente uma porta serial. Para resolver esta adversidade, foi feito um circuito multiplexador que comuta o dispositivo comunicando-se no momento com o Raspberry Pi. Este controle é feito diretamente por *software* e, como o GPS e o GSM funcionam em *threads* diferentes, um esquema de *mutex* também foi implementado para evitar que ambas as *threads* tentem acessar o mesmo recurso simultaneamente. Este *mutex* está representado pela variável 'self.lock' no código.

De todas as informações disponibilizadas pelo módulo GPS, as mais relevantes para este projeto são a latitude, a longitude, a velocidade e a direção de deslocamento. Desta forma, foi escolhida a mensagem $GPRMC para ser interpretada, pois ela possui todos estes valores.

Ao cruzar os valores obtidos do GPS destas medidas com os reais, todos eles alcançaram uma precisão suficiente para o projeto. Para mensurar esta precisão, foram coletados dados de geolocalização durante 7 minutos em um ambiente fechado em um dia nublado. O ponto real da medição está representado pelo pino no mapa da Figura 4.2. O mapa de calor em volta do pino representam as medições realizadas pelo GPS. Observa-se um erro máximo de aproximadamente 30 metros para condições adversas de coleta.

FIGURA 4.2 – Medição da precisão do módulo GPS



Fonte: (Autor, 2018)

4.2 ACELERÔMETRO

Antes de qualquer teste relactionado ao acelerômetro ser realizado, o código para a comunicação entre o Raspberry Pi e o MPU-6050 precisou ser desenvolvido. Este programa constituiu de duas classes: A classe base foi o I2CDevice, que contém operações a níveis de bit para facilitar o trabalho na escrita e leitura dos registradores I2C, conforme verificado no código no Apêndice B. Como a maior parte dos registradores do MPU-6050 são de 16 bits, e alguns registradores de configuração precisam mudar apenas alguns bits desses, esta classe provê diversas utilidades para facilitar este trabalho.

Herdando da I2CDevice está a classe MPU6050, apresentada no código do Apêndice C, que faz uso dos métodos da sua super classe, porém contém funcionalidades específicas deste módulo, como os endereços de registradores importantes, além de métodos para adquirir os valores de aceleração e velocidade angular, configurar as escalas destes, entre outros. No cenário para coleta de dados e classificação em tempo real, uma outra classe relacionada foi criada, apenas para fazer o MPU6050 funcionar em uma *thread* separada e funcionar independentemente.

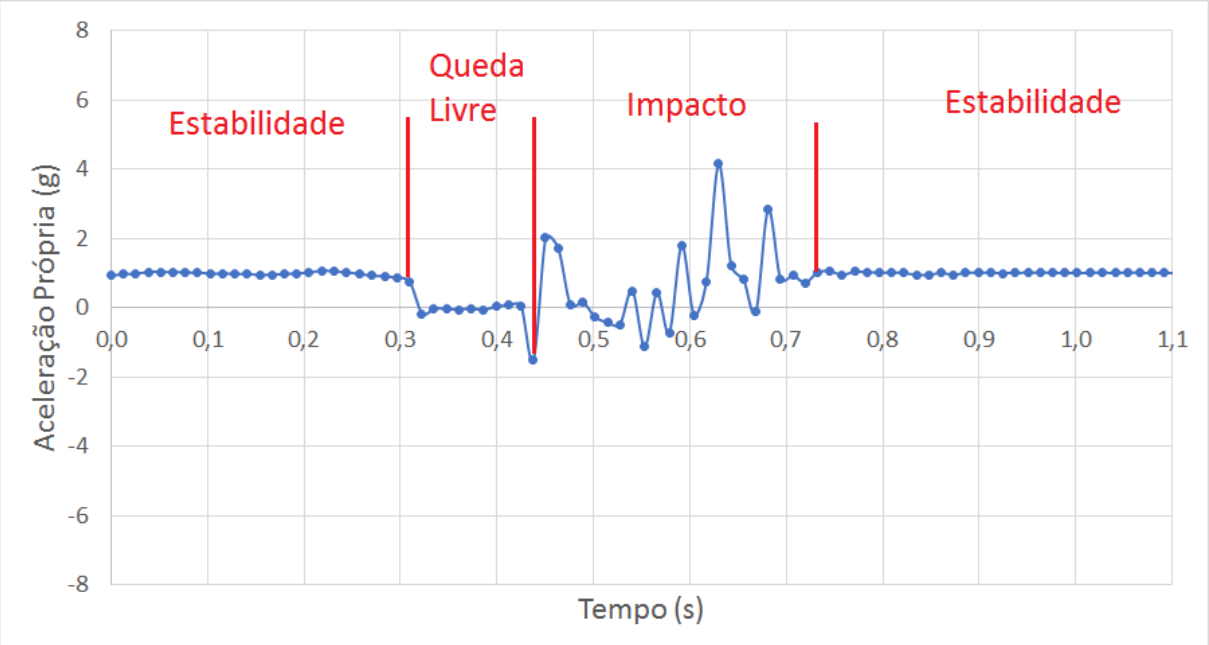
A precisão e exatidão das medidas do acelerômetro são fundamentais para o sucesso deste projeto, visto que este sensor, junto com o giroscópio, é responsável pela identificação correta da qualidade do pavimento das vias. Consequentemente, é necessário validar suas medições em pelo menos algum cenário de aceleração. Para isso, foi testado um cenário de queda livre do sensor de uma altura de 10cm.

Neste teste, os códigos relacionados ao MPU-6050 foram utilizados para a obtenção dos valores dos sensores em intervalos de 10ms. As medições do eixo Z do acelerômetro foram gravadas e podem ser observadas na Figura 4.3.

Conforme as anotações da figura, são verificadas três regiões distintas: estabilidade, queda livre e impacto. A região de estabilidade mostra os momentos em que o objeto está em repouso, e possui apenas a aceleração própria derivada da gravidade. Nestes instantes, que podem ser observados antes da queda e depois da estabilização do impacto, o eixo Z indica o valor de aproximadamente 1g.

Quando o objeto é liberado no ar, ele entra em queda livre, e sua aceleração própria diminui até atingir 0g. Enquanto o valor permanecer aproximadamente perto de zero, o objeto ainda está em queda. Consequentemente, é possível estimar (com uma

FIGURA 4.3 – Medidas do eixo Z do acelerômetro em queda livre a 10cm



Fonte: (Autor, 2018)

grande margem de erro), a altura da queda. No exemplo da Figura 4.3, o objeto caiu de 0,31s até 0,44s. Portanto, a altura da queda pode ser calculada com a Fórmula 4.1.

$$h = \frac{1}{2}gt^2 \tag{4.1}$$

Substituindo os valores na Fórmula 4.2, tem-se:

$$h = \frac{1}{2} \cdot 9,807 \cdot (0,44 - 0,31)^2 = 0,083m \tag{4.2}$$

Apesar de a altura da queda ser de 10cm, a altura calculada pelas leituras do acelerômetro foi de 8,3cm. Esta diferença se deve às imprecisões e demoras na leitura do acelerômetro, sua taxa de amostragem, e erros inerentes do próprio experimento, uma vez que a queda foi realizada manualmente. Contudo, é possível ter uma noção do tamanho da queda a partir deste cálculo nos instantes de queda livre.

A região de impacto é uma área de alta energia, com grandes picos de aceleração e alta variância. Além disso, é dependente da altura de queda, coeficientes elásticos, ângulo de impacto e superfície dos materiais. Entretanto, é possível constatar que nesta região há uma grande variação da aceleração a valores de até 4g durante um intervalo de tempo relativamente longo. Neste exemplo, as acelerações de impacto duraram aproximadamente 0,3 segundo, até finalmente estabilizar-se novamente.

4.3 GIROSCÓPIO

O giroscópio também está localizado dentro do MPU-6050, e por este motivo o mesmo código foi utilizado para a realização dos testes. Estas foram as leituras do giroscópio em ambiente sem movimentos, e depois em situação de rotação ao redor de seus eixos, a fim de verificar suas medidas.

4.3.1 Teste em ambiente estático

Para o teste estático, todo o circuito foi mantido em um ambiente sem perturbações mecânicas, onde o sensor não sofreu nenhum tipo de movimento durante dois minutos. Durante esse tempo, foram realizadas medidas de sua velocidade angular nos três eixos, salvando todas as medições em um arquivo. Após esta aquisição de dados, foram calculados o desvio padrão e a média das leituras por eixo, apresentados na Tabela 4.1. Por estar em um ambiente estático sem movimento, esperam-se valores próximos de zero.

TABELA 4.1 – Médias e desvios padrão da leitura do giroscópio

Eixo	Média ($°/s$)	Desvio Padrão
X	-1,831	0,0814
Y	1,119	0,1026
Z	-1.169	0,0874

Fonte: (Autor, 2018)

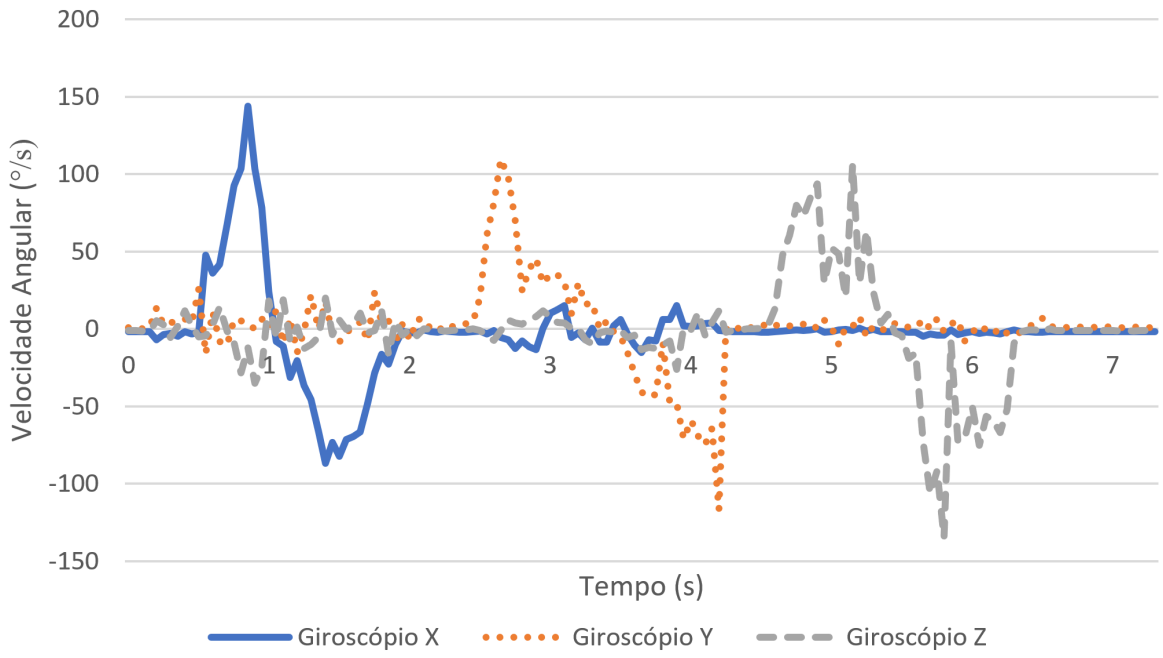
Em módulo, o *offset* do valor esperado de zero foi de aproximadamente 1, com um espalhamento baixo de aproximadamente 0,1 de desvio padrão. Como os valores em situações dinâmicas serão muito mais elevados do que 1 grau por segundo, um erro desta magnitude pode ser desconsiderado, e não deverá ter nenhum impacto relevante no objetivo final deste trabalho.

4.3.2 Teste de rotação nos eixos

Para o teste de rotação, todo o circuito foi colocado inicialmente em uma mesa em posição estática. Então, ele foi manualmente rotacionado a um ângulo de

aproximadamente 45 graus e de volta à posição inicial, em todos os eixos sequencialmente. Assim, espera-se um aumento e diminuição da velocidade angular de cada eixo independentemente. Por tratar-se de um teste manual, é possível que hajam pequenas mudanças nos valores dos outros eixos também. Por fim, para demonstrar o resultado obtido, o gráfico temporal destas medidas foi plotado na Figura 4.4. Nele, a linha sólida azul representa as medições do eixo X do giroscópio, enquanto a linha pontilhada laranja representa o eixo Y. A linha tracejada cinza, por sua vez, é o eixo Z do giroscópio.

FIGURA 4.4 – Teste de rotação do giroscópio



Fonte: (Autor, 2018)

Conforme esperado, é possível notar no gráfico dois picos de velocidade angular por eixo, um positivo e um negativo. Iniciando pelo eixo X, percebe-se a velocidade angular aumentando até aproximadamente 150 graus por segundo, e depois diminuindo até menos 100 graus por segundo. O mesmo comportamento é observado nos outros eixos em seguida, com variações da velocidade angular máxima. Independente disso, percebe-se que há uma ótima independência dos valores de medição de cada eixo, mesmo que este teste tenha sido feito manualmente. Ademais, percebe-se que os eixos estão ordenados corretamente, assim como o sentido deles.

4.4 GSM

Conforme descrito na Seção 3.3, o módulo GSM foi testado com alguns comandos AT, como para envio de mensagens, por exemplo. Porém nesta seção do trabalho, estão apresentados somente os testes referentes à comunicação TCP/IP, uma vez que esta é a única funcionalidade deste módulo relevante à este trabalho.

A leitura da documentação do SIM800L ajudou a definir os comandos AT necessários para a realização da comunicação TCP/IP através do módulo GSM. O primeiro comando executado foi o AT+CFUN=1, que habilita a completa funcionalidade RF (rádio-frequência) do GSM. A seguir, ativou-se o contexto GPRS (*General Packet Radio Services*) enviando-se AT+CGATT=1.

Após esta configuração inicial, foi executado o AT+CIPSHUT. Isto faz o GPRS fechar o contexto PDP (*Packet Data Protocol*), a fim de evitar problemas com qualquer contexto já iniciado anteriormente. Ainda, a instrução AT+CIPMUX=0 foi utilizada para configurar para uma única conexão IP. O modo com múltiplas conexões também poderia ser utilizado, mas foi optado pelo modo de conexão única por motivos de simplicidade. Perceba que os dois últimos comandos iniciaram-se com 'CIP', que é um prefixo comum para comandos do *toolkit* de aplicação TCP/IP da SIMCom.

A seguir foram definidas as configurações do APN (*Access Point Name*) através do comando AT+CSTT=APN,USER,PASS. Os parâmetros APN, USER e PASS são especificados de acordo com a operadora utilizada. No caso deste trabalho em que a operadora Vivo foi utilizada, os parâmetros foram 'zap.vivo.com.br', 'vivo' e 'vivo', respectivamente.

Finalmente, pode-se iniciar a conexão GPRS com a instrução AT+CIICR. Esta instrução depende da disponibilidade de sinal da operadora. AT+CIFSR também pode ser processado para verificar o endereço IP obtido. Para conectar com um dispositivo na *Internet*, foi então usado o comando AT+CIPSTART. Este aceita três argumentos: o tipo de conexão (TCP ou UDP), o host a se conectar (endereço IP ou domínio), e a porta. No caso deste trabalho, foi verificado o funcionamento de uma conexão com um servidor *Web*, então o protocolo TCP foi utilizado na porta 80. Por fim, para enviar dados por esta conexão, usa-se a instrução AT+CIPSEND seguido do número de bytes a ser enviado. Depois, é possível escrever estes dados para o modem GSM, terminando a sequência com o caractere hexadecimal '\x1A'. A resposta do servidor

será enviada então pela serial.

O código para envio e recebimento de uma requisição HTTP pode ser visto no Apêndice D. Este foi o módulo utilizado no código principal para envio das classificações de estrada para o servidor na *Internet*. Caso haja algum problema no envio de qualquer um desses comandos, o método irá levantar a exceção GsmException, que pode ser então tratada no código principal.

4.5 COLETA DE DADOS

Na Seção 3.5 foi discorrido o método para a coleta das leituras manuais da qualidade de pavimentos em diversas vias diferentes. Foi definida a classificação delas para esta etapa em 'Bom', 'Regular' e 'Ruim'. Os dados foram gravados com todas as leituras dos sensores em intervalos de 10ms, de forma que qualquer parâmetro do algoritmo de pós-processamento pudesse ser mudado posteriormente.

Como todos os dados foram recordados ao longo de 2 semanas, o tamanho do arquivo foi relativamente grande, de cerca de 280 MB. Por que o salvamento deste registro foi feito por completo, foi possível observar comportamentos diversos que auxiliaram no posterior tratamento de alguns tipos de erro, como sinais de GPS inválidos, algumas medições pontuais sem sentido, como uma velocidade momentânea de 2000 km/h, dados corruptos, entre outras situações.

Por fim, o pós-processamento deste arquivo de 280 MB foi realizado. Nele, as medições foram acumuladas em um intervalo de 10 segundos, e então extraída a média e variância de todos os eixos do acelerômetro, giroscópio, e da velocidade. Velocidades abaixo de 15km/h foram descartadas, assim como momentos de interrupção do sinal de GPS. Ainda, caso a qualidade do pavimento fosse alterada durante os 10 segundos, o vetor de valores acumulados era zerado, a fim de não misturar leituras de cunho diferente. Assim, o resultado final foi um arquivo mais comprimido de 40 MB, com mais de 11 horas de dados relevantes prontos para serem inseridos no algoritmo de SVM.

4.6 ALGORITMOS DE CLASSIFICAÇÃO

Para realizar a avaliação da qualidade das SVMs propostas, foi utilizada a biblioteca scikit-learn na linguagem Python, que é de uso muito comum na área de ciência de dados. Esta biblioteca já possui implementações de diversos algoritmos de inteligência

artificial, inclusive dos classificadores lineares, polinomiais e RBF, empregados neste trabalho.

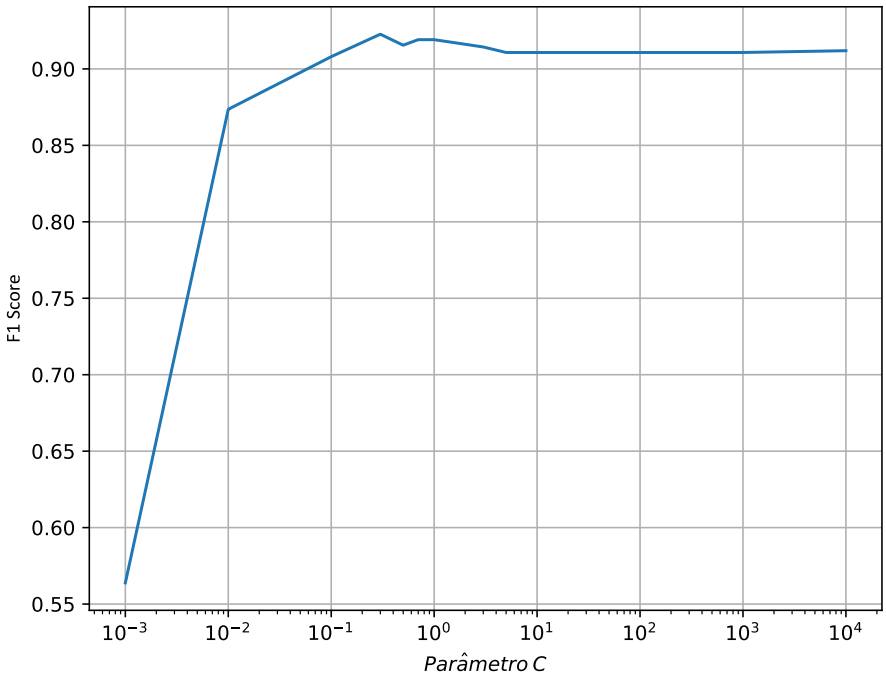
O código do Apêndice E foi o código final utilizado para o treinamento e avaliação da performance da SVM. Neste código, é possível observar alguns dos conceitos explicados anteriormente neste trabalho, como a separação entre os dados de treino e de teste, a varredura para busca dos hiper-parâmetros C e γ, além do treinamento e aplicação da SVM propriamente ditos. Como *features* para os classificadores, foram utilizados os indicadores de média e variância das seguintes medidas: aceleração nos eixos X, Y e Z; velocidade angular nos eixos X, Y e Z; e a velocidade linear do veículo. Desta forma, totalizaram-se então 14 *features* neste momento. Posteriormente foi testada a fusão de todos os três eixos em uma única medida através de seu módulo, de forma a evitar a necessidade de uma orientação específica do dispositivo, totalizando então em 6 *features*. Antes da aplicação das *features* no algoritmo, os dados foram normalizados para possuírem média 0 e variância 1 por classe.

4.6.1 Classificador Linear

O classificador linear, por ser o mais simples computacionalmente entre os três classificadores abordados neste trabalho, foi testado primeiramente. Para encontrar o modelo com melhor performance, foi realizada a varredura do hiper-parâmetro C, avaliando o seu F1 Score contra os dados de teste. Esta varredura foi feita em escala logarítmica, variando o valor de C entre 10^{-3} até 10^{4}. Na região mais próxima dos valores mais altos, foi feita então uma busca mais refinada pelo valor máximo. A Figura 4.5 ilustra os resultados obtidos para o caso com 3 classes, ou seja, para a metodologia 1. Nela, observa-se o valor máximo do F1 Score em 0,9227, para um C igual a 0,3. Portanto, este tornou-se o valor de referência para este classificador.

Após a especificação do valor de C, elaborou-se então a matriz de confusão para este classificador. Esta matriz está presente na Tabela 4.2. Observa-se que a maior parte dos dados está concentrada na classe 'Bom', devido à natureza das ruas onde os dados foram coletados. Independente disso, os índices de precisão e recall deste cenário foram muito bons, sendo o valor mínimo de recall de 81,38% na classe 'Regular', e o valor mínimo de precisão de 90,53% também na classe 'Regular'. Não à toa, por ser uma classe intermediária, que está mais sujeita a subjetividade, foi a que

FIGURA 4.5 – F1 Score em função de C - Linear 3 classes



Fonte: (Autor, 2018)

TABELA 4.2 – Matriz de confusão para o classificador linear com 3 classes

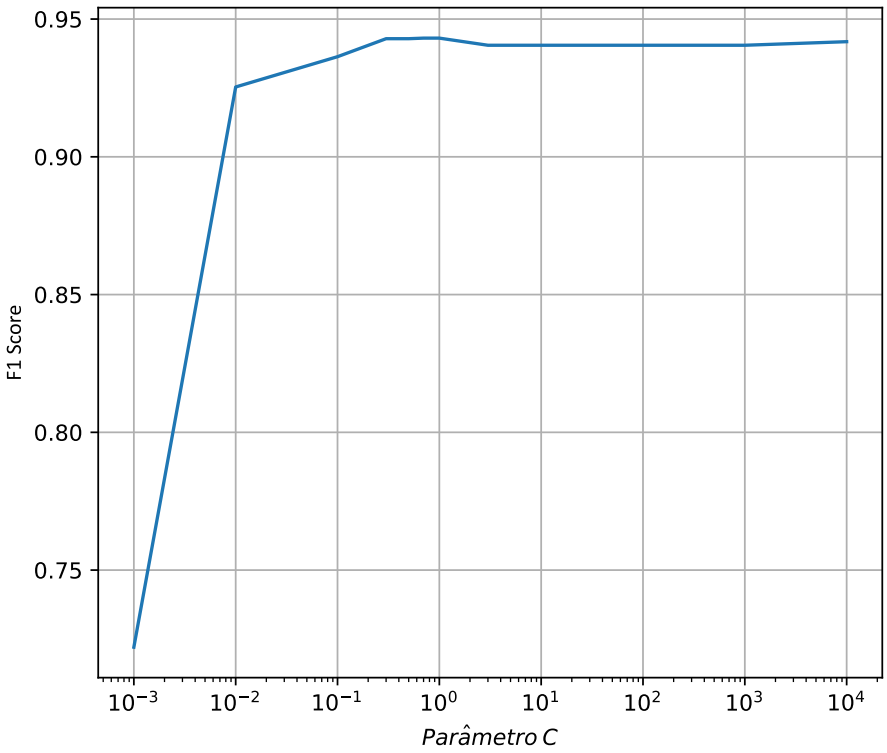
Previsto \ Atual	Ruim	Regular	Bom
Ruim	55	4	1
Regular	2	153	14
Bom	2	31	1765

Fonte: (Autor, 2018)

apresentou pior desempenho, ainda que dentro de valores aceitáveis.

Ainda trabalhando com o classificador linear, foi avaliada a metodologia 2, em que as classes ’Regular’ e ’Ruim’ fundem-se e tornam-se uma. O procedimento de avaliação foi o mesmo para a metodologia 1, encontrando primeiramente o valor de C através da maximização do F1 Score, e a seguir montando-se a matriz de confusão. A Figura 4.6 ilustra os diferentes valores de F1 Score obtidos para cada C diferente. Neste caso, o valor de C ideal foi superior ao encontrado para a metodologia 1, assumindo o valor de 0,7, para um F1 Score de 0,9431.

FIGURA 4.6 – F1 Score em função de C - Linear 2 classes



Fonte: (Autor, 2018)

Uma nota que deve ser feita é que apesar de este método ter apresentado um F1 Score superior à metodologia 1, eles não podem ser comparados diretamente, devido à diferença do número de classes. Portanto, eles representam informações de cunho diferente.

A seguir, foi montada a matriz de confusão para esta metodologia. Diferentemente da metodologia 1, esta possui somente 2 colunas e 2 linhas, devido ao número de classes, que foi diminuído para 2. A Tabela 4.3 mostra esta matriz para C igual a 0,7, conforme encontrado na etapa anterior.

Desta tabela, percebe-se um recall de 0,8704 e uma precisão de 0,9307. De maneira geral, este modelo possui um desempenho muito bom, considerando as variadas condições da coleta do sinal, e a não criticidade de valores como falso-positivos e falso-negativos.

4.6.2 *Kernel* Polinomial

O classificador polinomial pode assumir diversos graus, porém este trabalho abordou o polinômio de grau 2 somente. Neste tipo de *kernel*, os parâmetros relevantes

TABELA 4.3 – Matriz de confusão para o classificador linear com 2 classes

Previsto \ Atual	Ruim/Regular	Bom
Ruim/Regular	215	16
Bom	32	1764

Fonte: (Autor, 2018)

para varredura são o C e o γ. Desta forma, o gráfico da maximização do indicador F1 passa a ser tridimensional, ao contrário do bidimensional apresentado na Seção 4.6.1. Assim, esta visualização do F1 Score foi feita em formato de tabelas. Similarmente, os parâmetros foram variados na escala logarítmica de 10^{-3} até 10^{3}. A escolha por um valor de limite máximo menor deve-se à dificuldade do classificador polinomial treinar modelos com valores muito altos de C e γ, potencialmente causado por instabilidades matemáticas na estimação.

Também pelo método como o classificador polinomial funciona, uma pequena modificação precisou ser feita na normalização dos dados. Ao invés de normalizá-los para manter a média em 0 e a variância em 1, um escalador mínimo-máximo foi utilizado, onde as *features* são escaladas para possuírem valores de -1 até 1.

Primeiramente foi avaliada a metodologia 1, com 3 classes. Os valores de F1 para cada C e γ diferente pode ser encontrado na Figura 4.7.

FIGURA 4.7 – F1 Score em função de C e γ - Polinomial 3 classes

γ \ C	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3
1e-4	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117
1e-3	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117
1e-2	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.5312	0.8147
1e-1	0.3117	0.3117	0.3117	0.3117	0.5312	0.8147	0.8955	0.9224
1e0	0.3117	0.3117	0.5312	0.8147	0.8955	0.9224	0.9053	0.8514
1e1	0.5312	0.8147	0.8955	0.9224	0.9053	0.8514	0.8322	0.8226
1e2	0.8955	0.9224	0.9053	0.8514	0.8318	0.8216	0.8059	0.8011
1e3	0.9053	0.8514	0.8306	0.8283	0.8197	0.8073	0.8087	0.8079

Fonte: (Autor, 2018)

As células apresentadas em azul claro empataram como as que apresentaram

o máximo valor de F1, em 0,9224. São elas as com C=0,001 e γ=100; C=0,1 e γ=10; C=10 e γ=1; e C=1000 e γ=0,1. Foi preferido escolher um dos valores mais centrais como parâmetro para este modelo, e por isto a matriz de confusão apresentada na Tabela 4.4 foi elaborada para os valores de C=0,1 e γ=10.

TABELA 4.4 – Matriz de confusão para o classificador polinomial com 3 classes

Previsto \ Atual	Ruim	Regular	Bom
Ruim	54	1	3
Regular	3	155	15
Bom	2	32	1762

Fonte: (Autor, 2018)

A classe com o pior desempenho foi novamente a 'Regular', com um recall de 82,45% e uma precisão de 89,60%. Este classificador apresentou um desempenho pior do que o linear, com um decréscimo do F1 Score de 0,9431 para 0,9224. Em seguida, foi treinado o *kernel* polinomial para a metodologia 2, com 2 classes somente. Novamente, a normalização foi feita escalando os dados de -1 a 1. Os resultados da procura do melhor valor de F1 deste classificador estão expostos na Figura 4.8.

FIGURA 4.8 – F1 Score em função de C e γ - Polinomial 2 classes

γ \ C	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3
1e-4	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676
1e-3	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676
1e-2	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.6236	0.8881
1e-1	0.4676	0.4676	0.4676	0.4676	0.6236	0.8881	0.9302	0.9370
1e0	0.4676	0.4676	0.6236	0.8881	0.9302	0.9370	0.9349	0.9301
1e1	0.6236	0.8881	0.9302	0.9370	0.9349	0.9312	0.9317	0.9109
1e2	0.9302	0.9370	0.9349	0.9301	0.9296	0.9143	0.9031	0.9089
1e3	0.9349	0.9301	0.9294	0.9106	0.9161	0.9121	0.9135	0.9122

Fonte: (Autor, 2018)

Interessantemente, os valores máximos de F1 para 2 classes aconteceram para os mesmos valores dos parâmetros γ e C. O valor máximo de F1 para o *kernel*

polinomial de 2 classes foi de 0,9370. Sendo assim, os mesmos C=0,1 e γ=10 foram utilizados para a elaboração da matriz de confusão, disponível na Tabela 4.5.

TABELA 4.5 – Matriz de confusão para o classificador polinomial com 2 classes

Previsto \ Atual	Ruim/Regular	Bom
Ruim/Regular	212	18
Bom	35	1762

Fonte: (Autor, 2018)

Comparando o desempenho deste classificador ao linear, ele apresentou pioras em ambas as metodologias 1 e 2. Na metodologia 2, o classificador polinomial mostrou um F1 Score de 0,9370, o que é menor do que o 0,9431 encontrado para o linear. Além disso, o polinomial apresentou um tempo maior para treinamento, juntamente com uma maior complexidade de algoritmo, não se mostrando vantajoso para a aplicação como classificador.

4.6.3 *Kernel* RBF

O *kernel* RBF, assim como o polinomial, possui ambos os parâmetros γ e C para otimização. Seu treinamento, entretanto, foi mais veloz do que o polinomial, e por isso foi possível aumentar o intervalo dos valores de varredura para o intervalo de 10^{-4} até 10^{4}.

Para o treinamento do *kernel* RBF, a normalização dos dados foi realizada para resultar em média 0 e variância 1. A disposição dos valores de F1 Score conforme a varredura dos hiper-parâmetros foi feita da mesma forma do *kernel* polinomial. O resultado então para a metodologia 1, com 3 classes, pode ser visualizado na Figura 4.9, no mesmo formato de visualização em tabela.

De todos os valores analisados de C e γ, o modelo que apresentou melhor resultado foi para C=100 e γ=0,001, com um F1 Score de 0,9329. Este modelo apresentado mais detalhadamente pode ser observado na Tabela 4.6, onde a matriz de confusão para este classificador foi desenhada.

FIGURA 4.9 – F1 Score em função de C e γ - RBF 3 classes

γ \ C	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3	1e4
1e-4	0.3117	0.3117	0.3117	0.3117	0.3633	0.7151	0.8824	0.9180	0.9167
1e-3	0.3117	0.3117	0.3117	0.3442	0.7177	0.8843	0.9329	0.9092	0.8748
1e-2	0.3117	0.3117	0.3117	0.6690	0.8843	0.9134	0.8684	0.8483	0.8243
1e-1	0.3117	0.3117	0.3117	0.5408	0.8734	0.8681	0.8452	0.7941	0.7748
1e0	0.3117	0.3117	0.3117	0.3117	0.4283	0.4587	0.4563	0.4563	0.4563
1e1	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117
1e2	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117
1e3	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117
1e4	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117

Fonte: (Autor, 2018)

Para a metodologia 1, comparando somente o F1 Score, percebe-se prontamente que o *kernel* RBF apresentou os melhores resultados quando comparados ao *kernel* linear e polinomial de segundo grau. Com este modelo, percebe-se uma melhora de desempenho na classe 'Regular', que foi a que apresentou piores resultados anteriormente. Aqui, a classe supracitada apresentou um recall de 0,8764 e uma precisão de 0,8914.

TABELA 4.6 – Matriz de confusão para o classificador RBF com 3 classes

Previsto \ Atual	Ruim	Regular	Bom
Ruim	54	3	1
Regular	3	156	16
Bom	2	29	1763

Fonte: (Autor, 2018)

A seguir, o *kernel* RBF foi implementado para a metodologia 2, com 2 classes. O resultado foi apresentado também em formato de tabela na Figura 4.10.

FIGURA 4.10 – F1 Score em função de C e γ - RBF 2 classes

γ \ C	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3	1e4
1e-4	0.4676	0.4676	0.4676	0.4676	0.5857	0.8549	0.9294	0.9405	0.9431
1e-3	0.4676	0.4676	0.4676	0.5928	0.8577	0.9305	0.9438	0.9368	0.9309
1e-2	0.4676	0.4676	0.4676	0.8225	0.9292	0.9442	0.9325	0.9236	0.9132
1e-1	0.4676	0.4676	0.4676	0.7958	0.9329	0.9377	0.9082	0.8821	0.8639
1e0	0.4676	0.4676	0.4676	0.4676	0.6342	0.6614	0.6619	0.6619	0.6619
1e1	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676
1e2	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676
1e3	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676
1e4	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676

Fonte: (Autor, 2018)

Para o classificador RBF com 2 classes, o maior valor de F1 foi de 0,9442, uma leve melhora comparado ao classificador linear, que tinha sido de 0,9431. Diferentemente do polinomial, o *kernel* RBF para a metodologia 2 maximizou seu F1 Score em valores diferentes de C e γ, em 10 e 0,01 respectivamente. A matriz de confusão deste modelo está apresentada na Tabela 4.7.

TABELA 4.7 – Matriz de confusão para o classificador RBF com 2 classes

Previsto \ Atual	Ruim/Regular	Bom
Ruim/Regular	216	17
Bom	31	1763

Fonte: (Autor, 2018)

À vista de facilitar a comparação e visualização da performance de cada *kernel* e metodologia, foi concebida a Tabela 4.8, que contém essas informações resumidas, apresentando o F1 Score de cada cenário. Nesta tabela, torna-se visível a superioridade do *kernel* RBF em termos do F1 Score, que é um indicador que agrega as informações mais importantes de um classificador, o recall e a precisão.

Para o cenário com três classes, os hiper-parâmetros com maior F1 Score foram C=100 e γ=0,001. No caso de considerar-se duas classes somente, os valores dos parâmetros escolhidos foram C=10 e γ=0,01. Para aplicação no algoritmo de

avaliação da qualidade de estradas em tempo real, estes modelos são os preferíveis em termos de qualidade.

TABELA 4.8 – Comparação de desempenho entre os classificadores

	3 Classes	2 Classes
Linear	0,9227	0,9431
Polinomial	0,9224	0,9370
RBF	0,9329	0,9442

Fonte: (O Autor, 2018)

4.6.4 Independência de eixos

Conforme explicado na Seção 4.6, as *features* utilizadas nos classificadores anteriores utilizaram as medidas de cada eixo individualmente, e assim uma reorientação do dispositivo no carro poderia causar classificações incorretas. Uma forma de prevenir uma interferência desta reorientação é extrair o módulo das medidas de todos os eixos, e utilizá-las no classificador. Assim, não importa a orientação deste dispositivo, pois o módulo será sempre o mesmo independente da situação. Esta seção visa apresentar os resultados de um possível classificador com esta característica, utilizando o *kernel* RBF. Este *kernel* foi escolhido pois foi o que apresentou o melhor desempenho nos testes anteriores.

Assim como performado na Seção 4.6.3, foi realizada uma varredura dos parâmetros C e γ para determinação do melhor F1 Score, e cujos resultados podem ser encontrados nas Figuras 4.11 e 4.12, para 3 e 2 classes, respectivamente.

Como se está retirando *features* do sistema, e consequentemente informações relevantes, espera-se que a performance dos modelos com independência de eixo sejam piores. Deveras, percebe-se que o F1 Score do modelo com 3 classes caiu de 0,9329 para 0,8982 para a metodologia 1, e de 0,9442 para 0,9267 para a metodologia 2.

Para um produto que poderia ser instalados em diversos veículos, esta característica de independência de eixos pode ser muito importante, e este resultado mostra que apesar de perder um pouco de desempenho, este novo modelo ainda mostra-se eficiente para avaliação da qualidade de pavimentos.

FIGURA 4.11 – F1 Score em função de C e γ - RBF 3 classes com independência de eixo

γ \ C	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3	1e4
1e-4	0.3117	0.3117	0.3117	0.3117	0.3117	0.5316	0.8436	0.8821	0.8860
1e-3	0.3117	0.3117	0.3117	0.3117	0.5315	0.8436	0.8809	0.8818	0.8822
1e-2	0.3117	0.3117	0.3117	0.5322	0.8460	0.8806	0.8840	0.8829	0.8663
1e-1	0.3117	0.3117	0.3117	0.8124	0.8655	0.8982	0.8664	0.8431	0.8117
1e0	0.3117	0.3117	0.3117	0.5199	0.8321	0.8274	0.7880	0.7625	0.6881
1e1	0.3117	0.3117	0.3117	0.3117	0.4103	0.4249	0.4240	0.4238	0.4238
1e2	0.3117	0.3117	0.3117	0.3117	0.3117	0.3152	0.3151	0.3151	0.3151
1e3	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117
1e4	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117	0.3117

Fonte: (Autor, 2018)

FIGURA 4.12 – F1 Score em função de C e γ - RBF 2 classes com independência de eixo

γ \ C	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3	1e4
1e-4	0.4676	0.4676	0.4676	0.4676	0.4676	0.6904	0.9149	0.9243	0.9254
1e-3	0.4676	0.4676	0.4676	0.4676	0.6853	0.9149	0.9254	0.9236	0.9238
1e-2	0.4676	0.4676	0.4676	0.6913	0.9188	0.9267	0.9238	0.9192	0.9171
1e-1	0.4676	0.4676	0.4676	0.8912	0.9211	0.9257	0.9140	0.9068	0.9020
1e0	0.4676	0.4676	0.4676	0.7765	0.9096	0.9137	0.9028	0.8667	0.8335
1e1	0.4676	0.4676	0.4676	0.4676	0.6070	0.6364	0.6312	0.6299	0.6299
1e2	0.4676	0.4676	0.4676	0.4676	0.4676	0.4716	0.4714	0.4714	0.4714
1e3	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676
1e4	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676	0.4676

Fonte: (Autor, 2018)

4.7 DETECÇÃO EM TEMPO REAL

A detecção em tempo-real, conforme explicado na Seção 3.7, consiste na aplicação de um algoritmo que capture continuamente as leituras do acelerômetro, giroscópio e GPS durante um período de 10 segundos, extraia indicadores dessas medidas e as aplique em um dos modelos de inteligência artificial obtido na Seção 4.6.1. Após a classificação obtida pela SVM, o sistema deverá então enviar este resultado para um servidor na *Internet* através de um módulo GSM. Será responsabilidade do servidor então armazenar estas informações e disponibilizá-las para usuários.

O fluxo principal desta lógica acontece no código do Apêndice F. Neste, pode-se notar a presença de elementos citados anteriormente neste trabalho, como a configuração das *threads* do MPU-6050 e do GPS, além do carregamento do modelo da SVM. Para sincronizar o funcionamento de ambos o GPS e o GSM, que funcionam na mesma porta serial, o código principal utilizou uma combinação de um *mutex* e um multiplexador, que seleciona o dispositivo a ser lido no momento.

A classe MeasurementProcessor, mostrada no Apêndice G, é a responsável por acumular as medidas durante os 10 segundos seguidos, ou descartá-las conforme as condições dos sensores. Ela também é responsável por extrair os indicadores utilizados na SVM.

A fim de visualizar facilmente o resultado atual da SVM, três LEDs podem ser acesos de acordo. A cor do LED aceso representa a qualidade do pavimento dos últimos 10 segundos.

Por fim, uma requisição HTTP é montada com a detecção a ser enviada ao servidor e despachada para o módulo GSM. Caso haja uma falha na transmissão dessa requisição por qualquer motivo, a detecção é armazenada em um *buffer* e daqui 10 segundos será transmitida novamente junto com a nova detecção.

4.8 SERVIDOR

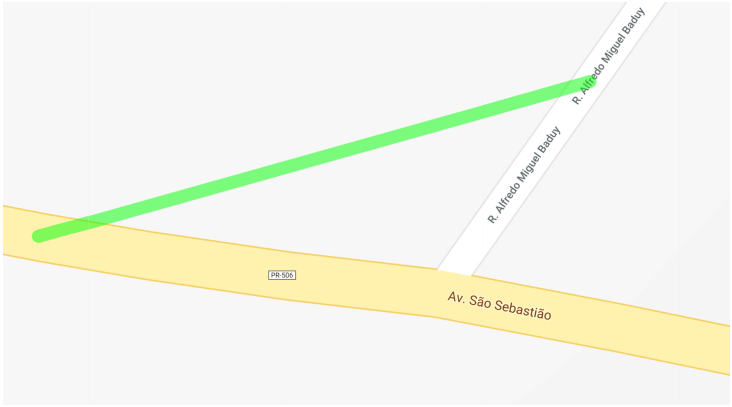
Para receber os dados do sistema embarcado, foi implementada uma API *Web* para a inserção e aquisição de detecções. Para obter as detecções presentes no servidor utiliza-se o protocolo HTTP (*Hypertext Transfer Protocol*) com uma requisição GET, e para adicionar uma nova detecção utiliza-se uma requisição POST passando os dados nos parâmetros. Adicionalmente, filtros como intervalos de latitude e longitude podem ser estipulados nas requisições GET também.

Os dados no servidor são armazenados em um banco de dados MySQL, em duas tabelas. A primeira delas chama-se *detections*, e possui as informações da qualidade do pavimento, a velocidade do automóvel, as coordenadas de início e do fim da detecção.

Contudo, esses dois pares de coordenadas não são suficiente para representar o caminho percorrido pelo automóvel do início até o fim da detecção. No caso de uma rua com curvas, por exemplo, traçar uma reta entre o ponto inicial e final traria um

resultado visualmente ruim, conforme pode ser visto na Figura 4.13.

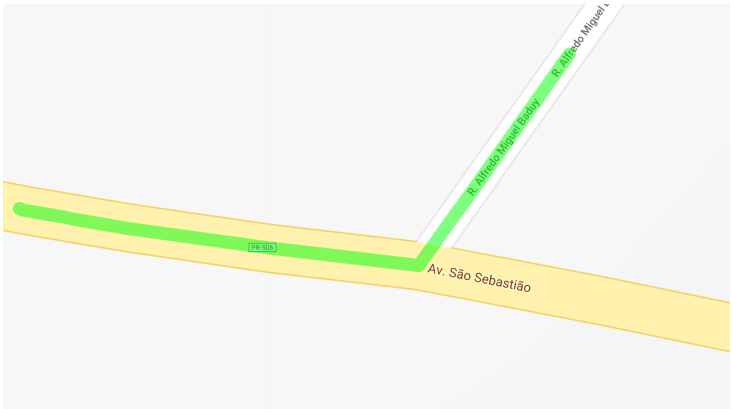
FIGURA 4.13 – Visualização de detecção sem interpolação



Fonte: (Autor, 2018)

Para resolver este problema, o servidor utilizou-se da API Roads, fornecida pela Google, para formar um caminho coerente dentro de autovias, através da interpolação de alguns pontos. Esta informação do caminho percorrido durante uma detecção é armazenada em uma tabela separada, a detections_path. No momento de mostrar as detecções para o usuário final, um *join* é realizado entre estas tabelas para mostrar o resultado propriamente formatado. Assim, a Figura 4.14 mostra a mesma detecção da Figura 4.13, porém após o tratamento com esta API do Google. Todo o código implementado no servidor, que realiza a comunicação com a API do Google e adiciona e recupera os registros do banco de dados, pode ser encontrado nos Apêndices H e I.

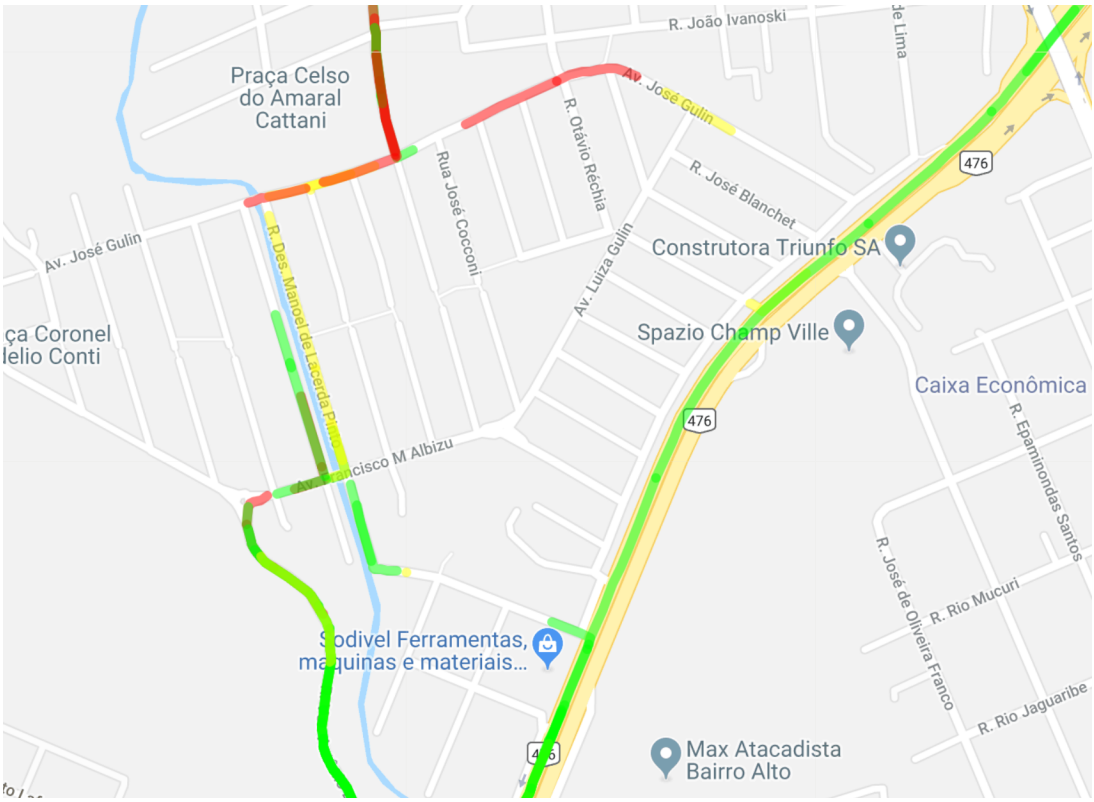
FIGURA 4.14 – Visualização de detecção com interpolação



Fonte: (Autor, 2018)

Uma vez que o usuário agora possui as coordenadas corretas das detecções e a qualidade do pavimento, foi possível utilizar uma API do Google Maps para facilitar esta visualização. O resultado final é apresentado na Figura 4.15. As estradas em verde estão em condições boas, as amarelas estão regulares, e as em vermelho estão em condições ruins.

FIGURA 4.15 – Resultado final da visualização das detecções



Fonte: (Autor, 2018)

5 CONCLUSÃO

Com o crescimento das cidades e áreas urbanas no Brasil, a demanda por automóveis e novas formas de locomoção tem aumentado muito nos últimos anos. Nos últimos 10 anos, houve um crescimento de 87% no tamanho da frota de veículos brasileira, passando de 53 milhões de veículos em 2008 para 99 milhões em 2018 (DENATRAN, 2018).

Este aumento do número de veículos nas ruas, além de causar maior tráfego nas cidades, ainda acelera o desgaste dos pavimentos urbanos, o que ocasiona em um número maior de acidentes e um gasto de manutenção maior nos veículos. Estudos do CNT mostram que apenas 7% dos pavimentos do país são avaliados, sendo 48% avaliados como regular, ruim ou péssimo (CNT, 2018). Mesmo dada a larga extensão de 1,5 milhão de km da malha rodoviária brasileira, este trabalho de avaliação das condições de pavimento continua sendo feito manualmente.

Desta forma, a fim de automatizar este processo, propôs-se a integração de um Raspberry Pi e sensores como acelerômetros e giroscópios para avaliação a condição de pavimentos através de um classificador como a SVM. Por fim, os dados coletados deveriam ser enviados a um servidor na *Internet* utilizando um módulo GSM.

A primeira etapa para o desenvolvimento deste trabalho foi a validação dos dados dos sensores e do *software* criado para sua leitura. Foram estudados o funcionamento do acelerômetro, giroscópio, GPS e GSM, sem nenhum problema encontrado.

Após garantir a operabilidade desses módulos em conjunto, foi então desenvolvido um plano de coleta de dados para treinamento da SVM. Foram coletados mais de 11 horas de dados válidos distribuídos em 4060 *data points*, com pavimentos classificados como 'Bom', 'Regular' ou 'Ruim'. Cada *data point* consistiu da média e variância dos acelerômetros e giroscópios em seus três eixos, além da velocidade do automóvel e sua geolocalização ao longo de 10 segundos.

De todos os *kernels* avaliados neste trabalho, o que apresentou melhor desempenho para os dados obtidos foi o RBF: para as três classes originais, ele apresentou um F1 Score de 0,9329 para C=100 e γ=0,001. Foi também avaliado o desempenho unindo-se as classes 'Ruim' e 'Regular', e este mesmo *kernel* apresentou um F1 Score de 0,9442 para este cenário, com os valores de C=10 e γ=0,01.

Para evitar a necessidade de uma orientação específica do dispositivo, um outro conjunto de *features* também foi avaliado, onde os três eixos dos acelerômetros e giroscópios foram agrupados em uma única medida através do cálculo de seu módulo. Neste caso, apesar de esperado, o classificador obteve um desempenho um pouco inferior, apresentando um F1 Score de 0,8982 para o cenário com três classes e 0,9267 para o cenário com 2 classes.

Ainda assim, em ambos os casos o classificador apresentou resultados bons o suficiente para o objetivo de classificação de qualidade de pavimentos, que é uma atividade inerentemente subjetiva. Falsos positivos ou falsos negativos pontuais também não são um problema muito sério, uma vez que eles podem ser corrigidos posteriormente por uma eventual redundância de avaliações.

A avaliação em tempo-real e o envio das detecções através do GSM para o servidor também foi desenvolvida e testada com sucesso. Um banco de dados do lado do servidor armazena e disponibiliza as informações coletadas para a *Internet*, democratizando assim o acesso à informação de qualidade dos pavimentos das vias brasileiras ao cidadão.

5.1 TRABALHOS FUTUROS

Ainda que o desempenho final da SVM não tenha sido ruim para a classificação de pavimento, é possível melhorá-lo através do estudo e aplicação de algumas técnicas. A utilização de filtros após a coleta de dados, por exemplo, é uma técnica que pode auxiliar a melhorar o desempenho do classificador. É sugerida a utilização de filtros passa-alta no acelerômetro e giroscópio, para que sejam mantidos somente os sinais de alta energia relevantes para este caso.

Além disso, também seria interessante a avaliação de performance do classificador em outros modelos de automóveis. É possível que conjuntos diferentes de suspensão e amortecimento afetem na classificação por acelerômetros, e deseja-se estudar o quanto de impacto esse fator possa ter na qualidade do modelo de classificação.

Outra sugestão seria a utilização de outros algoritmos de inteligência artificial para classificação, como redes neurais por exemplo, ou até mesmo um outro conjunto de *features* extraídas. Para isto, podem ser utilizados diversos outros indicadores, como

os valores máximos de aceleração, a derivada desta, outros momentos estatísticos, entre outros.

Finalmente, poderia também ser estudada a aplicação deste algoritmo em um aplicativo de celular. A facilidade do celular é que este já possui todos os sensores utilizados no trabalho e a plataforma de programação facilita o desenvolvimento de um *software* para realizar esta detecção. Entretanto, o uso de um aplicativo como este que fica constantemente lendo os sensores e processando dados pode gastar muita bateria. Além disso, como o celular não estará fixo dentro do carro, falsos positivos podem ocorrer com muita frequência caso o usuário esteja manipulando o celular manualmente. Estes efeitos também devem ser estudados e avaliados.

REFERÊNCIAS

14CORE. ***Wiring the MPU 6050 Sensor Mems Accelerometer GYRO***. 2018. <http://www.14core.com/wiring-the-mpu-6050-sensor-accelerometer-gyro-with-nano/>. Acesso em 9 jul. 2018.

ANDREJASIC, M. ***MEMS Accelerometers***. 2008. <http://mafija.fmf.uni-lj.si/seminar/files/2007_2008/MEMS_accelerometers-koncna.pdf>. Acesso em 02 out. 2017.

APOSTOLYUK, V. ***Theory and Design of Micromechanical Vibratory Gyroscopes***. 2006. <http://www.apostolyuk.com/files/papers/Springer.pdf>. Acesso em 15 mar. 2018.

BADDELEY, G. ***GPS - NMEA sentence information***. 2001. <http://aprs.gids.nl/nmea/>. Acesso em 7 jul. 2018.

BALBINOT, A.; BRUSAMARELLO, V. ***Instrumentação e Fundamentos de Medidas***. 2. ed. [S.l.]: LTC, 2011.

BETKE, K. **The NMEA 0183 Protocol**. *http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf*, 2000.

BURNETT, C. ***Sample I2C schematic***. 2018. <https://pt.wikipedia.org/wiki/I%C2%B2C#/media/File:I2C.svg>. Acesso em 7 jul. 2018.

CASPERSEN, K. M. ***What is the influence of C in SVMs with linear kernel?*** 2018. <https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel>. Acesso em 2 out. 2018.

CNT. ***Pesquisa CNT de Rodovias 2016***. 2016. <http://pesquisarodoviascms.cnt.org.br/Relatorio%20Geral/Pesquisa%20CNT%20(2016)%20-%20LOW.pdf>. Acesso em 02 out. 2017.

____. ***Anuário CNT do Transporte***. 2018. <http://anuariodotransporte.cnt.org.br/2018/Inicial>. Acesso em 22 ago. 2018.

DEALEXTREME. ***GY-NEO6MV2 Flight Controller GPS Module - Blue***. 2018. <http://www.dx.com/p/gy-neo6mv2-flight-controller-gps-module-blue-232595#.W0OzJflKiUk>. Acesso em 9 jul. 2018.

DENATRAN. ***Frota de Veículos***. 2018. <http://www.denatran.gov.br/index.php/estatistica/237-frota-veiculos >. Acesso em 25 out. 2018.

DURDA, F. ***Serial and UART Tutorial***. 1996. <https://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/ >. Acesso em: 27 out. 2017.

ERIKSSON, J. et al. **The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring**. ACM, New York, NY, USA, p. 29–39, 2008.

ESCOLARES, T. M. conteúdos. ***Transportes no Brasil***. 2015. <https://www.todamateria.com.br/transportes-no-brasil/>. Acesso em 08 out. 2017.

FOUNDATION, R. P. ***Raspberry Pi 3 Model B***. 2017. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acesso em: 24 out. 2017.

GROSSE, R. ***Linear Classification***. 2017. <http://www.cs.toronto.edu/ƒrgrosse/courses/csc321_2017/readings/L03%20Linear%20-Classifiers.pdf>. Acesso em 04 mai. 2018.

IDRIS, A. ***Confusion Matrix***. 2018. <https://medium.com/@awabmohammedomer/confusion-matrix-b504b8f8e1d1 >. Acesso em 3 out. 2018.

INFO, I. ***I2C Bus Specification***. 2018. <http://i2c.info/i2c-bus-specification>. Acesso em 7 jul. 2018.

INFORMATION, G. ***NMEA data***. 2015. <http://www.gpsinformation.org/dale/nmea.htm>. Acesso em 13 out. 2017.

INVENSENSE. ***MPU-6050 Datasheet***. 2017. <https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/ >. Acesso em: 03 out. 2017.

IRAZABAL, J.-M.; BLOZIS, S. ***Application Note - I2C Manual***. 2003. <https://www.nxp.com/docs/en/application-note/AN10216.pdf>. Acesso em: 25 out. 2017.

KALINSKY, D.; KALINSKY, R. ***Introduction to I2C***. 2001. <https://www.embedded.com/electronics-blogs/beginner-s-corner/4023816/Introduction-to-I2C>. Acesso em: 25 out. 2017.

KEIM, R. ***Back to Basics: The Universal Asynchronous Receiver/Transmitter (UART)***. 2016. <https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/>. Acesso em: 27 out. 2017.

KOCH, C.; BRILAKIS, I. **Pothole detection in asphalt pavement images**. *Advanced Engineering Informatics*, v. 25, n. 3, p. 507 – 515, 2011. Special Section: Engineering informatics in port operations and logistics.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. ***Introduction to Information Retrieval***. [S.l.: s.n.], 2009. <https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>. Acesso em 04 mai. 2018.

MEDNIS, A. et al. **Real time pothole detection using Android smartphones with accelerometers**. p. 1–6, June 2011.

METRO. **Brasil é o quinto país do mundo em mortes no trânsito, segundo OMS**. 2017. Acesso em 10 out. 2017.

NGA. ***Current GPS Satellite Data***. 2016. <http://earth-info.nga.mil/GandG/sathtml/satinfo.html>. Acesso em 03 out. 2017.

NISTICO, A. ***Working principle of a capacitive accelerometer***. 2013. <http://engineering-sciences.uniroma2.it/MENU/DOWNLOAD/TESI/2013/2013_tesi%20NISTICO%20Andrea.pdf>. Acesso em 02 out. 2017.

ÖZGÜR, A.; ÖZGÜR, L.; GÜNGÖR, T. **Text Categorization with Class-based and Corpus-based Keyword Selection**. In: *Proceedings of the 20th International Conference on Computer and Information Sciences*. Berlin, Heidelberg: Springer-Verlag, 2005. (ISCIS'05), p. 606–615.

PEIXOTO, L. S.; SILVA, A. C. Q. da. ***Acidentes decorrentes de vias públicas urbanas danificadas: a responsabilidade civil do município***. 2011. <http://www.ambito-juridico.com.br/site/?n_link=revista_artigos_leitura&artigo_id=11884>. Acesso em 02 out. 2017.

RINDLER, W. **Essential Relativity: Special, General, and Cosmological**. Springer, p. 61, 2013.

ROUSE, M. ***USART (Universal Synchronous/Asynchronous Receiver/Transmitter)***. 2012. <http://whatis.techtarget.com/definition/USART-Universal-Synchronous-Asynchronous-Receiver-Transmitter>. Acesso em: 27 out. 2017.

SCIKIT-LEARN. ***RBF SVM parameters***. 2018. <http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html>. Acesso em 2 out. 2018.

SEMICONDUCTORS, N. ***I2C-bus specification and user manual***. 2014. <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. Acesso em: 25 out. 2017.

SIMCOM. ***SIM800L Hardware Design V1.0***. 2013. <http://simcomm2m.com/En/module/detail.aspx?id=138 >. Acesso em 20 out. 2018.

SINHA, S. et al. **Design and Simulation of MEMS Differential Capacitive Accelerometer**. *ISSS International Conference on Smart Materials, Structures and Systems*, 2014.

UBLOX. ***NEO-6 u-blox 6 GPS Modules Datasheet***. 2011. <https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf >. Acesso em: 03 out. 2017.

VITTORIO, A. et al. **Automated Sensing System for Monitoring of Road Surface Quality by Mobile Devices**. *Procedia - Social and Behavioral Sciences*, v. 111, p. 242 – 251, 2014. Acesso em 3 out. 2018.

WEINBERG, Z. ***SVM Separating Hyperplanes***. 2018. <https://en.wikipedia.org/wiki/File:Svm_separating_hyperplanes.png>. Acesso em 6 jul. 2018.

WOODFORD, C. ***Accelerometers***. 2017. <http://www.explainthatstuff.com/accelerometers.html>. Acesso em 02 out. 2017.

WORLD, G. ***What Exactly Is GPS NMEA Data?*** 2015. <http://gpsworld.com/what-exactly-is-gps-nmea-data/>. Acesso em 13 out. 2017.

APÊNDICE A – CÓDIGO DA CLASSE GPS

CÓDIGO A.1 – Classe GPS

```
1  from threading import Thread
2
3  class GPS(Thread):
4      def __init__(self, serial, lock):
5          Thread.__init__(self)
6          self.serial = serial
7          self.signal_validity = False
8          self.latitude = 0.
9          self.longitude = 0.
10         self.speed = 0.
11         self.lock = lock
12
13     @staticmethod
14     def convert_degrees(degrees, minutes):
15         return degrees + minutes/60.
16
17     def calculate_decimal(self, value, direction):
18         degrees = int(value/100)
19         minutes = float(value) - (100*degrees)
20         abs_degree = self.convert_degrees(degrees, minutes)
21         if direction == 'W' or direction == 'S':
22             abs_degree *= -1.
23         return abs_degree
24
25     def parse_gprmc(self, msg):
26         params = msg.split(',')
27
28         if params[2] == 'V':
29             self.signal_validity = False
30             return
31         else:
32             self.signal_validity = True
33
34         try:
35             lat_raw = float(params[3])
36             lng_raw = float(params[5])
37             speed_raw = float(params[7])
38
39             self.latitude = round(self.calculate_decimal(lat_raw, params[4]),
                       7)
40             self.longitude = round(self.calculate_decimal(lng_raw, params
                   [6]), 7)
41             self.speed = round(speed_raw*1.852, 2)
42             self.signal_validity = True
43         except ValueError:
44             self.signal_validity = False
45
46         return self.latitude, self.longitude, self.speed
47
```

```
48      def get_latitude(self):
49          return self.latitude
50
51      def get_longitude(self):
52          return self.longitude
53
54      def get_coordinates(self):
55          return self.latitude, self.longitude
56
57      def get_speed(self):
58          return self.speed
59
60      def is_valid(self):
61          return self.signal_validity
62
63      def run(self):
64          while True:
65              self.lock.acquire()
66              data = self.serial.readline()
67              if data.startswith("$GPRMC"):
68                  self.parse_gprmc(data)
69              self.lock.release()
```

APÊNDICE B – CÓDIGO DA CLASSE I2C

CÓDIGO B.1 – Classe I2CDevice

```
import smbus

class I2CDevice(object):
    def __init__(self, register_address, bus_line):
        self.register_address = register_address
        self.bus = smbus.SMBus(bus_line)

    @staticmethod
    def merge_bits(original, value, left_bit_start, length):
        left_shift = (left_bit_start - length + 1)
        mask = ((0x1 << length) - 1) << left_shift
        value <<= left_shift
        value &= mask
        original &= ~mask
        return original | value

    def write_bits(self, register, value, left_bit_start, length):
        original = self.read_byte(register)
        if original is None:
            return None
        new_value = self.merge_bits(original, value, left_bit_start, length)
        return self.write_byte(register, new_value)

    def write_word(self, register, value):
        value_h = value >> 8
        value_l = value & 0xFF
        self.write_byte(register, value_h)
        return self.write_byte(register+1, value_l)

    def write_byte(self, register, value):
        try:
            return self.bus.write_byte_data(self.register_address, register,
                value)
        except IOError:
            return None

    def read_byte(self, register):
        try:
            return self.bus.read_byte_data(self.register_address, register)
        except IOError:
            return None

    def read_word(self, register):
        try:
            high = self.bus.read_byte_data(self.register_address, register)
            low = self.bus.read_byte_data(self.register_address, register +
                1)
            val = (high << 8) + low
            return val
```

```
48          except IOError:
49              return None
50
51      def read_word_2c(self, register):
52          val = self.read_word(register)
53          if val >= 0x8000:
54              return -((65535 - val) + 1)
55          else:
56              return val
```

APÊNDICE C – CÓDIGO DA CLASSE MPU6050

CÓDIGO C.1 – Classe MPU6050

```
1  from I2CDevice import I2CDevice
2
3  class MPU6050(I2CDevice):
4      def __init__(self):
5          self.BUS_LINE = 1
6          self.MPU_ADDRESS = 0x68
7          self.PWR_MGMT_1 = 0x6b
8          self.ACCEL_CONFIG = 0x1c
9          self.ACCEL_SCALE_START_BIT = 4
10         self.ACCEL_SCALE_LENGTH = 2
11         self.ACC_X_OFFSET_H = 0x06
12         self.ACC_X_OFFSET_L = 0x07
13         self.ACC_Y_OFFSET_H = 0x08
14         self.ACC_Y_OFFSET_L = 0x09
15         self.ACC_Z_OFFSET_H = 0x0a
16         self.ACC_Z_OFFSET_L = 0x0b
17         self.ACCEL_XOUT_H = 0x3b
18         self.ACCEL_YOUT_H = 0x3d
19         self.ACCEL_ZOUT_H = 0x3f
20         self.GYRO_CONFIG = 0x1b
21         self.GYRO_SCALE_START_BIT = 4
22         self.GYRO_SCALE_LENGTH = 2
23         self.GYRO_XOUT_H = 0x43
24         self.GYRO_YOUT_H = 0x45
25         self.GYRO_ZOUT_H = 0x47
26         self.accel_scale = 0.0
27         self.gyro_scale = 0.0
28         super(MPU6050, self).__init__(self.MPU_ADDRESS, self.BUS_LINE)
29
30     def set_psm_off(self):
31         self.bus.write_byte_data(self.MPU_ADDRESS, self.PWR_MGMT_1, 0)
32
33     def set_accelerometer_scale(self, scale):
34         if scale == 2:
35             config_bits = 0b00
36         elif scale == 4:
37             config_bits = 0b01
38         elif scale == 8:
39             config_bits = 0b10
40         elif scale == 16:
41             config_bits = 0b11
42         else:
43             return -1
44
45         ret = self.write_bits(self.ACCEL_CONFIG, config_bits, self.
               ACCEL_SCALE_START_BIT,
46                               self.ACCEL_SCALE_LENGTH)
47         self.accel_scale = scale
48         return ret
```

```
49
50      def set_accelerometer_x_offset(self, offset):
51          return self.write_word(self.ACC_X_OFFSET_H, offset)
52
53      def set_accelerometer_y_offset(self, offset):
54          return self.write_word(self.ACC_Y_OFFSET_H, offset)
55
56      def set_accelerometer_z_offset(self, offset):
57          return self.write_word(self.ACC_Z_OFFSET_H, offset)
58
59      def get_accelerometer_x(self):
60          return round(self.read_word_2c(self.ACCEL_XOUT_H)*self.accel_scale
                /32767., 5)
61
62      def get_accelerometer_y(self):
63          return round(self.read_word_2c(self.ACCEL_YOUT_H)*self.accel_scale
                /32767., 5)
64
65      def get_accelerometer_z(self):
66          return round(self.read_word_2c(self.ACCEL_ZOUT_H)*self.accel_scale
                /32767., 5)
67
68      def set_gyro_scale(self, scale):
69          if scale == 250:
70              config_bits = 0b00
71          elif scale == 500:
72              config_bits = 0b01
73          elif scale == 1000:
74              config_bits = 0b10
75          elif scale == 2000:
76              config_bits = 0b11
77          else:
78              return -1
79
80          ret = self.write_bits(self.GYRO_CONFIG, config_bits, self.
                GYRO_SCALE_START_BIT,
81                                self.GYRO_SCALE_LENGTH)
82          self.gyro_scale = scale
83          return ret
84
85      def get_gyro_x(self):
86          return round(self.read_word_2c(self.GYRO_XOUT_H)*self.gyro_scale
                /32767., 5)
87
88      def get_gyro_y(self):
89          return round(self.read_word_2c(self.GYRO_YOUT_H)*self.gyro_scale
                /32767., 5)
90
91      def get_gyro_z(self):
92          return round(self.read_word_2c(self.GYRO_ZOUT_H)*self.gyro_scale
                /32767., 5)
```

APÊNDICE D – CLASSE PARA COMUNICAÇÃO GSM

```
from GsmException import GsmException
from GsmHttpResponse import GsmHttpResponse

class GsmDevice(object):

    def __init__(self, serial):
        self.serial = serial

    def send_http(self, httpConnection, apnConfiguration):

        self.send_and_check('AT+CFUN=1')
        self.send_and_check('AT+CGATT=1')
        self.send_and_check('AT+CIPSHUT')
        self.send_and_check('AT+CIPMUX=0')
        self.send_and_check('AT+CSTT=' + apnConfiguration.get_apn() + ',' +
                                    apnConfiguration.get_user() + ',' +
                                    apnConfiguration.get_password())
        self.send_and_check('AT+CIICR')
        self.send_and_check('AT+CIFSR')
        self.send_and_check_cipstart('TCP', httpConnection.get_host(),
                                        httpConnection.get_port())

        request = httpConnection.build()

        self.send_and_read('AT+CIPSEND=' + str(len(request)))
        self.send(request + '\x1A')
        raw_response = self.read_until_closed()

        response = GsmHttpResponse(raw_response)
        return response.get_return_code(), response.get_response()

    def send_and_check(self, at_command):
        self.serial.write(at_command + '\r\n')
        self.serial.readline()
        if self.serial.readline() == 'ERROR\r\n':
            raise GsmException('Exception in command ' + at_command)

    def send_and_check_cipstart(self, protocol, host, port):
        self.serial.write('AT+CIPSTART="' + protocol +
                          '","' + host + '",' + str(port) + '\r\n')
        self.serial.readline()
        self.serial.readline()
        self.serial.readline()
        status = self.serial.readline()
        if status != 'CONNECT OK\r\n' and status != 'ALREADY CONNECT\r\n':
            raise GsmException('Exception in command AT+CIPSTART')

    def send_and_read(self, at_command):
        self.serial.write(at_command + '\r\n')
        self.serial.readline()

```

```
52      def send(self, at_command):
53          self.serial.write(at_command + '\r\n')
54
55      def read_until_closed(self):
56          response = ''
57          self.serial.readline()
58          while True:
59              line = self.serial.readline()
60              if line != 'CLOSED\r\n':
61                  response += line
62              else:
63                  return response
```

APÊNDICE E – CÓDIGO PARA TREINAMENTO E AVALIAÇÃO DA SVM

```
1  import sys, csv, pickle, numpy as np
2  from sklearn import svm
3  from sklearn.metrics import f1_score
4  from sklearn.metrics import confusion_matrix
5  from sklearn.preprocessing import MinMaxScaler
6
7  training_data = []
8  training_classes = []
9  test_data = []
10 test_classes = []
11
12 path = '../../data/'
13 filename = path + 'processed_data'
14 out_filename = path + 'results/'
15
16 if '-t' in sys.argv or '--two' in sys.argv:
17     two_classes = True
18 else:
19     two_classes = False
20
21 if '-a' in sys.argv or '--axis-independent' in sys.argv:
22     axis_indep = True
23 else:
24     axis_indep = False
25
26 c_parameter = 1
27 if '-c' in sys.argv:
28     ind = sys.argv.index('-c')
29     c_parameter = float(sys.argv[ind+1])
30
31 gamma_parameter = 1
32 if '-g' in sys.argv:
33     ind = sys.argv.index('-g')
34     gamma_parameter = float(sys.argv[ind+1])
35
36 kernel_type = 'linear'
37 if '-k' in sys.argv:
38     ind = sys.argv.index('-k')
39     kernel_type = sys.argv[ind+1]
40
41 sweep_run = False
42 if '-s' in sys.argv:
43     sweep_run = True
44
45 out_filename += kernel_type
46
47 if not sweep_run:
48     out_filename += '_c_' + str(c_parameter)
49     out_filename += '_g_' + str(gamma_parameter)
50
51 if two_classes:
```

```
52      filename += '_2'
53      out_filename += '_2'
54  if axis_indep:
55      filename += '_axis_indep'
56      out_filename += '_axis_indep'
57
58  filename += '.csv'
59  out_filename += '.csv'
60
61  with open(filename, 'r') as p_data:
62      csv_reader = csv.reader(p_data)
63      to_training_data = True
64      first_row = True
65
66      for row in csv_reader:
67          if not first_row:
68              r = np.array(row, dtype=float)
69
70              if to_training_data:
71                  training_classes.append(int(r[0]))
72                  if axis_indep:
73                      training_data.append([r[1],r[2],r[3],r[4],r[5],r[6]])
74                  else:
75                      training_data.append([r[1],r[2],r[3],r[4],r[5],r[6],r[7],
76                                            r[8],r[9],r[10],r[11],r[12],r[13],r
                                                  [14]])
77                  to_training_data = False
78              else:
79                  test_classes.append(int(r[0]))
80                  if axis_indep:
81                      test_data.append([r[1],r[2],r[3],r[4],r[5],r[6]])
82                  else:
83                      test_data.append([r[1],r[2],r[3],r[4],r[5],r[6],r[7],
84                                        r[8],r[9],r[10],r[11],r[12],r[13],r[14]])
85                  to_training_data = True
86          first_row = False
87
88  scaler = MinMaxScaler()
89  training_normalized = scaler.fit_transform(training_data)
90  test_normalized = scaler.transform(test_data)
91
92  if sweep_run:
93      sweep_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
94      f1_matrix = np.empty((len(sweep_values), len(sweep_values)))
95
96      for i, gamma in enumerate(sweep_values):
97          for j, c in enumerate(sweep_values):
98              classifier = svm.SVC(gamma=gamma, C=c, kernel=kernel_type,
                    degree=2, cache_size=4000)
99              classifier.fit(training_normalized, training_classes)
100             predicted = classifier.predict(test_normalized)
101             f1_matrix[i][j] = str(f1_score(test_classes, predicted, average=
                    'macro'))
102
103     with open(out_filename, 'w') as out_file:
104         out_file.write('gamma\\C')
```

```
105         for sweep in sweep_values:
106             out_file.write(',' + str(sweep))
107         out_file.write('\n')
108 
109         for i, row in enumerate(f1_matrix):
110             out_file.write(str(sweep_values[i]))
111 
112             for cell in row:
113                 out_file.write(',' + str(cell))
114             out_file.write('\n')
115 
116 else:
117     classifier = svm.SVC(gamma=gamma_parameter, C=c_parameter, kernel=
        kernel_type, degree=2)
118     classifier.fit(training_normalized, training_classes)
119     predicted = classifier.predict(test_normalized)
120     np.set_printoptions(precision=2, suppress=True)
121     conf_matrix = np.transpose(confusion_matrix(test_classes, predicted))
122     f1 = str(f1_score(test_classes, predicted, average='macro'))
123 
124     with open(out_filename, 'w') as out_file:
125         out_file.write(str(conf_matrix) + '\n')
126         out_file.write(f1)
```

APÊNDICE F – CÓDIGO PARA AVALIAÇÃO EM TEMPO REAL

```
import sys, serial, pickle, json, numpy as np
from datetime import datetime
from MPUThread import MPUThread
from gpiozero import LED
from MeasurementProcessor import MeasurementProcessor
from GsmDevice import GsmDevice
from GsmHttpConnection import GsmHttpConnection
from GsmApnConfiguration import GsmApnConfiguration
from GsmException import GsmException
from gps import GPS
from threading import Lock

class Main(object):
    def __init__(self):
        self.status = 0
        self.mux = LED(4)
        self.mux.on()
        self.serial_lock = Lock()
        self.mpu = MPUThread()
        self.mpu.setDaemon(True)
        self.mpu.start()
        self.serial = serial.Serial('/dev/ttyS0', 9600)
        self.gsm = GsmDevice(self.serial)
        self.apn_config = GsmApnConfiguration("zap.vivo.com.br", "vivo", "
            vivo")
        self.gsm_http_config = GsmHttpConnection("monetovani.com", "", "
            roads_api.php")
        self.gsm_http_config.set_method('POST')
        self.gps = GPS(self.serial, self.serial_lock)
        self.gps.setDaemon(True)
        self.gps.start()
        self.good_led = LED(17)
        self.regular_led = LED(27)
        self.bad_led = LED(22)
        self.turn_on_all_leds()
        self.svm = pickle.load(open('../../data/model_3_qualities.sav', 'rb'
            ))
        self.measurements = MeasurementProcessor()
        self.detections_buffer = []

    def main(self):
        try:
            while True:

                while not self.measurements.is_buffer_full():
                    acc_x, acc_y, acc_z = self.mpu.getAccelerationValue()
                    gyr_x, gyr_y, gyr_z = self.mpu.getGyroscopeValue()
                    speed = self.gps.get_speed()
                    latitude, longitude = self.gps.get_coordinates()
                    gps_validity = self.gps.is_valid()

```

```
49                  measurement_unit = [acc_x,acc_y,acc_z,gyr_x,gyr_y,gyr_z,
50                                      speed,latitude,longitude,gps_validity]
51                  if not self.measurements.add_measurement(measurement_unit
                        ):
52                      self.turn_off_all_leds()
53
54              measurements_output = self.measurements.get_processed_output
                    ()
55              indicators = np.reshape(measurements_output[:14], (1, -1))
56
57              road_quality = self.svm.predict(indicators)[0]
58
59              if road_quality == 1:
60                  self.bad_led.on()
61                  self.regular_led.off()
62                  self.good_led.off()
63              elif road_quality == 2:
64                  self.bad_led.off()
65                  self.regular_led.on()
66                  self.good_led.off()
67              else:
68                  self.bad_led.off()
69                  self.regular_led.off()
70                  self.good_led.on()
71
72              speed = measurements_output[12]
73              start_lat = measurements_output[14]
74              start_lng = measurements_output[15]
75              end_lat = measurements_output[16]
76              end_lng = measurements_output[17]
77              detection_time = str(datetime.now())
78
79              self.serial_lock.acquire()
80              self.serial.reset_input_buffer()
81              self.mux.toggle()
82              self.send_detection_to_server([road_quality,speed,start_lat,
                    start_lng,
83                                             end_lat,end_lng,detection_time])
84              self.mux.toggle()
85              self.serial_lock.release()
86
87          except KeyboardInterrupt:
88              self.exit()
89              return
90
91      def build_request(self, detections):
92          obj_representation = []
93          for detection in detections:
94              obj_representation.append({'sensor_id': '1',
95                                         'quality': detection[0],
96                                         'speed': detection[1],
97                                         'start_latitude': detection[2],
98                                         'start_longitude': detection[3],
99                                         'end_latitude': detection[4],
100                                        'end_longitude': detection[5],
101                                        'course': '360',
```

```
102                                     'reading_date': detection[6]})
103
104
105         data_json = json.dumps(obj_representation)
106         return data_json
107
108     def send_detection_to_server(self, detection_body):
109         self.detections_buffer.append(detection_body)
110
111         body = self.build_request(self.detections_buffer)
112         self.gsm_http_config.set_body(body)
113
114         try:
115             code, body = self.gsm.send_http(self.gsm_http_config, self.
                    apn_config)
116             self.detections_buffer = []
117         except GsmException as e:
118             print(e.message)
119
120     def turn_off_all_leds(self):
121         self.bad_led.off()
122         self.regular_led.off()
123         self.good_led.off()
124
125     def turn_on_all_leds(self):
126         self.bad_led.on()
127         self.regular_led.on()
128         self.good_led.on()
129
130     def exit(self):
131         sys.exit()
132         return
133
134 if __name__ == "__main__":
135     MAIN = Main()
136     MAIN.main()
```

APÊNDICE G – CLASSE MEASUREMENTPROCESSOR

```
1  class MeasurementProcessor(object):
2      SPEED_LIMIT = 15
3      TOTAL_SAMPLING_TIME = 10
4
5      def __init__(self):
6          self.measurements = []
7          self.start_time = 0
8          self.end_time = 0
9          self.start_latitude = -1
10         self.start_longitude = -1
11         self.latest_latitude = -1
12         self.latest_longitude = -1
13
14     def add_measurement(self, single_measurement):
15         speed = single_measurement[6]
16         gps_signal_validity = single_measurement[9]
17         latitude = single_measurement[7]
18         longitude = single_measurement[8]
19         if speed < MeasurementProcessor.SPEED_LIMIT or not
               gps_signal_validity:
20             self.__reset_bufer()
21             return False
22         if not self.measurements:
23             self.start_time = time.time()
24             self.start_latitude = latitude
25             self.start_longitude = longitude
26         self.measurements.append(single_measurement[:9])
27         self.end_time = time.time()
28         self.latest_latitude = latitude
29         self.latest_longitude = longitude
30         return True
31
32     def is_buffer_full(self):
33         return (self.end_time - self.start_time) > MeasurementProcessor.
               TOTAL_SAMPLING_TIME
34
35     def get_processed_output(self):
36         np_array = np.array(self.measurements)
37         variance = np.var(np_array, axis=0)
38         mean = np.mean(np_array, axis=0)
39         self.__reset_bufer()
40         return [mean[0],variance[0],mean[1],variance[1],mean[2],variance[2],
41                 mean[3],variance[3],mean[4],variance[4],mean[5],variance[5],
42                 mean[6],variance[6],self.start_latitude,self.start_longitude,
43                 self.latest_latitude,self.latest_longitude]
44
45     def __reset_bufer(self):
46         self.measurements = []
47         self.start_time = 0
48         self.end_time = 0
```

APÊNDICE H – CÓDIGO PRINCIPAL DA API

```php
<?php
require_once('RoadManagerDB.php');
require_once('RoadsGoogleApi.php');
require_once('MissingFieldsException.php');

$method = $_SERVER['REQUEST_METHOD'];
$path = explode('/', trim($_SERVER['PATH_INFO'],'/'))[0];
$input = file_get_contents('php://input');
$input_json = json_decode($input, true);

if (empty($input_json) && $method === "POST") {
    http_response_code(400);
    exit();
} else if($method !== "GET" && $method !== "POST") {
    http_response_code(405);
    exit();
}

$roadManagerDb = new RoadManagerDB();

if ($method === "POST") {
    try {
        for ($i=0; $i<sizeof($input_json); $i++) {

            $id = $roadManagerDb->insertDetection($input_json[$i]);
            $roadManagerDb->insertDetectionPath($id, $input_json[$i]['
                start_latitude'], $input_json[$i]['start_longitude'],
                                                $input_json[$i]['
                                                    end_latitude'],
                                                    $input_json[$i]['
                                                    end_longitude']);
        }
        http_response_code(201);
        echo $id;
    } catch (MissingFieldsException $e) {
        http_response_code(400);
        echo $e->getMessage();
    }
} else if ($method === "GET") {
    if ($path === 'detections') {
        $output = $roadManagerDb->getDetections($_GET);
    } else if ($path === 'sensors') {
        $output = $roadManagerDb->getSensors();
    }

    $json = json_encode($output);
    header("Content-Type: application/json");
    echo $json;
}
```

APÊNDICE I – CLASSE PARA COMUNICAÇÃO COM O BANCO DE DADOS

```php
<?php

require_once('MissingFieldsException.php');
require_once('DbOperationException.php');
require_once('RoadsGoogleApi.php');
require_once('RoadsGoogleApiException.php');

class RoadManagerDB {

    public function __construct() {
        $this->db = new mysqli("localhost", "user", "password", "roadmanager");
    }

    public function getDetections(array $filters) {

        $num_params = "";
        $query = "SELECT detections.id, detections.sensor_id, detections.speed, detections.course,
                    detections.quality, detections.reading_date,
                    detections_path.latitude, detections_path.longitude FROM detections
                    INNER JOIN detections_path ON detections.id = detections_path.detection_id";
        $params = [];

        if (sizeof($filters) > 0) {
            $query .= ' WHERE ';
        }

        foreach($filters as $key => $value) {
            switch ($key) {
                case 'south':
                    $query .= 'AND latitude >= ? ';
                    break;
                case 'north':
                    $query .= 'AND latitude <= ? ';
                    break;
                case 'west':
                    $query .= 'AND longitude >= ? ';
                    break;
                case 'east':
                    $query .= 'AND longitude <= ? ';
                    break;
                case 'onlyBadRoads':
                    $query .= 'AND quality < ? ';
                    $value = '3';
                    break;
                case 'fromDate':
                    $query .= 'AND reading_date >= ? ';
                    break;
```

```
48                  default:
49                      $query .= " AND " . $key . "= ? ";
50                  break;
51              }
52              $params[] = $value;
53              $num_params .= "s";
54          }
55          $query .= 'LIMIT 1000;';
56
57          // Remove aditional AND right after the WHERE clause
58          $query = str_replace('WHERE AND', 'WHERE', $query);
59
60          $stmt = $this->db->prepare($query);
61          $stmt->bind_param($num_params, ...$params);
62
63          $stmt->execute();
64
65          if($result = $stmt->get_result()) {
66              return $this->formatDetections($result->fetch_all(MYSQLI_ASSOC));
67          }
68      }
69
70      public function insertDetection(array $data) {
71          $key_params = ['sensor_id', 'start_latitude', 'start_longitude', '
                end_latitude',
72                         'end_longitude', 'speed', 'course', 'quality', '
                            reading_date'];
73
74          $orderedArray = [];
75          for ($i=0; $i<sizeof($key_params); $i++) {
76              if (!array_key_exists($key_params[$i], $data)) {
77                  throw new MissingFieldsException('Missing mandatory field ' .
                        $keys_params[$i]);
78              }
79              $orderedArray[] = $data[$key_params[$i]];
80          }
81
82          $query = 'INSERT INTO detections
83                    (sensor_id, start_latitude, start_longitude, end_latitude,
84                     end_longitude, speed, course, quality, reading_date,
                        created_date)
85                    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, NOW());';
86
87          if ($stmt = $this->db->prepare($query)) {
88              $stmt->bind_param("sssssssss", ...array_values($orderedArray));
89              if(!$stmt->execute()) {
90                  throw new DbOperationException();
91              }
92              return $this->db->insert_id;
93          }
94      }
95
96      public function insertDetectionPath(int $detection_id, float $start_lat,
            float $start_lng, float $end_lat, float $end_lng) {
```

```
97          $roadsApi = new RoadsGoogleApi();
98
99          $snappedPoints = $roadsApi->snapToRoads($start_lat, $start_lng,
                $end_lat, $end_lng);
100
101         if (array_key_exists('error', $snappedPoints)) {
102             throw new RoadsGoogleApiException($snappedPoints['error']['
                    message']);
103         }
104
105         $snappedPoints = $snappedPoints['snappedPoints'];
106         $query = 'INSERT INTO detections_path (detection_id, path_counter,
                latitude, longitude) VALUES (?, ?, ?, ?);';
107
108         if ($stmt = $this->db->prepare($query)) {
109
110             for ($i=0; $i<count($snappedPoints); $i++) {
111                 $coordinates = $snappedPoints[$i]['location'];
112                 $path_count = $i + 1;
113                 $stmt->bind_param("ssss", $detection_id, $path_count,
                        $coordinates['latitude'], $coordinates['longitude']);
114
115                 if(!$stmt->execute()) {
116                     throw new DbOperationException();
117                 }
118             }
119         }
120
121         $stmt->close();
122         return true;
123     }
124
125     public function getSensors() {
126         $query = 'SELECT * from sensors;';
127         $stmt = $this->db->prepare($query);
128         $stmt->execute();
129
130         if($result = $stmt->get_result()) {
131             return $result->fetch_all(MYSQLI_ASSOC);
132         }
133     }
134
135     private function formatDetections(array $detections): array {
136         $formatted = [];
137
138         for($i=0; $i<count($detections); $i++) {
139             if ($i == 0) {
140                 $previousId = $detections[$i]['id'];
141                 $path = [];
142             }
143             else if ($previousId !== $detections[$i]['id']) {
144                 $detection = [];
145                 $detection['id'] = $detections[$i-1]['id'];
146                 $detection['sensor_id'] = $detections[$i-1]['sensor_id'];
147                 $detection['speed'] = $detections[$i-1]['speed'];
148                 $detection['course'] = $detections[$i-1]['course'];
```

```
149                 $detection['quality'] = $detections[$i-1]['quality'];
150                 $detection['reading_date'] = $detections[$i-1]['reading_date'
                        ];
151                 $detection['path'] = $path;
152                 $formatted[] = $detection;
153                 $path = [];
154                 $previousId = $detections[$i]['id'];
155             }
156 
157             $latLngPair = [];
158             $latLngPair['lat'] = $detections[$i]['latitude'];
159             $latLngPair['lng'] = $detections[$i]['longitude'];
160             $path[] = $latLngPair;
161         }
162         if ($i > 0) {
163             $i--;
164             $detection = [];
165             $detection['id'] = $detections[$i]['id'];
166             $detection['sensor_id'] = $detections[$i]['sensor_id'];
167             $detection['speed'] = $detections[$i]['speed'];
168             $detection['course'] = $detections[$i]['course'];
169             $detection['quality'] = $detections[$i]['quality'];
170             $detection['reading_date'] = $detections[$i]['reading_date'];
171             $detection['path'] = $path;
172             $formatted[] = $detection;
173         }
174 
175         return $formatted;
176     }
177 }
```