Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia Eletrônica

MONITORAMENTO DE SISTEMAS DE TRANSPORTE E DE RODOVIAS COM ARDUINO E SHIELD: GSM, GPS E GPRS

Autor: Derick Horrana de Souza da Trindade
Orientador: Marcus Vinicius Batistuta

Brasília, DF
2015

Derick Horrana de Souza da Trindade

MONITORAMENTO DE SISTEMAS DE TRANSPORTE E DE RODOVIAS COM ARDUINO E SHIELD: GSM, GPS, GPRS.

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Orientador: Dr. Marcus Vinicius Batistuta.

Brasília, DF
2015

CIP – Catalogação Internacional da Publicação*

Trindade, Derick Horrana de Souza.
Monitoramento de Sistemas de Transporte com Arduino e Shield: GSM, GPS e GPRS / Derick Horrana de Souza da Trindade. Brasília: UnB, 2014. 103 p.: il.; 29,5 cm.

Monografia Engenharia Eletrônica – Universidade de Brasília Faculdade do Gama, Brasília, 2014. Orientação: Marcus Vinicius Batistuta.

1. GPS. 2. Arduino. 3. GSM. 4. MPU-6050.
5. Manutenção. 6. Rodovias. 7. Monitoramento. 8. Veículo.
9. Rastreamento. 10. SHIELD.

CDU Classificação

MONITORAMENTO DE SISTEMAS DE TRANSPORTE COM ARDUINO E SHIELD: GSM, GPS E GPRS

Derick Horrana de Souza da Trindade

Monografia submetida como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica da Faculdade UnB Gama - FGA, da Universidade de Brasília, em 09 de dezembro de 2015 apresentada e aprovada pela banca examinadora abaixo assinada:

Professor (Titulação): Marcus Vinicius Batistuta, UnB/ FGA.
Orientador

Professor (Titulação): Fabiano Araújo Soares, UnB/ FGA.
Membro Convidado

Professor (Titulação): Cristiano Jacques Miosso, UnB/ FGA.
Membro Convidado

Brasília, DF
2015

Esse trabalho é dedicado a toda minha família, pelo incentivo e apoio constante.

AGRADECIMENTOS

Quero agradecer primeiramente a Deus, pela força e sabedoria ao longo da minha vida universitária. Á minha família, por não medir esforços para eu chegar até esta fase da minha vida. Ao meu orientador, Professor Marcus Vinicius Batistuta, por ter incentivado e apoiado a minha ideia. E a todos aqueles que de alguma forma me ajudaram, os meus sinceros agradecimentos.

RESUMO

Este trabalho tem como propósito analisar a utilização do sistema de rastreamento e monitoramento de veículos dentro do perímetro urbano do Distrito Federal, com o foco aplicativo no levantamento de informações a respeito do trecho que vai da BR-020 até o final da EPIA Sul. A ideia principal será abordar a associação de dispositivos eletrônicos para a elaboração de um protótipo adequado para rastrear a localização do veículo (utilizando o GPS), delimitar uma área de interesse (trecho BR-020 ⁄ EPIA Sul), ler os sinais dos sensores e transmitir mensagens (SMS) através de uma rede de telefonia celular (GSM). No trabalho também estará incluso uma solução, que envolve um sensor associado ao sistema embarcado, a ser instalada em veículos, que tenha a capacidade de identificar quando e onde o carro passou por um buraco. Ou seja, além de obter a posição georreferenciada, o dispositivo proposto irá monitorar em tempo real o estado de uma determinada rodovia. Ao longo dessa dissertação será descrita e mostrada a implementação do protótipo do sistema de monitoramento baseado em uma placa SHIELD GPS/GPRS/GSM V3.0 com a integração de um sensor MPU-6050, com giroscópio e acelerômetro acoplados. Essa placa, devidamente programada, será capaz de ler e interpretar os valores do sensor, transmitindo as informações coletas, por meio da rede GSM, a um computador portátil ou a um aparelho telefônico móvel. A posição instantânea do veículo será enviada junto com possíveis variações de aceleração e de direção em uma rodovia. A perturbação registrada será localizada por GPS, permitindo ao usuário identificar trechos da rodovia com buracos. As informações serão identificadas através do dispositivo e serão transmitidas e demonstradas em um notebook à medida que o veículo se movimenta. Este protótipo será desenvolvido de modo a permitir a instalação em qualquer veículo que necessite de um sistema de medição autônomo, com acelerômetro no eixo Z e um dispositivo GPS, para diagnóstico remoto. A intenção real será, portanto, adequar a implementação do trabalho, tanto para que exista um histórico de dados precisos que auxiliem as empresas e os órgãos públicos no monitoramento da frota de veículos quanto na manutenção de vias, afim de reduzir acidentes de trânsito ou quaisquer outros prejuízos provenientes das condições precárias das rodovias do Distrito Federal.

Palavras-chave: GPS. Arduino. GSM. MPU-6050. Manutenção. Rodovias. Monitoramento. Veículo. Rastreamento. SHIELD**.**

.

ABSTRACT

This term paper has the purpose of analyze the tracking system usage and monitoring vehicles within the urban perimeter of Brasília-DF, focusing on survey of informations about the stretch from the BR- 020 to the end of EPIA South. The main idea will be approach the association of electronic devices for the preparation of an appropriate prototype to track the vehicle's location (using GPS), delimits an area of interest (BR- 020 / EPIA southern stretch), reads the sensor signals and transmits messages (SMS) via a mobile network (GSM). In the term paper will also be included a solution that involves a sensor associated with the embedded system to be installed in vehicles, which has the ability to identify when and where the car passed over a hole. It means that the proposed device obtains the geo-referenced position and monitors in real time the status of a particular highway. Along this essay will be described and shown the implementation of the monitoring system prototype based on a board SHIELD GPS / GPRS / GSM V3.0 with the integration of a MPU- 6050 sensor, with coupled gyroscope and accelerometer. This board, properly programmed, will be able to read and interpret the sensor values, transmitting the collected information via GSM network, a laptop or a mobile phone. The instantaneous position of the vehicle will be sent with possible variations in acceleration and direction on a highway. The recorded disturbance will be located via GPS, allowing the user identify regions with holes. The information will be highlighted on the device as the vehicle moves. This prototype will be developed to permit installation in any vehicle that needs other remote diagnostic possibilities. The real intention will be the implementation of the work, to have an accurate historical data that help businesses and government agencies in monitoring the fleet of vehicles, as the maintenance of roads in order to reduce traffic accidents or any other damages from the poor conditions of the roads of the Brasília-DF.

Keywords: GPS.Arduino.GSM.MPU-6050. Maintenance. Highways. Monitoring. Vehicle. Tracking. SHIELD

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

BR	Rodovia Federal
BSC	*Base Station Controller*
BSS	*Base Station Subsystem*
BTS	*Base Transceiver Station*
DETRAN	Departamento de Trânsito
DF	Distrito Federal
DMP	*Digital Motion Processor*
EPIA	Estrada Parque Indústria e Abastecimento
GPRS	*General Packet Radio Services*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
HLR	*Home Location Registrer*
IRI	Índice de Rugosidade Internacional
LA	Área Local
LAC	Código de Área Local
MEMS	Sistema Microeletromecânico
MS	Estação Móvel
MSC	Centro de Comutação Móvel
NMEA	*National Marine Electronics Association*
PDN	Protocolo de Dados Padrão
RTPC	Rede de Telefonia Pública Comutada
SAA	Sistema de Operação e Suporte
SIM	*Subscriber Identity Module*
SMD	*Surface Mount Device*
SMS	*Short Message Service*
SS	Sistema de Comutação
USB	*Universal Serial Bus*
VLR	Registro Local de Visita
ITS	Sistema de Transporte Inteligente
FIFO	*First In First On*
I2C	*Inter-Intergrated Circuit*
3D	Três Dimensões
IDE	*Integrated Development Environment*
UART	*Universal Asynchronous Receiver/Transmitter*

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

O desenvolvimento da microeletrônica vem proporcionando soluções tecnológicas descomplicadas nos diversos ramos da ciência, onde a miniaturização dos componentes eletrônicos promove a evolução de novas tecnologias capazes de encontrar uma saída prática e versátil dos problemas mais simples aos mais complexos. Os grandes benefícios da redução da escala dos circuitos, como microcontroladores, são a aplicação de projetos que acompanham de perto o usuário.

Segundo HEUBERGER [1] a microeletrônica é a mais importante condutora de inovações em vários setores industriais. Graças aos avanços nas produções modernas, nas tecnologias altamente complexas e nos sistemas inteligentes podem ser construídos a um baixo custo. Em conjunto com a tendência mundial no sentido do desenvolvimento de novos produtos eletrônicos tais como dispositivos móveis inteligentes e sistemas inteligentes autônomos, está o aumento da penetração de quase todos os sistemas técnicos com eletrônica. Ainda assim, produtos microeletrônicos tem o comprometimento de apresentar a funcionalidade rica e de alto desempenho, bem como de alta qualidade, segurança e confiabilidade, a fim de serem competitivos no mercado.

Próximo a essa percepção os meios transportes trouxeram um ambiente de duas faces para a evolução tecnológica. Na primeira têm-se os benefícios de acessibilidade a vários lugares e de deslocar-se de um ponto a outro em tempo cada vez mais reduzido. Porém têm-se também os impactos negativos resultante do crescimento da rede de transporte, em que suas malhas cada vez mais necessitam de acompanhamento de manutenção em tempo real, constante e, também, seus usuários necessitam de respostas cada vez mais rápidas.

Desta maneira, segundo SINGH e GUPTA [2], o gerenciamento do crescente tráfego tem se tornado um grande problema em todo o mundo. E é nesse contexto em que o Sistema de Transporte Inteligente (ITS), com a ajuda das novas tecnologias, procura fornecer a solução para tais problemas. De acordo com SINGH

e GUPTA[2], ITS é um sistema integrado que programa uma ampla gama de comunicação, controle, detecção de veículos e tecnologias eletrônicas para resolver e gerenciar os problemas de deslocamento e trajeto de veículos.

Consequentemente, antes do levantamento da problemática do crescimento do sistema de transporte e suas possíveis soluções, deve-se observar uma forma preventiva que busque evitar ou acabar com questões relacionadas a congestionamentos, má qualidade rodoviária, gestão de fluxo do transporte público e informações sobre o posicionamento veicular. Neste contexto o monitoramento cria a possibilidade de prevenir e de findar com esses quesitos negativos.

O presente trabalho busca conciliar a coleta de dados do monitoramento remoto de um veículo e suas possibilidades com as necessidades do mapeamento de uma via, resultando em um levantamento minucioso de um ponto ou de um trecho de possível atenção, por meio um protótipo automático de monitorização capaz de acompanhar os meios de transporte rodoviários. A alternativa encontrada baseada em um sistema de GPS/GSM/GPRS para pesquisa foi à facilidade de enumerar as informações sobre o veículo e a via em estudo, observando as variações da rodovia por intermédio dos sensores acoplados a placa Arduino.

O objeto de estudo será o perímetro urbano das rodovias do Distrito Federal, onde houve o aumento de 103,6%, no período de 2000 a 2014, do número de veículos, de acordo com o levantamento do Departamento de Trânsito (DETRAN) [3]. Será utilizado como fonte de pesquisa a BR-020/EPIA/BR-040 que passa pelos principais pontos de acesso ao planto piloto, e que consequentemente sofrem os desgastes gerados pelo crescimento do número de veículos, e também, de um automóvel particular para os testes. Com isso, será feita a coleta, o processamento e o tratamento dos dados dos principais pontos onde a variação da aceleração for mais brusca(Figuras 39,41,43,45 e 47).

1.2 PROBLEMATIZAÇÃO E PROPOSTA

O aproveitamento do monitoramento de rodovias e da automação veicular tem se tornado cada vez mais essencial nas medidas de segurança para rastreamento de veículos. Em comunhão a essas tecnologias surge a possibilidade de utilizar um equipamento capaz de capturar informações do meio pelo qual se

transita, objetivando identificar variações nos padrões da via. Um exemplo prático da aplicação do projeto em estudo é o dimensionamento do comportamento dos pavimentos ao longo de um período de utilização constante. Uma vez que o tráfego de veículos é um dos atores predominantes no processo de desgaste e transtorno em rodovias.

De acordo com NOPIAH [4], uma das principais fontes de vibração na cabine do carro é a vibração devido à interação entre o pneu e a superfície da estrada. Basicamente, o nível de vibração é dependente da velocidade do carro, do padrão do pneu e principalmente da rugosidade da superfície da estrada. Geralmente o ruído que é gerado pela vibração no sistema de amortecimento do veículo, acoplado ao seu interior, afeta as emoções do condutor e diminui o nível de foco durante a condução. Este ruído pode também ser descrito como uma fonte de aborrecimento para os seres humanos, onde a vibração indesejada pode interferir na comunicação entre os passageiros, afetar a boa condução e contribuir para a perturbação da atenção do motorista. A vibração no interior do veículo reduz o conforto, que influencia diretamente o foco da condução e pode levar a um acidente fatal. Exposição à vibração também pode causar movimento relativo entre o objeto visualizado e a retina, o que resulta numa imagem desfocada. Esta condição visual também prejudica o desempenho do condutor.

Sendo assim, o principal problema a ser resolvido pelo presente trabalho será aplicar um conjunto de métodos e ferramentas para medir, analisar e descrever o comportamento do veículo, tanto na identificação da sua posição quanto na verificação do nível de vibração do sistema veículo com interior do veículo. Para isso será necessário compreender a tendência e característica da rugosidade na rodovia para, assim, através deste estudo, avaliar o nível de vibração para uma definição de métrica de qualidade da estrada.

Dessa forma será efetuado o controle de um determinado sistema de transporte, verificando sua posição e identificando as deficiências nas rodovias do perímetro urbano do DF, procurando contribuir com a redução dos custos de transportes e com gastos de reparações quando as deficiências nas rodovias se agravam.

1.3 OBJETIVOS

1.3.1 Objetivos Gerais

Compreender e elaborar um sistema eletrônico, que possibilitará a obtenção de informações dos meios de transporte terrestres e das malhas rodoviárias do perímetro urbano do Distrito Federal, é a principal preocupação deste trabalho. Esse sistema se destinará a capturar dados como coordenadas geográficas e sinais de rugosidade da estrada, sejam eles tanto de aceleração positiva quanto de aceleração negativa. Os respectivos dados serão armazenados no protótipo, onde serão analisados, processados, tratados e posteriormente enviados por meio de um canal com infraestrutura sem fio (GSM/GPRS) para um computador ou um dispositivo celular. Isso viabilizará a otimização do monitoramento em tempo real não só do veículo como também o controle das condições das vias urbanas de Brasília, a fim de avaliar as mudanças na condição das rodovias para prever as necessidades de investimento e melhoria.

1.3.2 Objetivos Específicos

O primeiro objetivo será a montagem da placa Arduino/Shield que será acompanhada de duas antenas, sendo uma GPS e a outra GSM. Na placa estará incluso um conector para encaixar fones de ouvido, dois sensores, sendo um acelerômetro de dois eixos e um giroscópio de dois eixos, e um display.

O segundo objetivo será concluído com o desenvolvimento do programa que será responsável por fazer a interação do telefone móvel com o dispositivo proposto por meio de mensagem de texto.

O terceiro objetivo será a embarcação do programa na placa e possíveis ajustes operacionais, caso seja necessário.

O quarto objetivo será efetivar os testes do dispositivo em um veículo do decorrer da via começando na BR-020, continuando na via EPIA e findando na BR-040. Esse trecho servirá de amostra para a obtenção de dados e formação de tabelas e gráficos. O material coletado será relacionado às casualidades que ocorrerem com o veículo durante o percurso.

Por fim, o quinto objetivo será encerrado com a avalição das informações captadas nos testes e consequentemente a confecção do relatório de validação do projeto.

1.4 JUSTIFICATIVA

O acompanhamento ininterrupto de uma determinada rodovia por meio de um sistema que se desloca junto a um automóvel traz flexibilidade para o levantamento de um diagnóstico atualizado do próprio meio de transporte e da via em análise.

Informações sobre as condições da superfície das estradas são muito úteis para os usuários da estrada, porque com a disponibilidade de tais informações, os condutores podem evitar ou ser cautelosos com o mau caminho pela frente. Além disso, para as autoridades rodoviárias, as informações são muito importantes, pois elas podem ser aplicadas em processos de tomada de decisão, especialmente para o planejamento estratégico como o planejamento de gestão de ativos, planejamento de manutenção e programação [5].

Na aplicação do projeto notou-se que a frota de veículos aumenta o processo natural de desgaste das malhas rodoviárias nos perímetros urbanos dos grandes centros, exigindo a avalição, a manutenção assim como a reparação dos danos provocada por esse fenômeno. Pensando nisso propõem-se um trabalho de supervisão permanente que colabore com a identificação de áreas mais propicias à degradação. E com isso reduza tempo e custos de monitoramento físico, realizado por agentes dos órgãos responsáveis.

Vale ressaltar que o projeto pode ser utilizado em veículos de serviço público, bem como em particulares de forma versátil, com a inclusão de sensores de acordo com as necessidades do cliente, e transmita remotamente relatórios instantâneos de alterações ou armazene em sua memória estes para diagnósticos futuros. O desenvolvimento desse trabalho propunha-se a criar um sinal de aviso, para quando ocorra uma perturbação que fuja dos padrões em análise, exemplo uma depressão que caracterize um buraco, a informação seja transmitida automaticamente a uma central ou ao usuário final.

A coleta dos parâmetros do trabalho em estudo fornecerá dados precisos que beneficiarão os órgãos públicos, na conformação de vias, que visem reduzir eventualidades.

Nesse aspecto a rugosidade é uma importante característica identificada porque afeta não só uma condução de qualidade, mas também os custos relacionados com o atraso, o consumo de combustível e custos de manutenção. Esse papel se concentra principalmente na rugosidade da estrada como condição da superfície da estrada. Observa-se que a manutenção preventiva tanto quanto a manutenção corretiva são essenciais na redução também das causas de acidentalidades [5].

Segundo DHAR [6] dispondo de recursos limitados, o órgão responsável vai saber exatamente onde investir e tomar preferência de ação antes de as estradas ficarem gravemente danificadas. Também os dados históricos gerados por este sistema serão inestimáveis para os planejadores e pesquisas que podem, então, construir melhores estradas e redes rodoviárias.

Com isso, o projeto irá corroborar com a criação de um programa de manutenção permanente e adequação de vias. Ao mesmo tempo facilitará o estabelecimento do modelo padrão de coleta de informações sobre as ocorrências de acidentes de trânsito e a confecção das estatísticas do trânsito.

1.5 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está estruturado em quatro capítulos, sendo que este capítulo apresenta caráter introdutório.

O capítulo dois discute a fundamentação teórica, em que foi realizado o estudo detalhado das seguintes etapas: rastreamento de veículos; rastreamento via celular; monitoramento de estradas e rodovias.

No capítulo três são descritos os procedimentos necessários para a elaboração do projeto proposto. Contém, também, a contextualização a respeito de como será a apresentação e análise dos dados que serão obtidos. Consiste em recomendações para desenvolvimento póstero.

Por fim, em anexos estarão os programas utilizados como base para os testes de execução do protótipo. Inclusos também os *Datasheets* do Arduino Uno, SHIELD V3.0 e MPU-6050.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 RASTREAMENTO DE VEÍCULOS

Rastrear e monitorar meios de transporte vem sendo, nos últimos anos, motivo de interesse e estudo. A explicação para a atenção em tais temas consiste no desenvolvimento e avanço de sistemas eletrônicos e computacionais de baixo custo no mercado, o que proporciona consequentemente a germinação de novas pesquisas sobre o supervisionamento de veículos [7]. Com isso, a capacidade de detectar com precisão a localização do veículo e seu status é o principal objetivo de sistemas de monitoramento de trajetória de automóveis.

Estes sistemas são implementados usando várias técnicas híbridas que incluem: comunicação sem fio, posicionamento geográfico e aplicações embarcadas [8].

Figura 1: Funcionamento do Sistema de Comunicação Módulo Rastreador/Central. Fonte: [40]

2.1.1 Tecnologia GPS

O Sistema de Posicionamento Global, também conhecido como *NavStar*, foi construído pelos militares norte-americanos com o objetivo de localizar posições em qualquer lugar da Terra com uma grande precisão [9].

Essencialmente, o GPS é um sistema espacial de posicionamento por rádio que fornece a hora ,a posição tridimensional e a informação de velocidade para os usuários devidamente equipados em qualquer lugar perto ou na superfície da terra

[9] [10]. Constitui-se basicamente em satélites, receptores e estações de monitoramento e controle [10].

Figura 2: Satélite GPS. Fonte: [41]

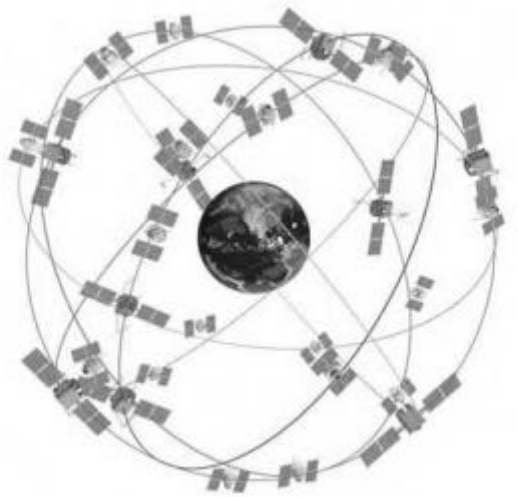

Figura 3: Constelação de Satélites GPS. Fonte: [4]

Seu funcionamento, embora construído com uma tecnologia complexa, possui um conceito bem simples e ocorre da seguinte forma: o receptor GPS recebe um sinal a partir de cada satélite GPS, onde esses mesmos satélites transmitem a hora exata em que os seus sinais foram enviados [7] [10]. A partir da diferença de tempo entre o instante em que o sinal foi transmitido e o instante em que foi recebido, o receptor pode estabelecer a distância em que ele está de cada satélite, e do mesmo modo, a posição certa em que os satélites estão no céu [10]. Dessa forma, obtidos os tempos de deslocamento dos sinais dos três satélites GPS e a suas localizações

no céu, o receptor GPS pode determinar a sua posição em três dimensões: leste, norte e altitude.

Com base nas distâncias calculadas é possível calcular a posição do receptor à superfície da terra com base num processo de triangulação. A distância ao satélite constitui o diâmetro duma esfera de centro na localização do satélite. Com base na intercepção de duas esferas obtém-se uma circunferência de pontos onde a posição do receptor é provável. Com base num terceiro satélite, ou seja, uma terceira esfera, apenas dois pontos da circunferência anterior constituem a possível posição do receptor [9] e [10].

Normalmente um destes pontos localiza-se a milhares de quilômetros da terra, o que permite a fácil resolução do problema.

Para determinar a localização dos satélites GPS, dois dados são necessários para o receptor GPS: o almanaque e a efeméride. Ambos são transmitidos continuamente pelos satélites GPS, e seu receptor GPS recebe e guarda esses dados. Basicamente o almanaque compreende informações sobre o estado, disponível ou indisponível, dos satélites. Já a efeméride é responsável por fornecer dados adicionais sobre a órbita de cada satélite, definindo assim a localização de cada satélite [9] e [11].

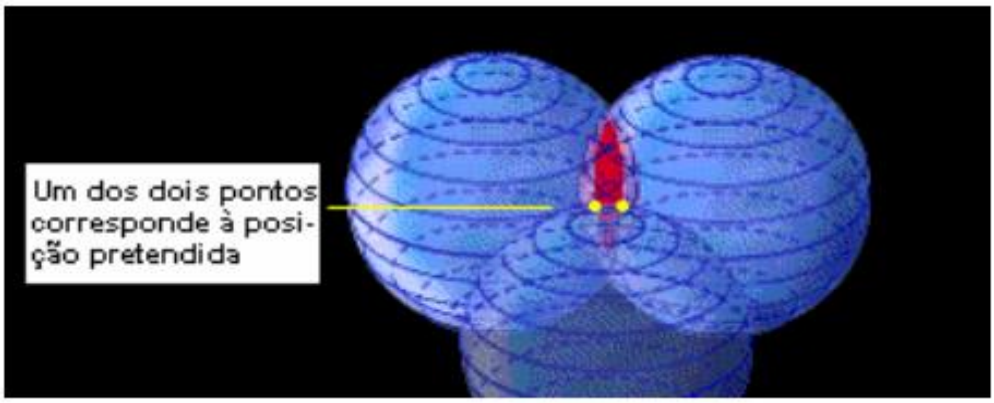

Figura 4: Definição do Ponto de Observação. Fonte: [9]

2.1.2 Aquisição de Dados

De modo geral receptores GPS enviam as informações obtidas utilizando sentenças definidas pelo padrão NMEA (*National Marine Electronics Association*). O NMEA consiste de sentenças, onde a primeira palavra, que caracteriza um tipo de

dados, define a interpretação do resto da frase. Cada tipo de dados tem a sua própria interpretação única e é definido no padrão NMEA. Algumas frases podem repetir as mesmas informações, mas isso não as independe de fornecerem novos dados [12].

Seja qual for o dispositivo ou programa que lê os dados, ele pode apenas observar a sentença de dados em que está interessado e simplesmente ignorar outras frases que não se preocupa [12]. O documento completo é vendido pela NMEA, porém partes significativas, como várias das sentenças, estão disponíveis na Internet e em manuais de receptores GPS. Alguns receptores também enviam outras sentenças, criadas pelos seus fabricantes, que são similares às sentenças do padrão [12]. As regras de formação das sentenças são apresentadas abaixo [11] [12]:

1 → As sentenças se iniciam com o caractere “$”;

2 → Os próximos caracteres identificam a fonte das sentenças, por exemplo, “GP” para GPS, “GL” para GLONASS e “PGRM” para sentenças proprietárias da empresa *Garmin*;

3 → Os próximos 3 caracteres, antes da primeira vírgula, identificam a sentença propriamente dita (ex.: “GGA” → *Fix Information*, “GSA” → *Overall Satellite Data*);

4 → Todos os campos seguintes, exceto o último, encontram-se delimitados por vírgulas;

5 → O último campo é terminado com um asterisco, que é imediatamente precedido por dois caracteres que representam um número hexadecimal de dois dígitos referente a uma soma de verificação (*Checksum*).

6 → As sentenças são terminadas por caracteres CR (*Carriage Return*) e LF (Line Feed).

Exemplo – Sentença GLL (*Geographic Latitude and Longitude*) $GPGLL,4916.45,N,12311.12,W,225444,A,*1D

“$” : Inicio da sentença.

“GP” : Fonte de sentença (receptor GPS).

“GLL” : Identificação da sentença.

“4916.45” : Latitude em graus (49) e minutos (16.45).

“N” : Latitude referente ao hemisfério norte .

“12311.12” : Longitude em graus (123) e minutos (11.12).

“W” : Longitude situada a oeste de Greenwich.

“225444” : Horário em que a posição foi obtida (22h54m44s - UTC).

“A” : Indica validade da posição (A → válido, V → inválido).

“*” : Indica o último campo.

“1D” : Soma de verificação (*Checksum*).

2.2 RASTREAMENTO VIA CELULAR

Rastreamento via celular é uma das aplicações em sistemas de telefonia com mais rápido e mais exigente crescimento no mercado. Desempenha atualmente um progresso contínuo entre todas as novas subscrições de telefone em volta do Mundo [13] [14]. Esse sistema de rastreamento utiliza quase que integralmente a tecnologia GSM/GPRS para a transmissão de informações. A tecnologia GSM (Sistema Global para Comunicações Móveis) é um serviço de comunicação pessoal conhecida por proporcionar conexão entre dispositivos móveis [14].

É governada por um padrão internacional desenvolvido pelo Instituto Europeu de Normas de Telecomunicações. A primeira especificação foi introduzida em 1990 e evoluiu desde então. O conjunto padrão inclui o tradicional circuito comutado móvel comunicação, bem como o padrão de comutação de pacote , como GPRS e EDGE. A especificação foi prorrogada a fim de apoiar os serviços de localização em 1999 [13].

2.2.1 Tecnologia GSM

GSM (*Global System for Mobile Communications*) é o modelo de tecnologia portátil mais conhecido entre telefones celulares. É também uma rede aberta e digital, utilizada para trocar dados de serviços de voz e móveis. O sistema compartilha elementos comuns com outras tecnologias utilizadas em telefones móveis, como a transmissão digital e a utilização de células. A rede GSM é basicamente dividida em três sistemas principais: o sistema de comutação (SS), o sistema de estação base (BSS), e o sistema de operação e de suporte (SAA) [14].

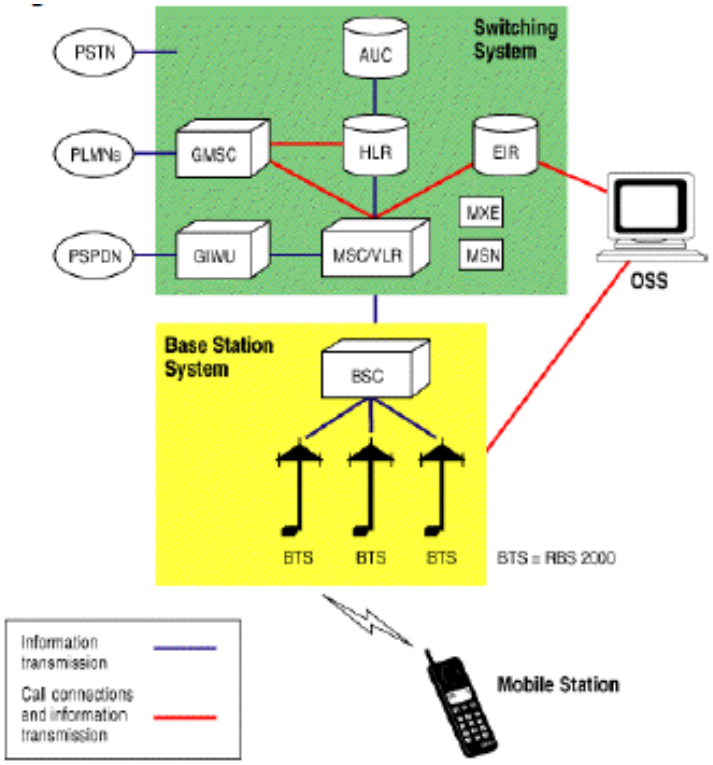

Figura 5: Elementos da Rede GSM. Fonte: [14].

2.2.2 As Operações e a Infraestrutura GSM

A área de cobertura de uma rede GSM é dividida em células, onde torres de transmissão conhecidas como Estações de Base (BTS) determinam a localização e o tamanho das células [13].

As Estações Base podem ser equipadas com múltiplas antenas apontando em direções diferentes. A área dos sinais transmitidos destas antenas direcionadas determinam setores dentro de cada célula onde cada setor tem um único

identificador, ID de chamada célula. Estas células são agrupadas em áreas lógicas conhecidos como Área Local (LA) [13] [14].

Uma LA tem um identificador inteiro chamado de Código de Área Local (LAC) e tem um controlador de estação base (BSC). A LA é um importante meio para controlar a mobilidade na rede, como Estações Móveis (MS). Essas estações são obrigadas a comunicar a sua nova posição quando se deslocam de uma LA para outra. Uma LA Nem grande nem pequena é ótima do ponto de vista da sinalização de rede, ao passo que seu tamanho é otimizado para minimizar o tráfego de sinalização [14].

O elemento menor da infraestrutura GSM é a MS (por exemplo, um telefone celular ou um Shield GSM) com o seu cartão SIM [14]. Essencialmente ele é responsável por medir o sinal circundante da força das BTSs / células. Além disso, a MS pode também se conectar a uma BTS [14] [15].

Já as vizinhas BTSs estão ligadas a uma Estação Base de Controle (BSC). Cada BSC geralmente pertence a uma LA, e elas estão conectados a um Centro de Comutação Móvel (MSC). Cada MSC tem um banco de dados chamado de Registro Local de Visitante (VLR) que registra o LAC da MS e telefones conectados dentro da área do MSC. Além disso, a rede tem uma base de dados global, que também contém a última LAC conhecida e identificadores de celulares das MS, chamada Registrador Local (HLR). MSC usa HLR quando o alvo MS está em outro MSC [14] [15].

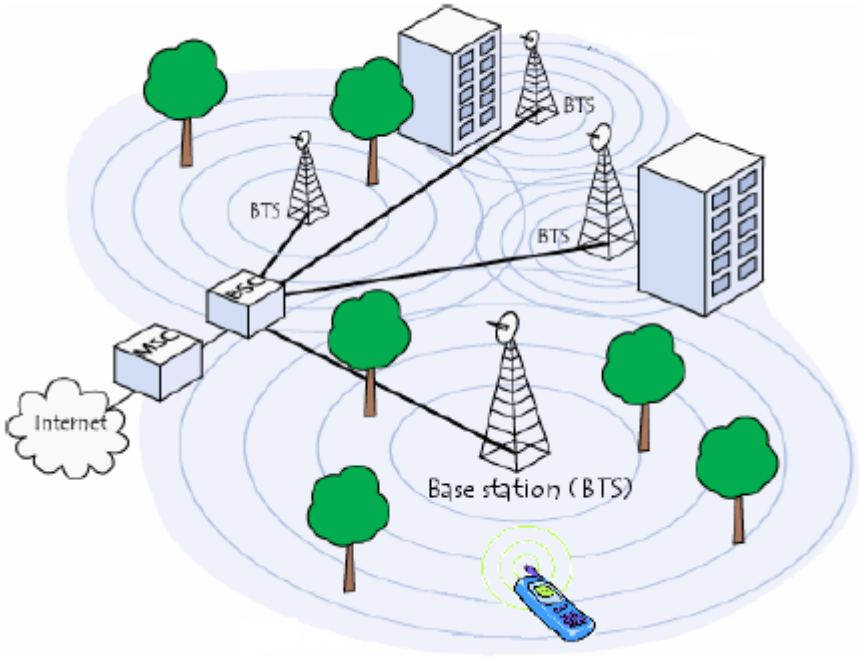

Figura 6: Rede GSM. Fonte: [15].

2.2.3 Tecnologia GPRS

O GPRS (*General Packet Radio Service*) foi elaborado para disponibilizar o tráfego de dados por pacotes na rede GSM, esse serviço trouxe o benefício à rede de telefonia celular de ser interligada a Internet. A tecnologia GPRS garantiu a comunicação GSM, por meio da técnica de comutação de pacotes para a rede celular. Nessa estruturação a mensagem transmitida ao sair é fragmentada em pacotes que recebem o endereço de destino em seu cabeçalho [30]. A comutação de pacotes usa a rede somente quando houver dados a serem enviados. Este serviço complementa o Serviço de Dados de Comutação de Circuitos e o Serviço de Mensagens Curtas. Com isso os pacotes quando na rede são direcionados pelo melhor caminho até chegarem ao seu destino, segundo a apresentação à ANATEL da Consulta Pública 198 sobre o uso das bandas de frequência para o fornecimento de Serviços de Comunicações Móveis Terrestres [16] [30].

A utilização do GPRS beneficiará a utilização do Protocolo Internet (IP) para a rede GSM. Este serviço une um grande número de redes de dados públicas e privadas, integrando protocolos de dados padrão (PDN), como o TCP/IP [16].

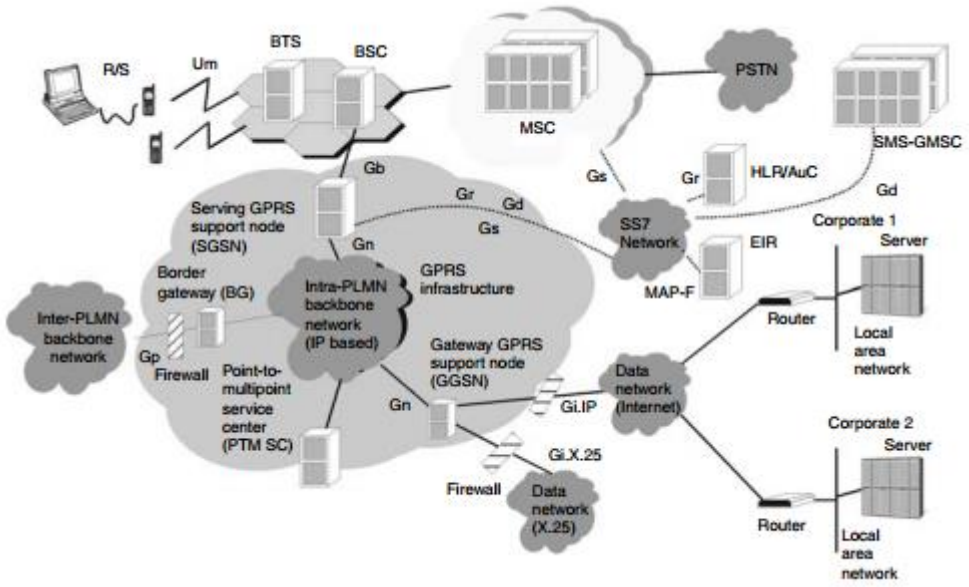

Figura 7: Visão Funcional do GPRS. Fonte: [30].

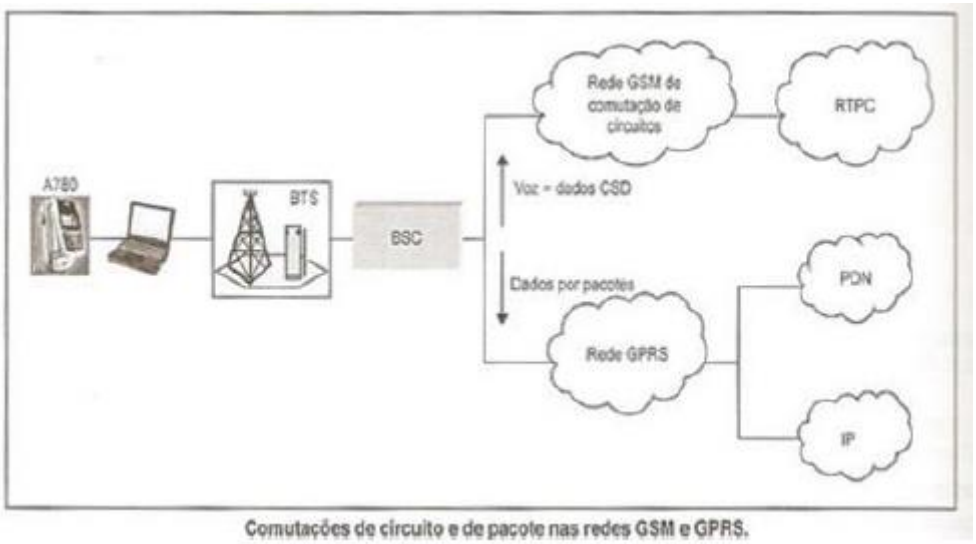

Figura 8: Tecnologia GPRS. Fonte: [16].

Na Figura 8 é observado como ocorre o processo de comutação da mensagem para dados de voz e dados por pacotes. A nuvem RTPC (Rede de Telefonia Pública Comutada) exemplifica a transmissão de voz, enquanto a nuvem IP (*Internet Protocol*) representa a ligação da mensagem com a Internet. O elemento BSC (Estação Base de Controle) é um nó de uma rede móvel celular que conecta a BTS (Estação de Base), é a interface rádio do terminal móvel composto de uma antena e um transceptor, todos componentes da arquitetura de uma rede móvel celular. Já a nuvem PDN é uma rede de circuitos de comutação de pacotes, que age

na transmissão de dados na forma digital à central de comutação de serviços móveis.

2.3 MONITORAMENTO DE ESTRADAS E RODOVIAS

A péssima conservação das rodovias e a quantidade de buracos são hoje duas realidades atuais do Distrito Federal. Os condutores são obrigados a ter bastante precaução quando vão dirigir seus veículos, em razão de determinadas vias terem mais buracos e fissuras do que passagens livres e alinhadas [17]. Esse cenário acaba prejudicando não só o condutor, mas também as empresas que prestam os serviços de manutenção, uma vez que as mesmas não realizam as reparações com frequências regulares e nem possuem um parâmetro de escolha acerca de quais locais as obras devem ser realizadas. O período de espera para reparação da via aumenta ainda mais a deterioração do asfalto, e isso acaba gerando prejuízos nos motoristas, com manutenção dos veículos, e nas empresas, com o aumento dos custos das obras [18].

Apesar do investimento feito com reparações, poucas pessoas ficam satisfeitas com a qualidade das estradas onde vivem ou trabalho. A razão é que estradas ruins danificam veículos, e, por muitas vezes, se tornam perigosas para motoristas e pedestres [17] [20].

Segundo [19] manter nossas estradas em boas condições é um problema desafiador porque são duras condições meteorológicas, a carga de tráfego apresenta-se de forma inesperada, além do desgaste normal degradam estradas, até mesmo bem- acabadas, em um período relativamente curto de tempo (semanas ou meses). Afinal os orçamentos municipais são geralmente restritos, e definir quais as estradas e quais tipos de concerto precisam se torna importante. Além disso, informar os condutores sobre as condições perigosas de uma estrada especialmente à noite ou quando a iluminação é fraca seria um recurso útil para sistemas de navegação. E este trabalho procura responder a esta necessidade.

Portanto, nesta parte do trabalho a ideia será monitorar o trecho da via entre a BR-020 e o final da EPIA Sul, dentro do Distrito Federal, usado um sistema embarcado composto inicialmente por: um Arduino Uno, um Shield GPS/GPRS/GSM

modulo V3.0 e um sensor MPU-6050 que contém em um único chip com um acelerômetro e um giroscópio tipo MEMS. A proposta será monitorar remotamente o estado da rodovia, apontando em que momento e em qual local o carro passou por um buraco.

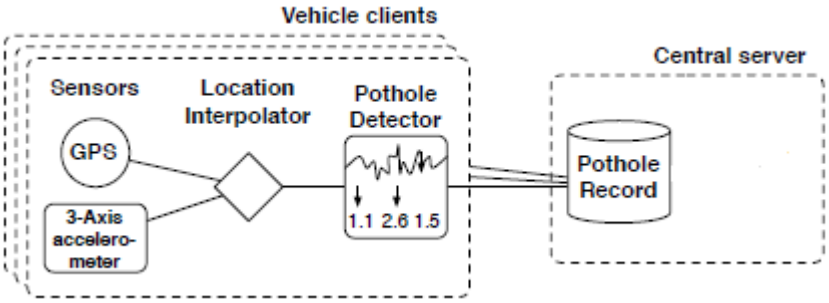

Figura 9: Arquitetura de Monitoramento da Rodovia. Fonte: Adaptada de [19].

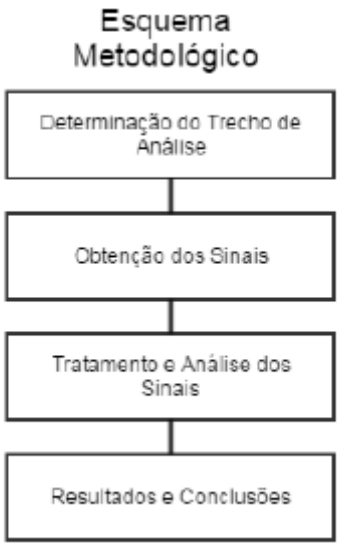

Figura 10: Esquema Metodológico de Monitoramento. Fonte: Autoria Própria.

2.4 PARTE ELETRÔNICA

2.4.1 Arduino Uno

A cada dia que passa a plataforma Arduino vem conquistando novos usuários. Esse sucesso é devido á sua simplicidade e ao fato de não necessitar conhecer profundamente a eletrônica e as estruturas de linguagens para criar gadgets, robôs ou pequenos sistemas inteligentes [20].

Figura 11: Arduino Uno. Fonte: [21].

Ele foi desenvolvido em 2005 com o objetivo de criar uma plataforma para o desenvolvimento simples e descomplicado de projetos interativos, utilizando um microcontrolador. Ele faz parte do que se conhece por computação física: área da computação em que o software interage diretamente com o hardware, tornando possível integração fácil com sensores, motores e outros dispositivos eletrônicos [22].

Ele é fundamentado em uma placa microcontrolada, com acessos de entrada/saída (I/O), sobre as quais foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma construção similar a das linguagens C e C++. O Arduino utiliza o microcontrolador Atmega. Este microcontrolador (também denominado MCU) é um computador em um chip, que contém um microprocessador, memória e periféricos de entrada/saída. Ele pode ser embarcado no interior de algum outro dispositivo, que, neste caso, é o Arduino, para que possa controlar suas funções ou ações, segundo [23].

Basicamente esse microcontrolador é responsável por manipular, interpretar e transformar os sinais de tensão e corrente a partir da entrada.

Em síntese, o Arduino é um conjunto de componentes que podem ser vistos como uma única unidade de processamento hábil suficiente para avaliar e medir sinais do meio externo, e transforma-los em sinais elétricos equivalentes. Isso é possível através de sensores ligados aos terminais de entrada do Arduino, que após obterem a informação processam-na computacionalmente. Por fim, ele pode ainda

intervir no gerenciamento ou no acionamento de algum outro elemento eletroeletrônico conectado ao terminal de saída. A figura abaixo apresenta o diagrama esquematizado exemplificando a função de cada componente do Arduino uno.

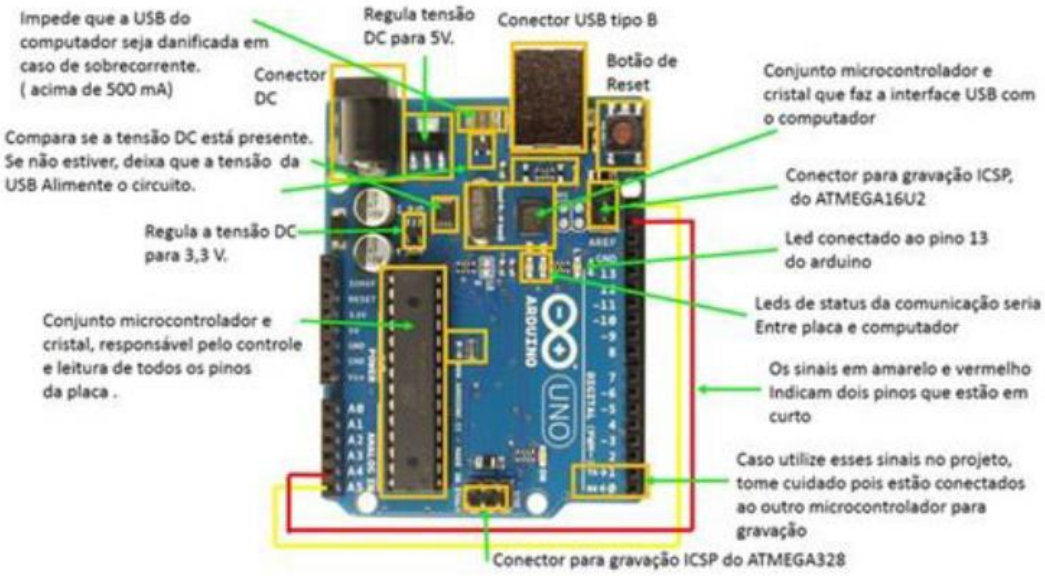

Figura 12: Blocos Identificados da Placa Arduino Uno. Fonte :Autoria Própria.

2.4.2 Shield GPS/GPRS/GSM

A placa Shield GPS / GPRS / GSM V3.0 será utilizada na proposta do trabalho. Esta placa é produzida pela DFRobot e possui a tecnologia quad-band embutida. Isso significa dizer que o sistema suporta quatro bandas de frequência diferentes para GSM: 850 MHz, 900 MHz, 1800 MHz e 1900 MHz [22].

A maioria dos países tem pelo menos uma rede GSM, a maior parte das redes GSM em todo o mundo usa uma das quatro bandas. Logo, ao comportar todas as quatro bandas, o sistema quad-band é compatível com uma ampla porção das redes GSM. Além do mais, ele também suporta a tecnologia GPS, para a navegação por satélite. Com isso é possível o sistema de controle enviar mensagens, utilizando a rede GSM, indicando as coordenas geográficas da posição do Shield [13].

Essencialmente o Shield é controlado através de comandos, segundo [25] e o seu design permite transferir as funções GSM e GPS diretamente ao computador e à placa Arduino. Na placa ainda está inclusa uma antena de alto ganho SMD para o

GPS, como também para o GSM e um chip SIM908 [22]. Juntos, todos esses componentes se combinarão e atuarão apresentando uma interface padrão das funções GPS, GSM e GPRS, proporcionando perfeitamente o rastreamento e monitoramento de meios de transporte em qualquer que seja o local e o momento, com cobertura de sinal.

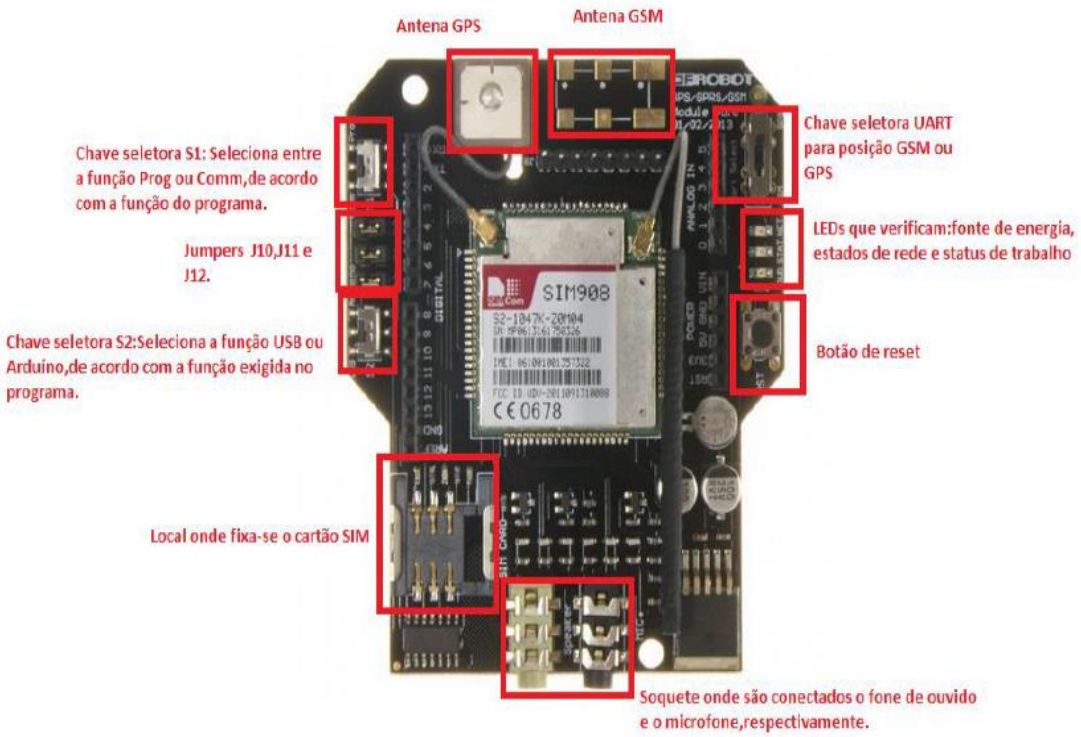

Figura 13: Blocos Identificados da Placa Shield GPS/GPRS/GSM. Fonte: Autoria Própria.

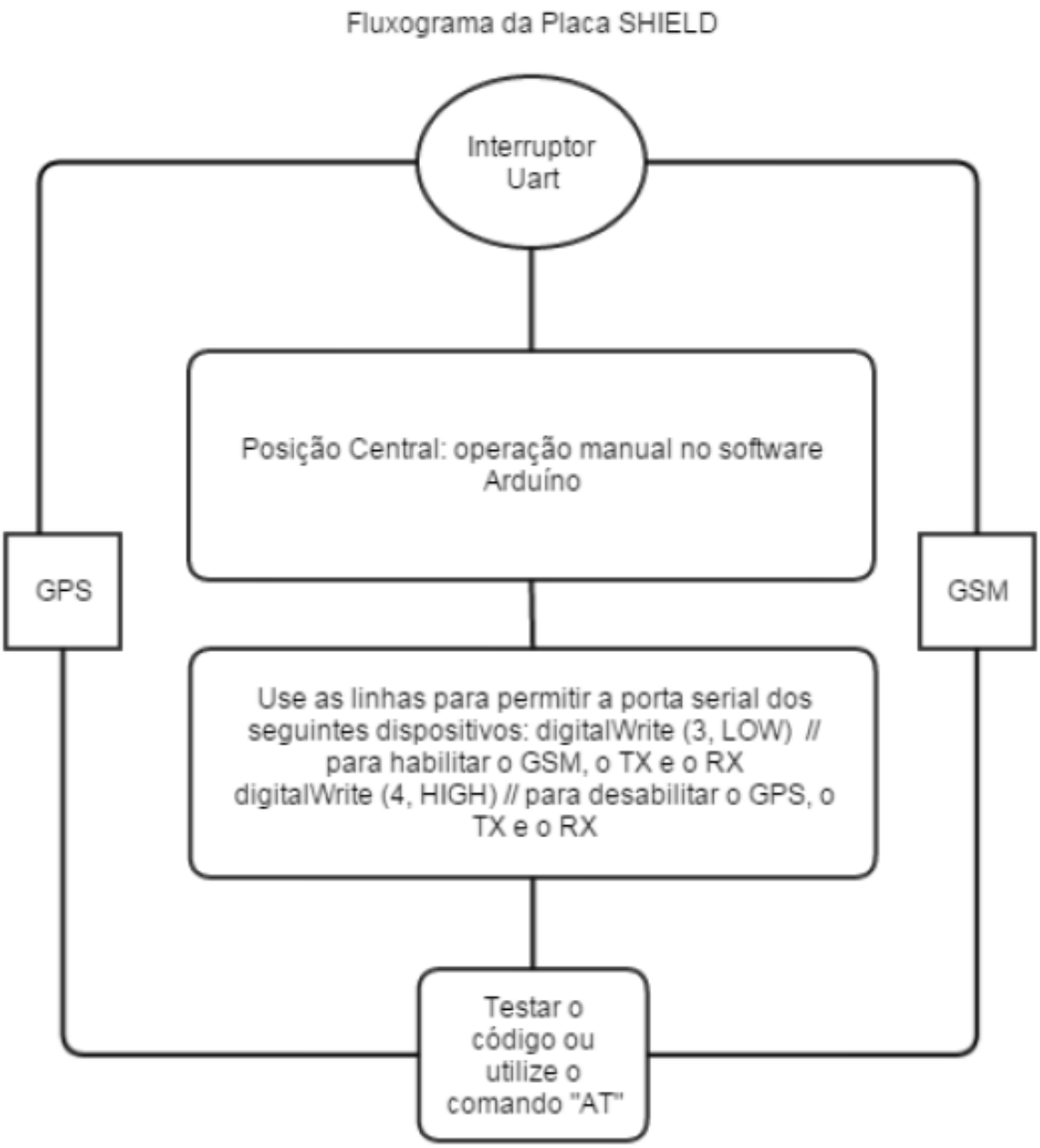

Figura 14: Fluxograma do SHIELD - Interruptor UART. Fonte: [22].

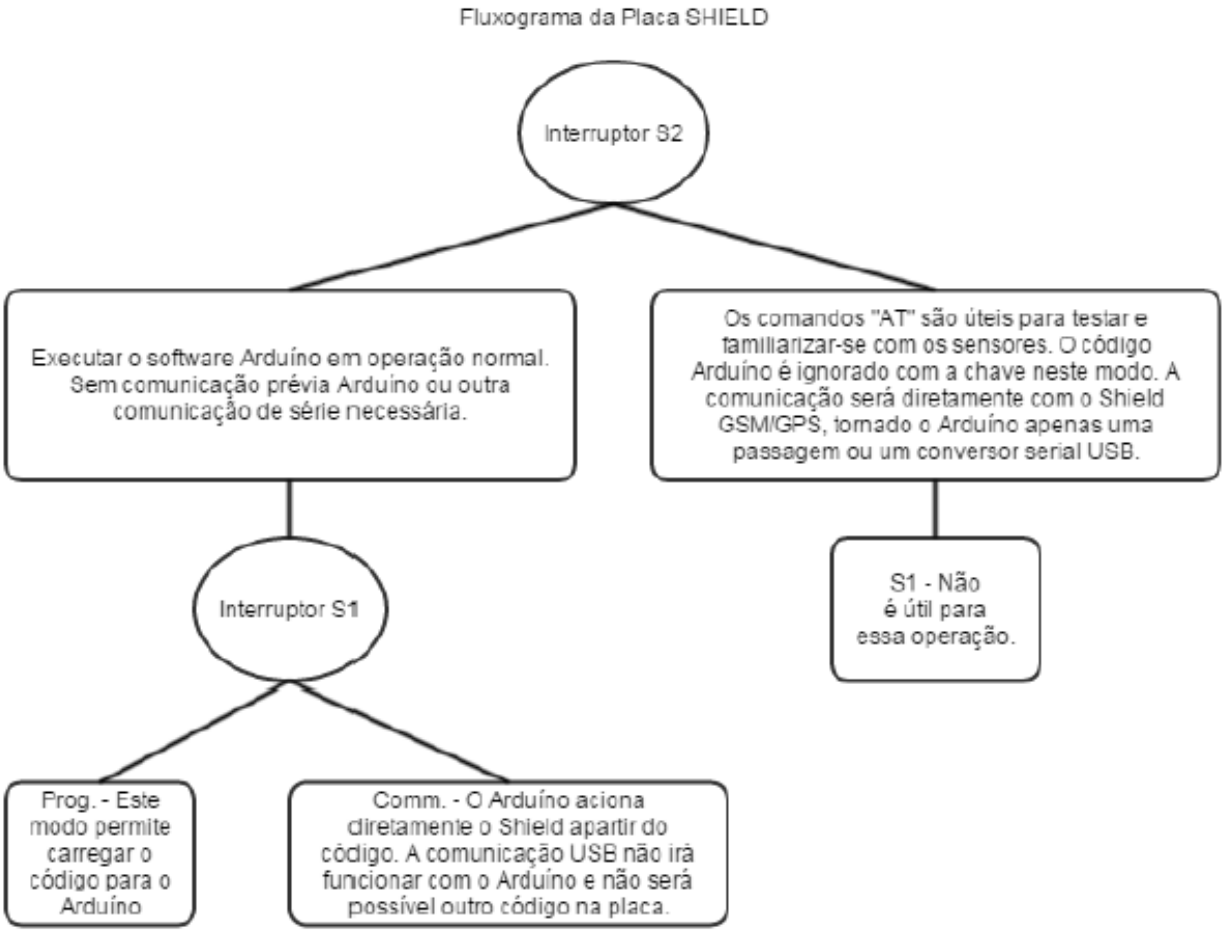

Figura 15 : Fluxograma de Funcionamento do Shield- Chave S2. Fonte: [22].

2.4.3 Sensor MPU-6050

A placa GY-521 é baseada no CI MPU-6050(Anexo, Datasheet MPU-6050) da InvenSense. Este CI possui, no mesmo revestimento, um acelerômetro e um giroscópio de alta precisão com tecnologia MEMS [20] [21]. No total são seis eixos, sendo três para o acelerômetro e três para o giroscópio. Na parte mais interna ele apresenta um recurso chamado DMP (Digital Motion Processor). O DMP possibilita que o algoritmo de detecção de movimento seja processado no próprio CI, livrando o microcontrolador dessa tarefa. O DMP também faz a aquisição do acelerômetro, giroscópio e sensor adicional e faz o processamento dos dados. O resultado pode ser lido diretamente.

O MPU-6050 possui internamente conversores A/D de 16 bits de resolução para cada canal, onde todos os sinais podem ser amostrados ao mesmo tempo. Internamente há um buffer FIFO de 1024 bytes, onde os valores podem ser armazenados e depois lidos, conforme configuração desejada. Outro recurso interessante do MPU-6050 é o sensor de temperatura interno que permite medidas de -40 °C a +85 °C [27].

A comunicação é feita através do padrão I2C usando os pinos SCL e SDA. A Figura abaixo exibe a pinagem da placa GY-521.

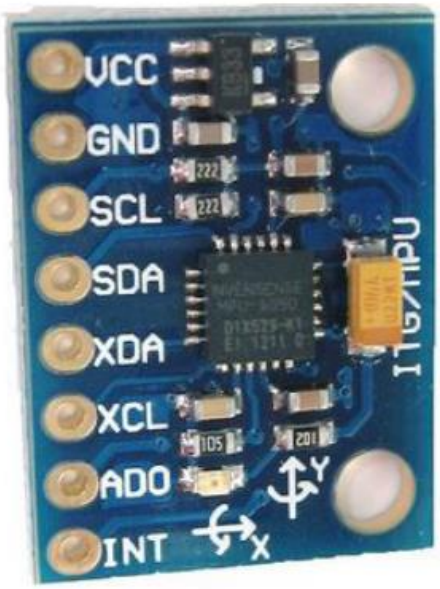

Figura 16: Placa GY-521. Fonte: [28].

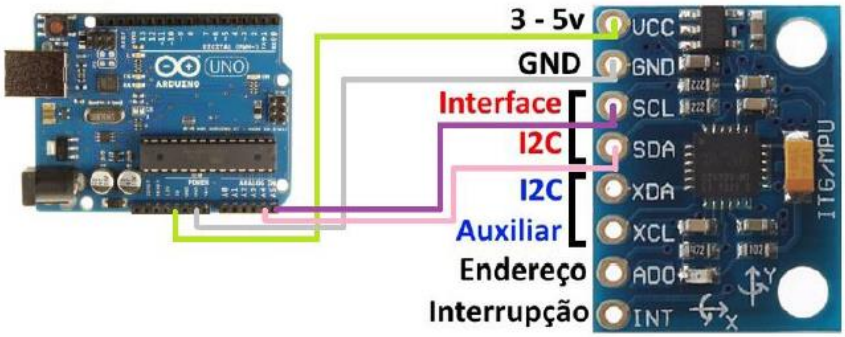

Figura 17: Conexão entre Arduino e o Sensor. Fonte: [29].

3 MATERIAIS E MÉTODOS

3.1 DESENVOLVIMENTO DO SISTEMA

O sistema de monitoramento proposto neste trabalho é projetado para rastrear e monitorar veículos e rodovias. Consiste basicamente em uma integração de várias tecnologias de comunicação moderna: GPS, GSM e GPRS. Para fornecer a localização e informação de tempo em qualquer lugar na terra, o Sistema de Posicionamento Global será utilizado como um aparelho de navegação global por satélite com base no espaço. As informações de localização fornecidas pelo GPS serão enviadas via SMS e poderão ser visualizadas usando o Google Earth. Neste trabalho também serão exploras características e relações de aceleração e vibração que podem ser úteis para expressar ou estimar condição rugosidade da rodovia em análise.

O sistema é composto por:

- Um sensor acelerômetro e giroscópio (MPU6050) capaz de captar aceleração em três eixos ortogonais;
- Um microcontrolador (Arduino) para realização da aquisição e processamentos dos dados;
- Um Shield GPS, GPRS e GSM;
- Jumpers para conexão entre o microcontrolador e o sensor;
- Um cabo USB para a comunicação com o computador;
- Três softwares para interpretar os dados processados: IDE do Arduino, IDE do Processing e Google Earth.

3.2 HARDWARE

Os componentes de hardware utilizados para a construção deste trabalho representam toda parte eletrônica empregada para os fins de detecção de posição do veículo e de rugosidade da rodovia. Nessa parte estão o Shield GPS, GPRS e GSM, o sensor MPU6050, o microcontrolador (Arduino Uno), as conexões e toda a troca de informações entre os dispositivos.

3.2.1 Junção do Shield GPS\GPR\GSM com Arduino

Para obter a posição do veículo e verificar o trajeto por onde o protótipo passou, foi utilizado o Shield GPS\GPRS\GSM interligado ao Arduino Uno.

O sistema tem dois módulos principais, como se mostra nas figuras 18, 19 e 20. O primeiro módulo é o dispositivo rastreador que será instalado no automóvel em movimento. Este módulo compõe-se de: um SHIELD GPS\GPRS\GSM, um microcontrolador Arduino Uno e um computador portátil. Funciona da seguinte forma: o receptor GPS do SHIELD obtém as informações de localização a partir de satélites, sob a forma de latitude e longitude em tempo real [25].

O microcontrolador tem uma tarefa importante: processar a informação obtida do GPS, extrair valores desejados e transmiti-los. Seja por SMS utilizando a rede GSM ou via USB utilizando um computador portátil [25]. O segundo módulo é composto por um smartphone que trabalha associado ao protótipo. O smartphone fica responsável por receber o SMS que inclui as coordenadas de GPS. A eficácia do sistema é confiável devido à suficiência da rede de comunicação utilizada [25] [22].

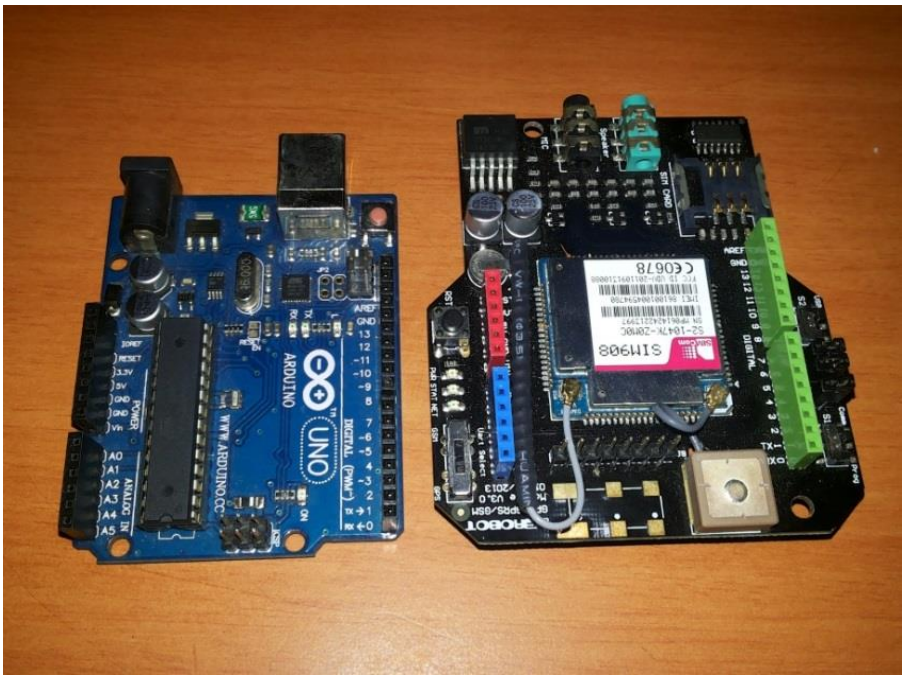

Figura 18: Microcontrolador e SHIELD. Fonte: Autoria Própria

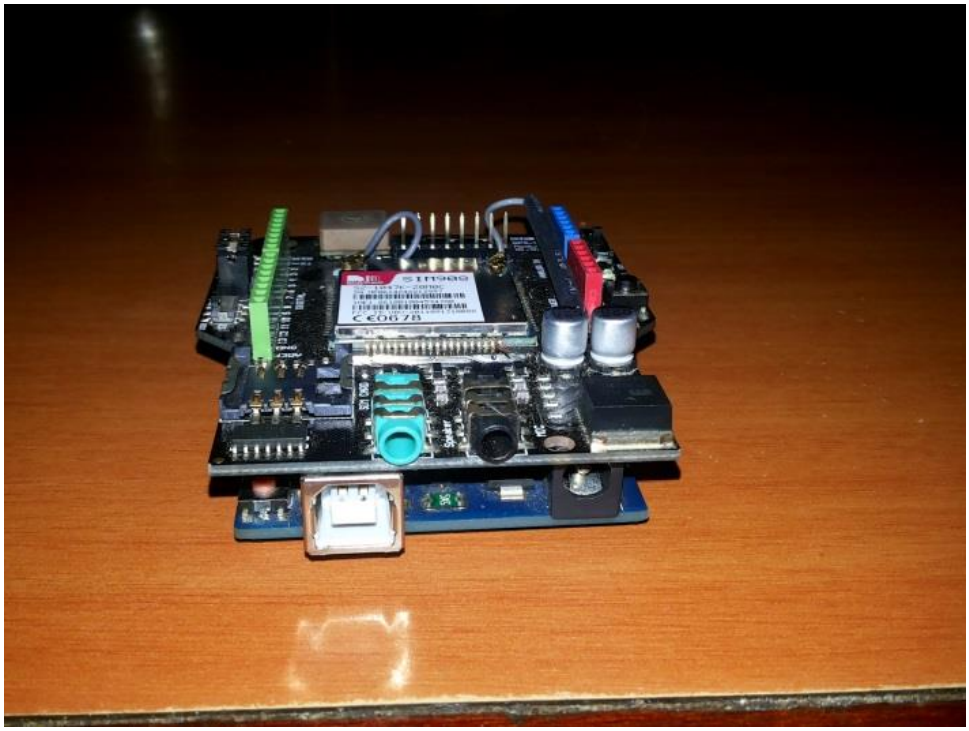

Figura 19: Junção dos Componentes. Fonte: Autoria Própria

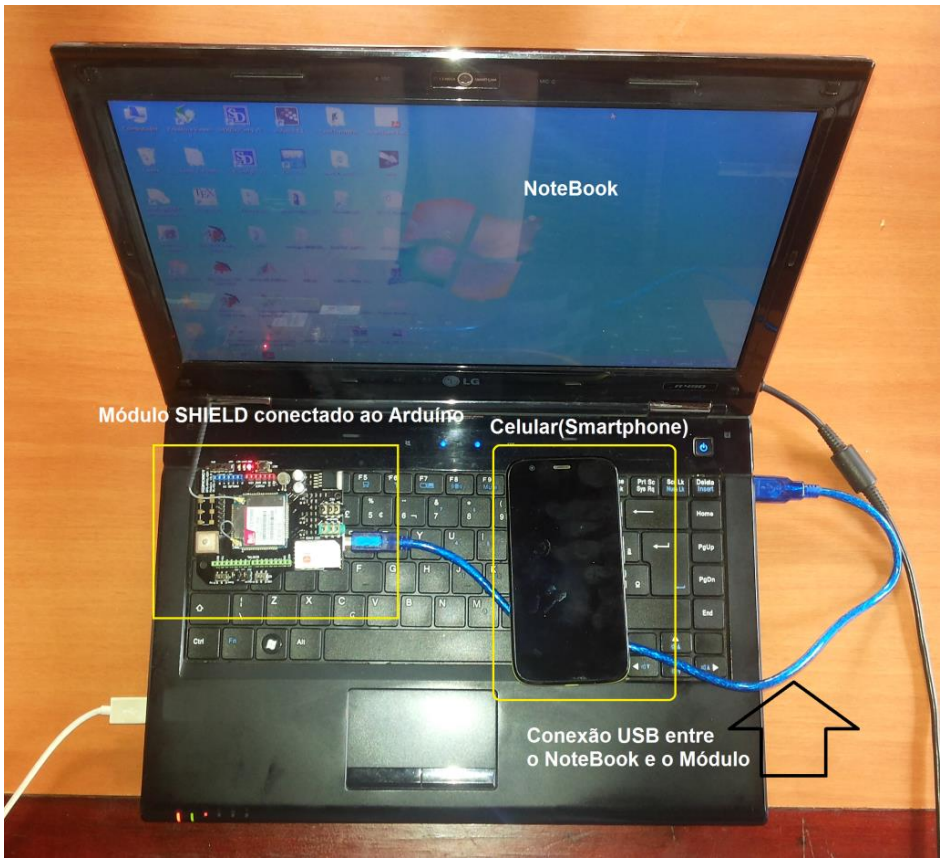

Figura 20: Esquema de Funcionamento da Coleta de dados GPS. Fonte: Autoria Própria.

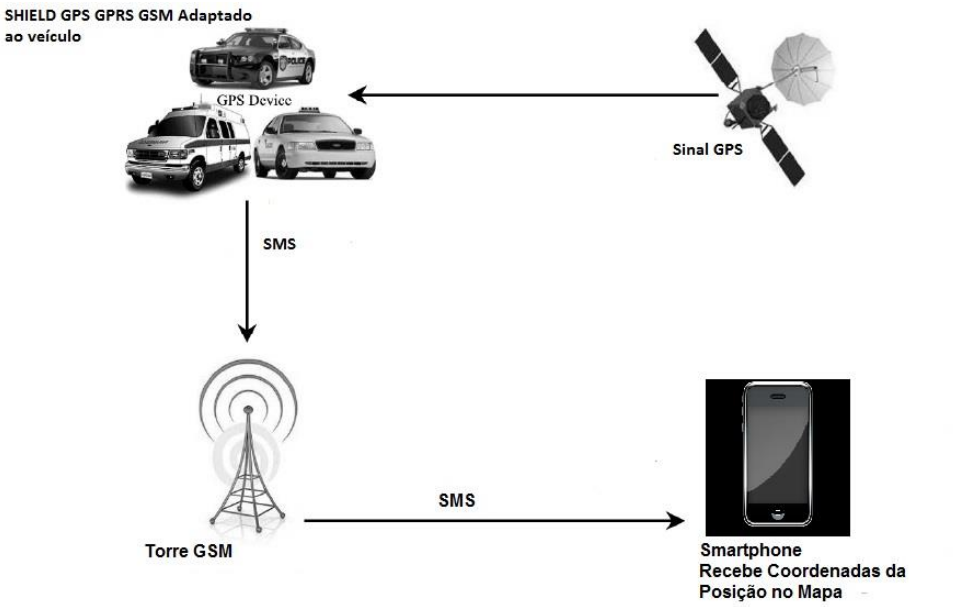

Figura 21: Diagrama de blocos do sistema de rastreamento via SMS. Adaptada de [27].

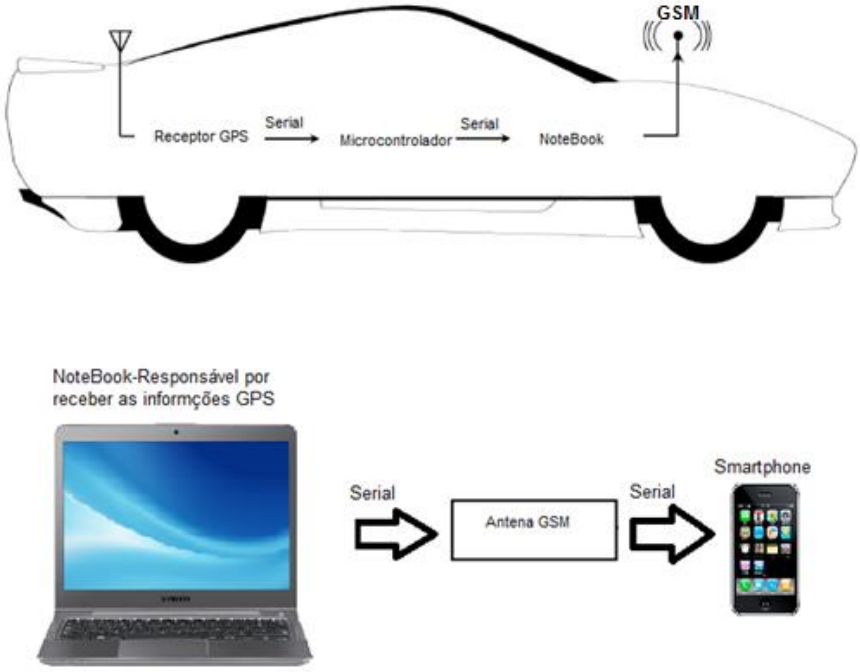

Figura 22: A arquitetura do sistema dentro do veículo: rastreamento por GPS utilizando a rede GSM. Fonte: Adaptada de [27]

3.2.1.1 Conexão Arduino e SHIELD

Para a interligação Arduino e SHIELD foi utilizado um circuito seguindo as recomendações do fabricante do SHIELD de forma a garantir a confiabilidade dos dados e garantia de comunicação. A conexão foi efetuada de forma que o SHIELD operasse conforme o seu diagrama esquemático, disposto na figura 23 [25].

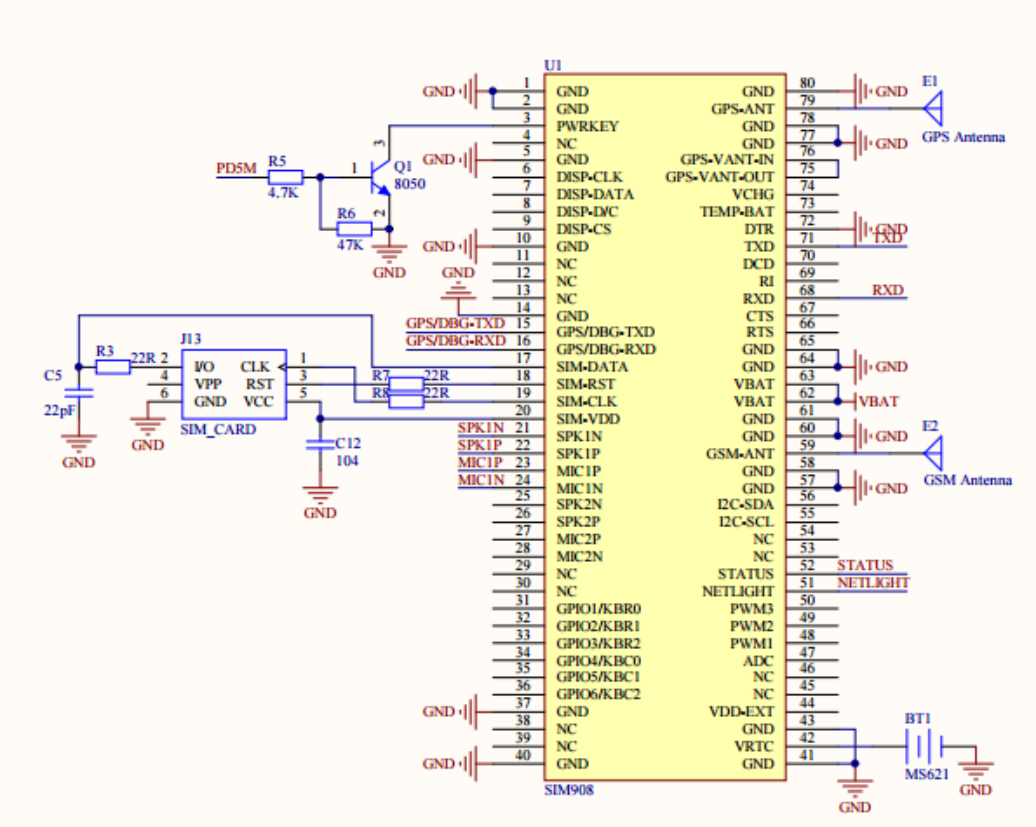

Figura 23: Diagrama Esquemático do SHIELD - Conexão de Pinos. Fonte: [25].

3.2.2 Junção entre o SHIELD GPS\GPRS\GSM, Arduino Uno e o Sensor MPU-6050

O funcionamento básico é descrito da seguinte forma: o sensor MPU6050 capta as alternâncias de aceleração dos três eixos (x, y e z) através do componente acelerômetro e também a variação angular de cada eixo através do componente giroscópio. Essas informações são então transferidas ao microcontrolador Arduino Uno por comunicação I2C, que as formatam em um pacote e então as enviam via porta serial, dentro da IDE de desenvolvimento do Arduino [28].

O sensor MPU-6050 basicamente consiste em um Sistema Microeletromecânico (MEMS) baseado na integração de três eixos-giroscópio e três

eixos-acelerômetro com um Processador Digital de Movimento (DMP) em um único chip. Além disso, o MPU-6050 utiliza protocolos de comunicação I2C. Algumas das principais características do o dispositivo são dadas abaixo [29]:

- Fusão de Movimento pelo chip DMP
- Faixa de tensão de alimentação 2.375 - 3.46V
- Sensibilidade do eixo transversal mínima entre acelerômetro e giroscópio
- Buffer FIFO de 1024 bytes reduz o consumo de energia, permitindo o processador host ler os dados em rajadas e, em seguida, entrar em um modo de baixo consumo, possibilitando o MPU6050 recolher mais dados.

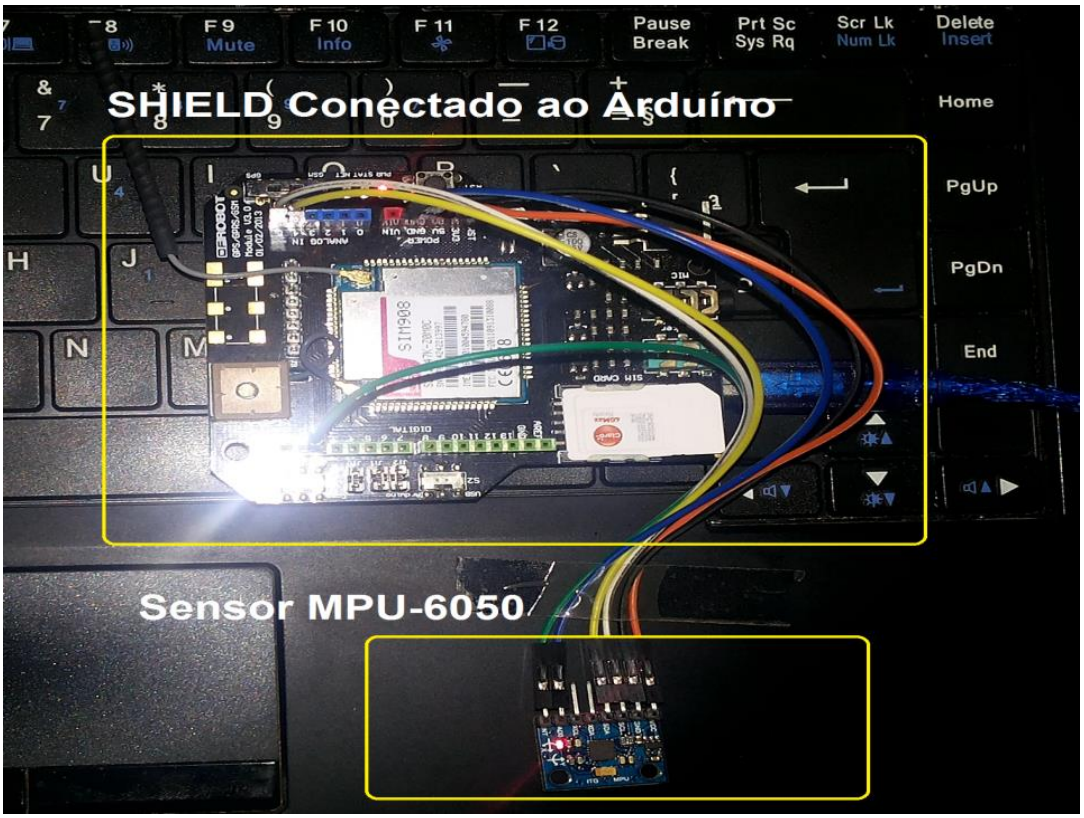

Figura 24: Esquema de Funcionamento da Coleta de Dados do Sensor MPU-6050. Fonte: Autoria Própria.

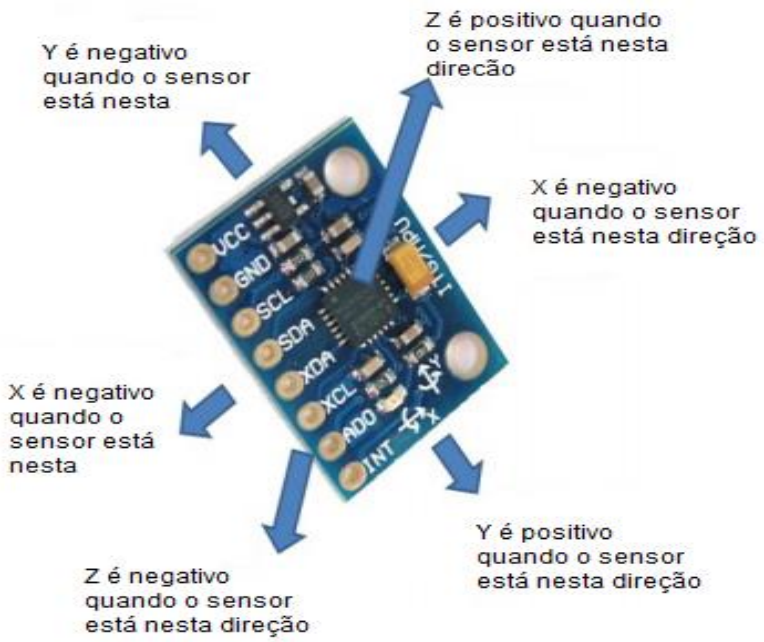

Figura 25 Princípio do Algoritmo de Reconhecimento. Adaptada de [30].

3.2.2.1 Arduino e Sensor MPU-6050

Para unir o Arduino e o sensor, foi fundamental respeitar as recomendações do fabricante do acelerômetro de forma a garantir a segurança dos dados e da comunicação. As seguintes ligações, seguindo o diagrama esquemático da placa, foram efetuadas de forma que o sensor trabalhasse no modo de comunicação I2C:

Tabela 1: Mapeamento de pinos entre sensor MPU6050 e Arduino. Fonte: [32].

MPU-6050	Arduino Uno
SCL	A5
INT	PIN2
ADO	GND
GND	GND
VCC	3.3V
SDA	A4

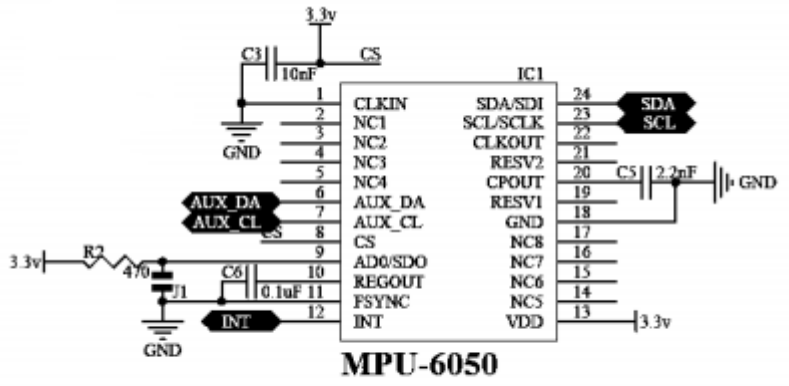

Figura 26: Diagrama Esquemático do Sensor MPU-6050. Fonte: [29].

3.3 SOFTWARE

3.3.1 Aquisição de Posição

Para encaixar as posições adquiridas pelo GPS e visualizar o status do automóvel durante o trajeto adotado (BR-020/EPIA/BR-040) foi utilizado o software Google Earth, que suporta a maioria dos dispositivos de posicionamento global.

O módulo GPS acoplado tem protocolo NMEA 0183 que permite a transmissão de informações do sistema SHIELD\Arduino a um PC. Basicamente este protocolo é composto por várias frases, começando com o caractere $, com no máximo 79 caracteres por comprimento. A mensagem NMEA que lê dados, como a posição e o tempo são: $ GPRMC [33]. Consequentemente, apenas a informação “$ GPRMC” é usada para determinar a localização do automóvel, pois assim o texto SMS se torna otimizado e reduzido.

Sendo assim o estado do automóvel, juntamente com informações $ GPRMC são enviados através de uma antena GSM SMD de alto ganho [25]. Por conseguinte, o receptor GSM, neste caso o smartphone, recebe a transmissão SMS e obtém as coordenadas de GPS e as informações sobre o status do automóvel.

Os dados de GPS transmitidos também são enviados via porta USB. Os dados resultaram de parâmetros de posição e são instantaneamente armazenados em um arquivo TXT. O arquivo TXT é exportado para um arquivo KML, compatível

com programa Google Earth. Por isso, o Google Earth irá ver a localização e a disposição do automóvel no mapa lendo o arquivo KML.

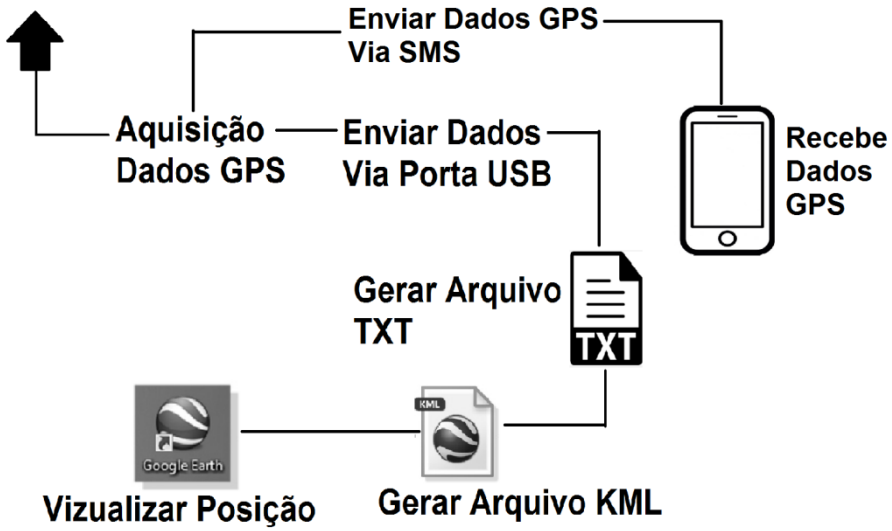

Figura 27: Diagrama de Funcionamento do Software Para aquisição de Dados GPS. Fonte: Autoria Própria

3.3.2 Condição da Superfície da Estrada

3.3.2.1 Métrica

Para abordar as características de medição de rugosidade, incluindo aspectos de precisão, primeiramente é importante definir a escala de rugosidade [34]. O Índice de Rugosidade Internacional (IRI) é uma escala de rugosidade padrão relacionada com medidas obtidas pelos sistemas de perfilômetros, que nada mais são do que um conjunto de equipamentos que determinam e medem o perfil da superfície do pavimento mediante o contato e deslizamento de uma roda sobre o mesmo. Devido à superfície rugosa dos pavimentos a roda sobe e desce descrevendo a topografia da superfície sobre a qual desliza. Os mais importantes perfilômetros são os perfilômetros inerciais a laser [42].

Já o IRI foi selecionado no interesse de incentivar a utilização de uma medida de rugosidade comum em todos os projetos significativos em todo o mundo [34].

O modelo mais empregado e mais utilizado para o desenvolvimento de um controlador de baixo nível para uma suspensão do veículo é o modelo *quarter-car*, onde apenas um quarto do veículo é estudado [36]. Para obter a análise dinâmica do sistema de suspensão do veículo, deve-se estabelecer o modelo de veículo apropriado [36]. Por isso neste trabalho é utilizado o modelo *quarter-car*, com 2 graus de liberdade, que é o modelo de dinâmica de base para a concepção de um sistema de suspensão onde o movimento apenas na direção Z é levado em consideração, como mostrado na Figura 28.

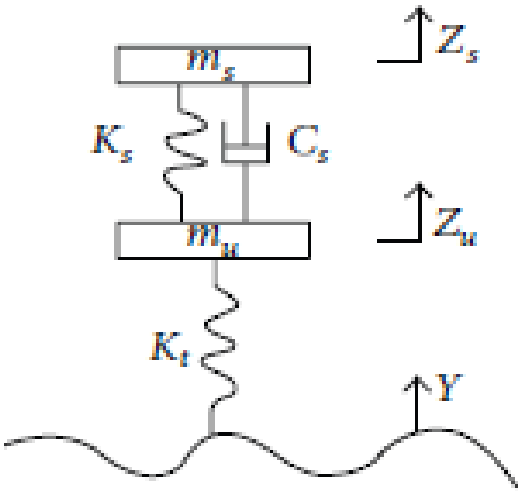

Figura 28: Modelo de Quarter-Car. Fonte: [36].

De acordo com a figura 28, a equação cinética do modelo *quarter-car* é dada a seguir [36]:

$ms\ddot{Z}\,s + Cs(\dot{Z}s - \dot{Z}u) + Ks(Zs - Zu) = 0,$
$mu\ddot{Z}\,u + Cs(\dot{Z}u - \dot{Z}s) + Ks(Zu - Zs) + Kt(Zu - Y) = 0$

Eliminando as massas a partir das equações anteriores:

$\ddot{Z}s + Cs(\dot{Z}s - \dot{Z}u) + K1(Zs - Zu) = 0,$
$u\ddot{Z}\,u + C(\dot{Z}u - \dot{Z}s) + K2(Zu - Zs) + K1Zu = K1Y$

Usando a resposta do modelo *quarter-car* em uma viagem com velocidade de 80 km / h, calculado para cada ponto ao longo da distância de curso, o IRI pode ser definido da seguinte forma:

$$\frac{1}{L}\int_0^L |Zs - Zu|\, dx = \text{IRI}$$

Onde L é a distância ao longo da estrada em que a medição é executada, Zs é a velocidade vertical da massa suspensa e Zu é a velocidade vertical da massa não suspensa.

Esse sistema é conhecido por ser capaz de refletir as características de aceleração, velocidade e deslocamento da vibração do corpo no sistema de suspensão do veículo, com excitação [36] [37].

Comparado com o modelo de suspenção completo do veículo, o modelo *quarter-car* com 2 graus de liberdade tem menos design e parâmetros de desempenho justamente para simplificar a entrada do sistema [37].

3.3.2.2 Arduino - Interface com MPU-6050

3.3.2.2.1 Calibração do Sensor

Esta parte do trabalho se concentra no arranjo do sensor MPU-6050, acelerômetro e giroscópio, em relação ao microcontrolador Arduino Uno e como o *Processing,* programa utilizado para a calibração do sensor, está configurado para interpretar os dados de série para exibição visual de uma animação em três dimensões.

A figura 29 exemplifica bem como as inclinações e rotações acontecem em diferentes direções do plano 3D do *Processing,* fundamentado nos dados do MPU-6050.

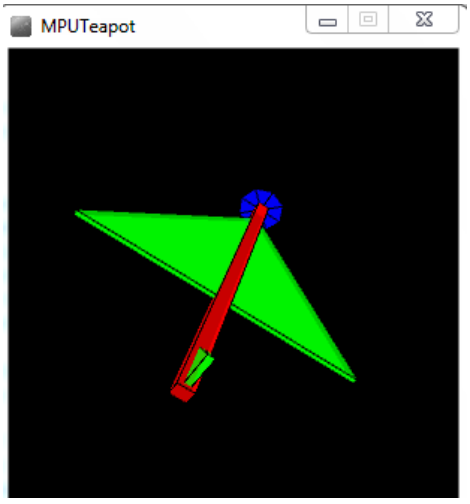

Figura 29: Modelando os Valores do MPU-6050 em Três Dimensões Usando o Processing. Fonte:[26].

Para executar a análise e visualizar a calibração do sensor, basicamente o que precisa ser feito é estabelecer uma conexão entre o MPU-6050 e o Arduino via barramento I2C, os dados de 8bits SDA e o *clock* SCL. É importante programar o Arduino para enviar os dados a partir do MPU-6050 ao programa de processamento visual via comunicação serial [39].

Figura 30: Programação no Arduino. Fonte: [26].

O *Processing* é programado para desenhar objeto 3D em diferentes direções com base nos dados de MPU-6050 programados no Arduino.

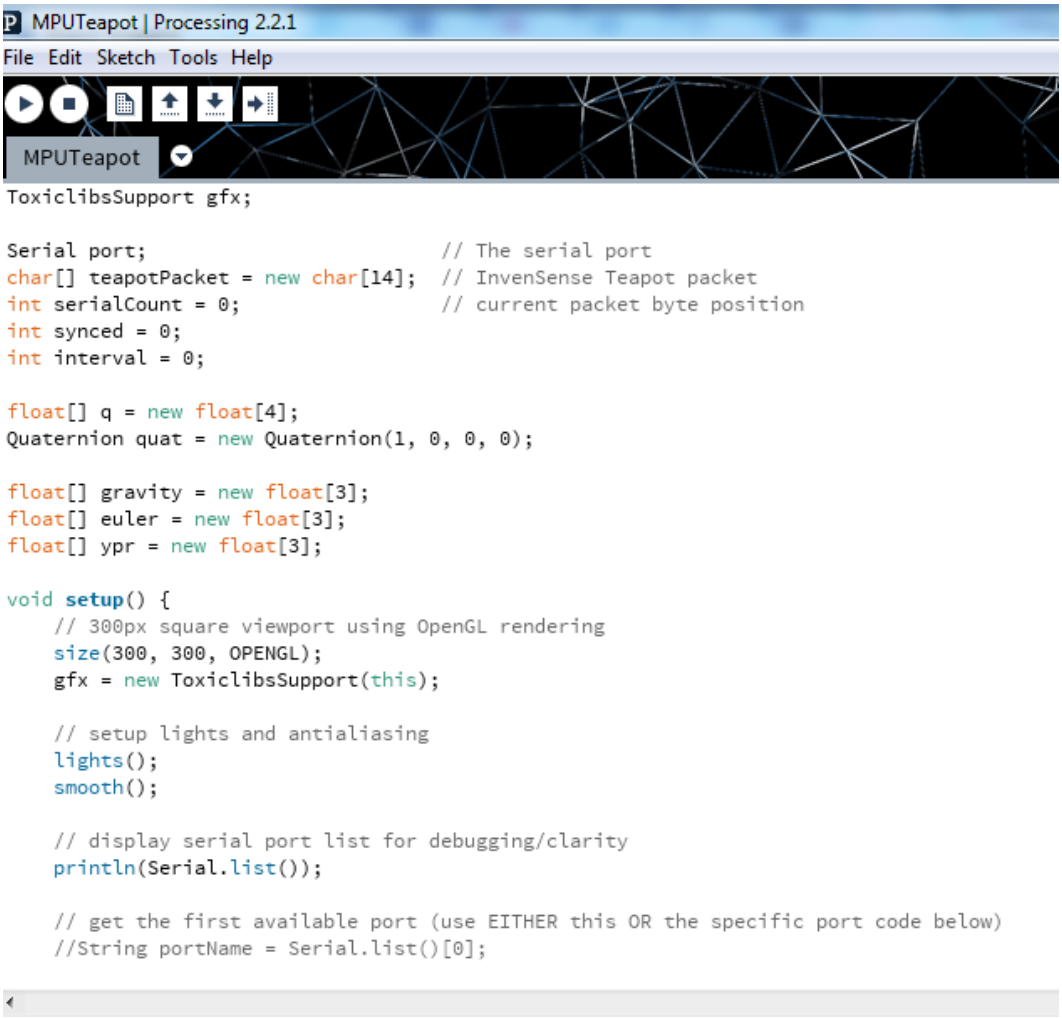

Figura 31: Programação no Processing. Fonte: [26].

Um detalhe importante, que permite o usuário alcançar tais resultados de calibração, é o MPU-6050 possuir internamente um recurso chamado DMP (*Digital Motion Processor*). O DMP permite que o algoritmo de detecção de movimento seja analisado no próprio circuito integrado, liberando assim o Arduino dessa função [5]. O DMP faz a leitura do acelerômetro, giroscópio e sensor de temperatura e faz a análise dos dados. O resultado pode ser identificado diretamente [5] [31]. E é isso que está acontecendo durante a execução do programa de teste.

3.3.2.2.2 Aquisição de Dados da Rodovia

Com o objetivo de identificar as fissuras e rugosidade da rodovia, primeiramente, foram levadas em conta as acelerações nos eixos X, Y e Z.

Inicialmente, caso acontecesse uma perturbação muito abrupta nos valores de entrada, o programa detectaria um possível problema na estrada.

176	Acel. X = 316	Y = -5048	Z = 15140	Gir. X = -845	Y = -311	Z = -62	Temp = 30.55
177	Acel. X = 904	Y = -4496	Z = 15484	Gir. X = -217	Y = 23	Z = -28	Temp = 30.69
178	Acel. X = 936	Y = -4528	Z = 16444	Gir. X = -542	Y = -362	Z = -131	Temp = 30.60
179	Acel. X = 964	Y = -4480	Z = 15140	Gir. X = -1105	Y = 352	Z = 49	Temp = 30.55
180	Acel. X = 804	Y = -4912	Z = 13100	Gir. X = -1057	Y = 227	Z = 72	Temp = 30.55
181	Acel. X = 344	Y = -2200	Z = 15604	Gir. X = -641	Y = 7	Z = 37	Temp = 30.60
182	Acel. X = 1856	Y = -1920	Z = 18056	Gir. X = -851	Y = 43	Z = 74	Temp = 30.60
183	Acel. X = -172	Y = -1232	Z = 16748	Gir. X = -823	Y = 351	Z = 174	Temp = 30.51
184	Acel. X = 1836	Y = -4500	Z = 14684	Gir. X = -1388	Y = 482	Z = 234	Temp = 30.51
185	Acel. X = 288	Y = -3272	Z = 16104	Gir. X = -1029	Y = -203	Z = 31	Temp = 30.46
186	Acel. X = -472	Y = -6184	Z = 16584	Gir. X = -982	Y = 780	Z = -4	Temp = 30.46
187	Acel. X = 2388	Y = -3388	Z = 20160	Gir. X = -896	Y = -122	Z = -56	Temp = 30.46
188	Acel. X = 892	Y = -5496	Z = 15124	Gir. X = -1550	Y = 372	Z = 49	Temp = 30.41
189	Acel. X = -240	Y = -532	Z = 16284	Gir. X = -621	Y = 5	Z = 235	Temp = 30.51
190	Acel. X = 2108	Y = -4868	Z = 18228	Gir. X = -1229	Y = 104	Z = -100	Temp = 30.60

Figura 32: Variação Abrupta das Acelerações Nos Eixos X, Y e Z. Fonte: Autoria Própria.

Com a análise em cima de diversas tentativas e testes realizados com o módulo Arduino\SHIELD\MPU-6050 no trecho compreendido entre BR-020\ EPIA\ BR-040, ficou constatado que seria fundamental simplificar a abordagem e interpretação dos dados coletados, pois seria de maior interesse finalizar a ideia principal deste trabalho.

Por isso as simplificações efetivadas foram:

- Estudar e analisar apenas a variação da aceleração lida no eixo Z pelo módulo.
- Entender como “buraco” qualquer alteração fora da escala determinada para o eixo Z.
- Conferir duas leituras de aceleração dentro de uma determinada distância, seguindo o padrão IRI.

4 RESULTADOS E DISCUSÕES

Este capítulo irá apresentar a integração dos módulos individuais e teste final. Em primeiro lugar, os módulos foram combinados um de cada vez e foram testados para demonstrar a suas funcionalidades .

Uma vez que cada módulo estava trabalhando, os respectivos dados foram coletados e analisados.

Informações como uma análise de execução dos módulos, graficos relacionados a perfis de pavimento e dados da posição, verificados tanto no *smartphone* quanto no *Google Earth*, serão apresentados.

4.1 DADOS GPS

O módulo (Arduino Uno\SHIELD GPS, GPRS e GSM) em atividade foi testado no trecho que vai do início da EPIA Norte até o fim da EPIA Sul .A priori o teste experimental foi realizado sem o uso de uma bateria alcalina de 9V, sendo o módulo alimentado via cabo USB através de um notebook. Isso acabou possibilitando a aquisição dos dados instantaneamente via porta serial, por meio da IDE do Arduino. Isso significa dizer que a informação era adquirida imediatamente pelo usuário ,conforme o deslocamento do carro. Seja essa informação tanto de posição(longitude e latitude) quanto de aceleração(eixo Z do sensor).

O procedimento seguido para a simulação limitou-se em dois importantes estágios. O primeiro passo foi obter os dados GPS do SHIELD através do programa, ressaltando que os resultados são mais corretos se o módulo estiver em um ambiente externo com melhor acesso aos sinais de satélite GPS. Dependendo do local onde o módulo se encontra a intensidade do sinal vai variar, e pode ser que leve tempo para que o SHIELD forneça a localização de forma precisa.

Após carregar o programa (Anexo I. 1) no Arduino, o interruptor UART deve ser colocado na posição GSM e o serial monitor deverá ser iniciado para que os comandos AT sejam fornecidos [18] [19] :

- AT
- AT+CGPSIPR=9600 (seleciona a velocidade de comunicação)

- AT+CGPSPWR=1 (liga o GPS)
- AT+CGPSRST=1 (coloca o GPS em modo de autonomia)

É importante certificar-se de que cada comando acima obteve a resposta OK do Shield

Feito isso, deve-se colocar o interruptor UART na posição GPS e esperar a transmissão de dados no próprio serial monitor.

```
$GPGGA,005744.000,1551.307467,S,04757.341826,W,1,10,0.87,1044.110,M,-9.715,M,,*43
$GPGLL,1551.307467,S,04757.341826,W,005744.000,A,A*5E
$GPGSA,A,3,18,22,26,29,21,16,15,14,25,20,,,1.71,0.87,1.47*0E
$GPGSV,3,1,12,18,75,182,23,22,55,302,40,26,48,265,41,29,43,080,25*73
$GPGSV,3,2,12,21,40,168,35,16,29,232,35,15,21,115,34,14,17,001,26*7F
$GPGSV,3,3,12,25,15,015,29,20,14,142,31,31,06,322,,27,03,220,*75
$GPRMC,005744.000,A,1551.307467,S,04757.341826,W,33.703,188.4,201115,,,A*5E
$GPVTG,188.4,T,,M,33.703,N,62.451,K,A*08
$GPZDA,005744.000,20,11,2015,,*50

$GPGGA,005745.000,1551.316963,S,04757.343687,W,1,10,0.87,1042.491,M,-9.714,M,,*47
$GPGLL,1551.316963,S,04757.343687,W,005745.000,A,A*51
$GPGSA,A,3,18,22,26,29,21,16,15,14,25,20,,,1.71,0.87,1.47*0E
$GPGSV,3,1,12,18,75,182,23,22,55,302,42,26,48,265,38,29,43,080,23*79
$GPGSV,3,2,12,21,40,168,28,16,29,232,36,15,21,115,33,14,17,001,26*77
$GPGSV,3,3,12,25,15,015,29,20,14,142,30,31,06,322,,27,03,220,*74
$GPRMC,005745.000,A,1551.316963,S,04757.343687,W,33.620,189.3,201115,,,A*57
$GPVTG,189.3,T,,M,33.620,N,62.298,K,A*0D
$GPZDA,005745.000,20,11,2015,,*51

$GPGGA,005746.000,1551.326264,S,04757.345052,W,1,10,0.87,1041.338,M,-9.714,M,,*44
$GPGLL,1551.326264,S,04757.345052,W,005746.000,A,A*55
$GPGSA,A,3,18,22,26,29,21,16,15,14,25,20,,,1.71,0.87,1.47*0E
$GPGSV,3,1,12,18,75,182,24,22,55,302,41,26,48,265,47,29,43,080,21*77
$GPGSV,3,2,12,21,40,168,31,16,29,232,35,15,21,115,31,14,17,001,27*7F
$GPGSV,3,3,12,25,15,015,29,20,14,142,29,31,06,322,,27,03,220,*7C
$GPRMC,005746.000,A,1551.326264,S,04757.345052,W,33.775,188.6,201115,,,A*56
$GPVTG,188.6,T,,M,33.775,N,62.586,K,A*00
$GPZDA,005746.000,20,11,2015,,*52

$GPGGA,005747.000,1551.335480,S,04757.346361,W,1,10,0.87,1040.489,M,-9.714,M,,*47
$GPGLL,1551.335480,S,04757.346361,W,005747.000,A,A*5A
$GPGSA,A,3,18,22,26,29,21,16,15,14,25,20,,,1.71,0.87,1.47*0E
$GPGSV,3,1,12,18,75,182,23,22,55,302,40,26,48,265,44,29,43,080,24*77
$GPGSV,3,2,12,21,40,168,24,16,29,232,33,15,21,115,31,14,17,001,23*79
$GPGSV,3,3,12,25,15,015,27,20,14,142,28,31,06,322,,27,03,220,*73
$GPRMC,005747.000,A,1551.335480,S,04757.346361,W,33.633,189.8,201115,,,A*55
$GPVTG,189.8,T,,M,33.633,N,62.321,K,A*07
$GPZDA,005747.000,20,11,2015,,*53
```

Figura 33: Dados GPS. Fonte: Autoria Própria

Verificando no Google Earth as coordenadas GPS que foram obtidas pelo SHIELD e separando os valores da linha $GPRMC conforme a figura 33, a posição exata no trecho será registrada conforme a figura 35.

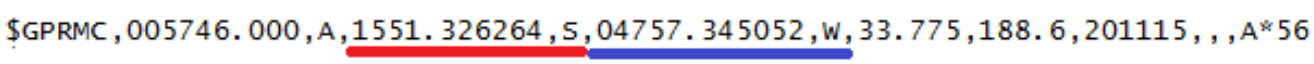

Figura 34: Latitude e Longitude. Fonte: Autoria Própria.

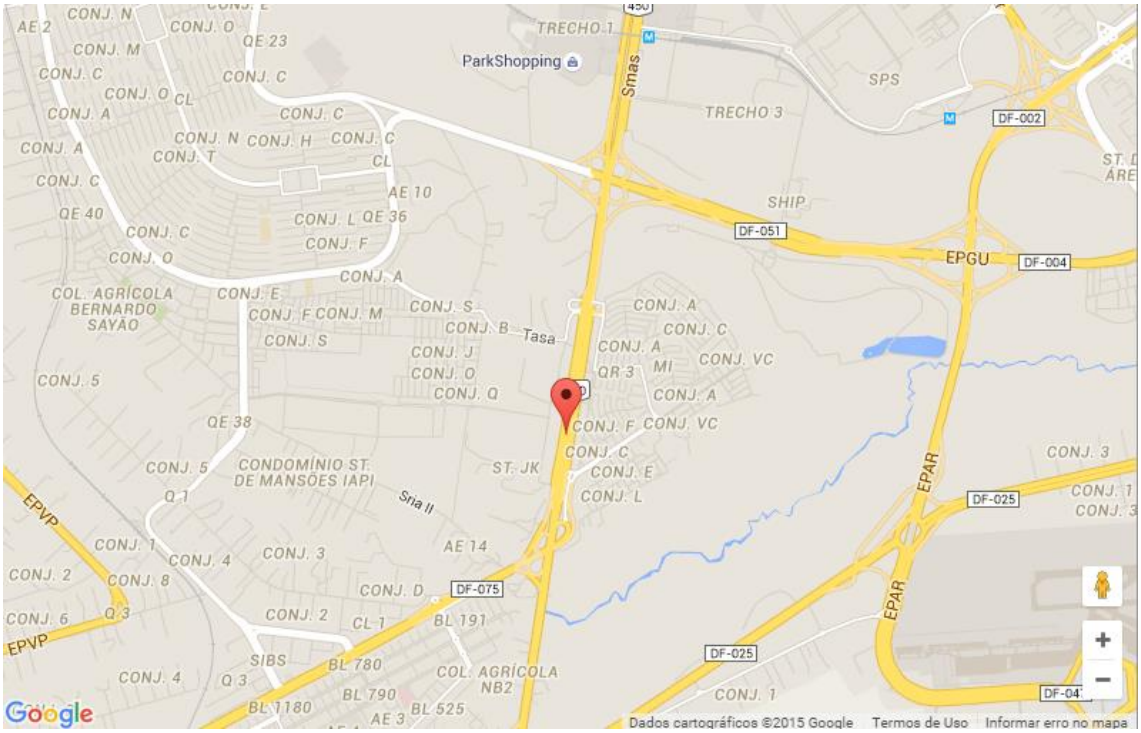

Figura 35: Representação da Posição Adquirida pelo SHIELD.

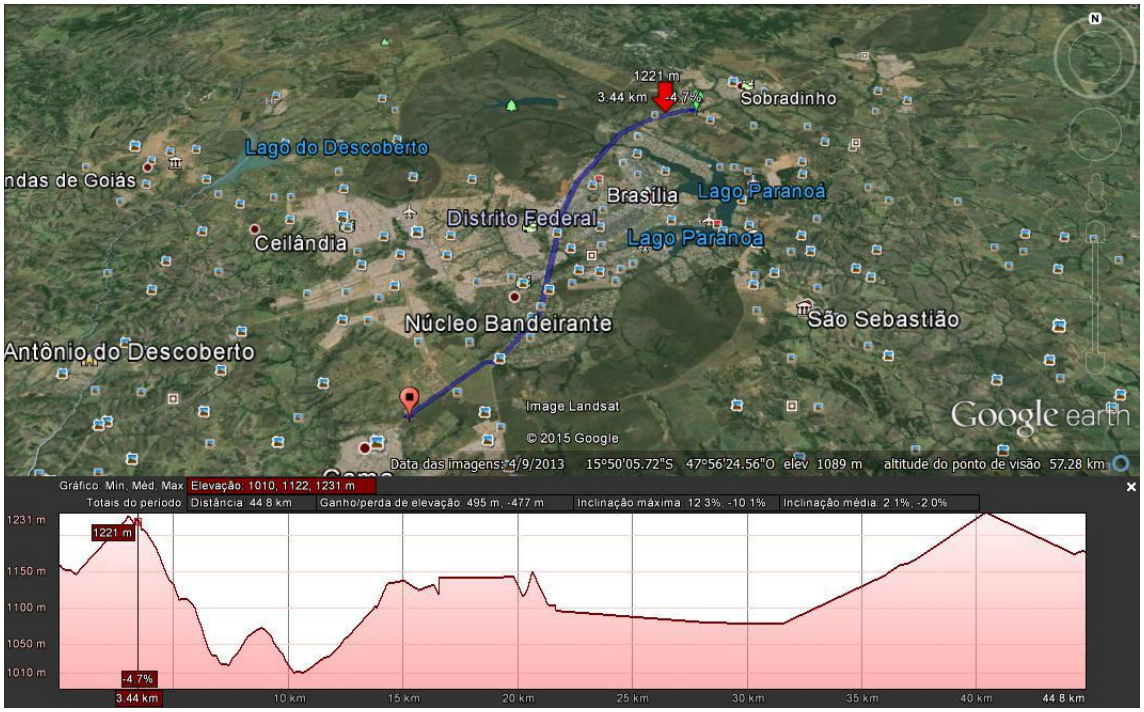

Figura 36: Representação de Toda a Rota Construída a Partir dos Dados Transmitidos pelo GPS. Fonte: Autoria Própria.

O segundo passo consistiu em carregar o módulo com um segundo programa (Anexo I.2) e , por meio de uma rede GSM ,fazer o envio das coordenadas do GPS

remotamente. Feito isso, é deve-se esperar a recepção de uma mensagem SMS indicando o local em que o usuário se encontra.

4.2 MEDINDO A RUGOSIDADE

Este sistema depende do acelerômetro e a amostragem de resultados consistentes para um dado buraco. Nesta seção, serão descritos alguns experimentos que foram realizados para validar o funcionamento do módulo conectado ao sensor.

4.2.1Aquisição de dados

Os instrumentos utilizados para a detecção, medição e leitura da aceleração foram instalados em um Fiat Palio Attractive 2013, 4 portas, como mostrado na figura 36. O Arduino Uno conectado ao sensor de aceleração MPU-6050 foram usados como dispositivos de verificação e foram posicionados juntamente com o *notebook* sobre o colo do passageiro frontal esquerdo para facilitar nivelamento. Esse modelo de perfilómetro é bem simples e pode ser montado em qualquer veículo comercial

Figura 37: Carro Teste. Fonte :Autoria Própria

O procedimento experimental ocupou a maior parte da elaboração do sistema, pois para a evolução do algoritmo “medidor de rugosidade” foi fundamental as simulações e arranjos constantes. Dessa forma, o estudo da variação da aceleração no módulo pode ser mais detalhado em diversos tipos de programa ao longo do desenvolvimento instrumental desse trabalho.

Primeiramente , testes identificando e amostrando os dados resultantes diretamente do MPU-6050 utilizando o microcontrolador Arduino Uno, foram realizados. Assim, conseguiu-se uma interpretação melhor dos valores.

Na Figura 37 são apresentados os resultados decorrentes do acelerômetro.

```
Acel. X = 1400 | Y = -1472 | Z = 16644 | Gir. X = -586 | Y = 504 | Z = 971 | Temp = 30.84
Acel. X = 1252 | Y = -2600 | Z = 16364 | Gir. X = -2897 | Y = 571 | Z = -1500 | Temp = 30.88
Acel. X = 1296 | Y = -2896 | Z = 16716 | Gir. X = -1055 | Y = 193 | Z = 49 | Temp = 30.74
Acel. X = 1328 | Y = -3040 | Z = 16796 | Gir. X = -891 | Y = 264 | Z = 48 | Temp = 30.88
Acel. X = 1352 | Y = -3112 | Z = 16640 | Gir. X = -912 | Y = 269 | Z = 40 | Temp = 30.84
Acel. X = 1384 | Y = -3048 | Z = 16624 | Gir. X = -891 | Y = 294 | Z = 36 | Temp = 30.84
Acel. X = 1432 | Y = -3020 | Z = 16808 | Gir. X = -921 | Y = 261 | Z = 47 | Temp = 30.88
Acel. X = 1388 | Y = -2972 | Z = 16816 | Gir. X = -880 | Y = 276 | Z = 39 | Temp = 30.88
Acel. X = 1348 | Y = -3096 | Z = 16744 | Gir. X = -915 | Y = 234 | Z = 52 | Temp = 30.79
Acel. X = 1204 | Y = -3008 | Z = 16688 | Gir. X = -889 | Y = 292 | Z = 50 | Temp = 30.74
Acel. X = 1260 | Y = -3040 | Z = 16604 | Gir. X = -876 | Y = 276 | Z = 45 | Temp = 30.79
Acel. X = 1260 | Y = -3124 | Z = 16364 | Gir. X = -858 | Y = 289 | Z = 66 | Temp = 30.84
Acel. X = 1444 | Y = -2820 | Z = 16548 | Gir. X = -896 | Y = 243 | Z = 83 | Temp = 30.79
Acel. X = 1312 | Y = -2884 | Z = 16788 | Gir. X = -896 | Y = 247 | Z = 43 | Temp = 30.84
Acel. X = 1364 | Y = -3072 | Z = 16692 | Gir. X = -898 | Y = 260 | Z = 49 | Temp = 30.74
```

Figura 38: Dados do Acelerômetro. Fonte: Autoria Própria

Na terceira coluna são mostrados os valores da aceleração no eixo Z,a uma frequência de 3.33 Hz ,fornecidos pelo sensor MPU-6050,os quais foram aproveitados propriamente no projeto(olhar figura 25 e 37). Após os dados do acelerômetro aparecem os dados do giroscópio, isto é, dados que são apresentados em função da rotação em torno dos eixo X ,Y e Z.

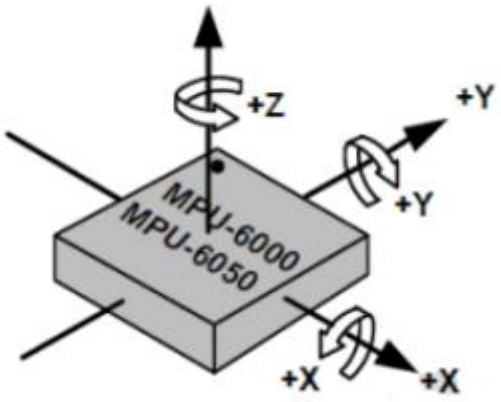

Figura 39: Orientação dos Eixos de Sensibilidade e Polaridade de Rotação. Fonte: [29].

Logo em seguida vem os valores de temperatura, em graus Celsius. O processamento que realiza a conversão de valores acontece de forma encapsulada no dispositivo acelerômetro. Nesta parte do trabalho foram aproveitados somente os valores fornecidos pelo sensor, que conta com um microprocessador interno.

4.2.2 Resultado da Análise

Observou-se que muitas abordagens de medição de rugosidade da superfície têm sido desenvolvidas. De acordo com Sayers [4], as abordagens podem ser agrupadas nas seguintes quatro classificações com base em quão diretamente as suas medidas dizem respeito à IRI [35]. ·

- Classe 4 - uma medida de rugosidade não é reproduzível ou estável com o tempo, e só pode ser comparada à IRI por avaliação subjetiva
- Classe 3 - uma medida obtida a partir de um RTRRMS é calibrado para a escala IRI por correlação com medidas de referência a partir de um sistema de classe 1 ou 2
- Classe 2 - um método baseado em perfil é utilizado. É reproduzível e estável com o tempo, e é calibrado de forma independente de outros instrumentos de medição de rugosidade.
- Classe 1 - um método baseado em perfil semelhante à classe 2 é usada. Uma medição baseada em perfil qualifica-se como uma medida de Classe 1 caso ela seja tão precisa que mais melhorias na precisão não seria aparente.

Portanto o perfilômetro desenvolvido visa utilizar como base a ideia da Classe2, reproduzível e estável com o tempo e é calibrado de forma independente

de outros instrumentos de medição de rugosidade. Em outras palavras, pode medir diretamente usando perfis de superfície e respostas dinâmicas do veículo. E dessa forma o desenvolvimento foi realizado.

Para a análise, um perfil teste foi executado para a velocidade de condução de 80 km/h. Essa velocidade foi escolhida tendo em consideração a adequação de medição tais como a consistência de condução velocidade e também por ser a velocidade limite permitida na rodovia adotada. Outro dado importante foi a adequação dos dados dentro da distância estabelecida. A seção de teste considerada foi de 40 km de comprimento em pavimentação asfáltica, onde os dados foram divididos em cinco partes de 8km. A seguir, os dados são apresentados juntamente com os respectivos trechos medidos.

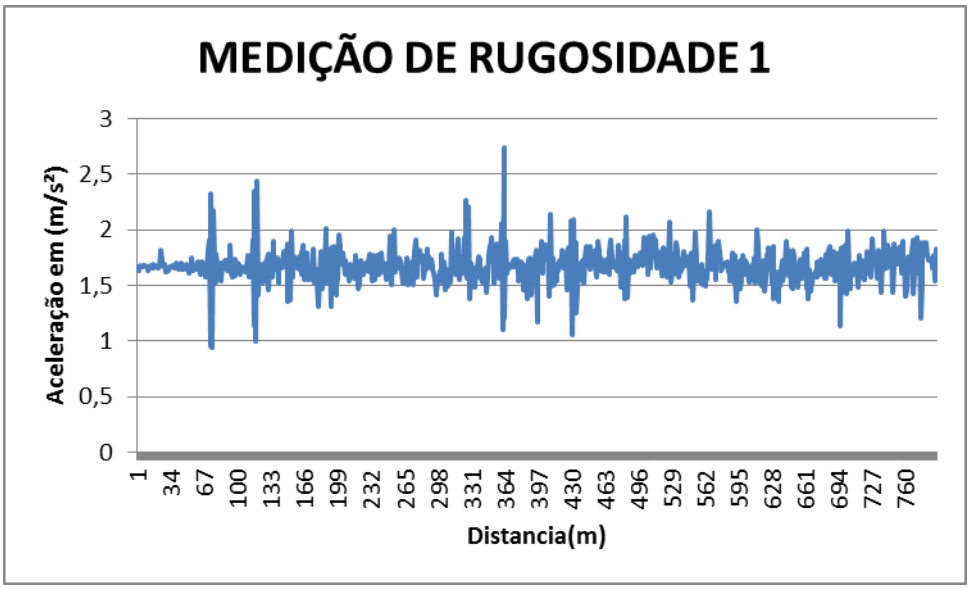

Figura 40: Medição de Rugosidade 1.

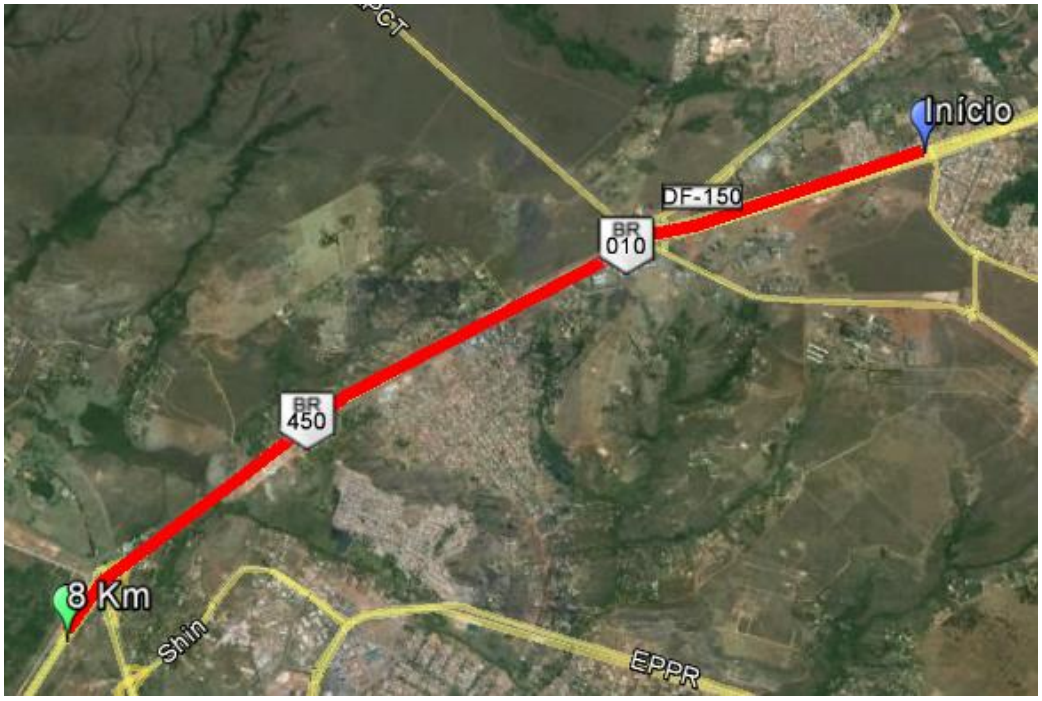

Figura 41: Dados do perfil medido(1)

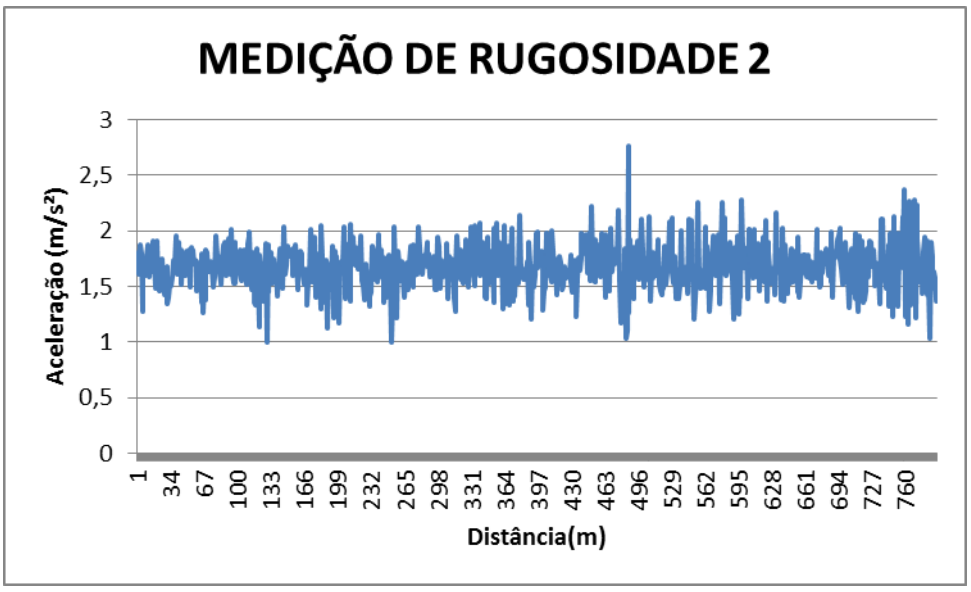

Figura 42: Medição de Rugosidade 2

Figura 43: Dados do perfil Medido (2)

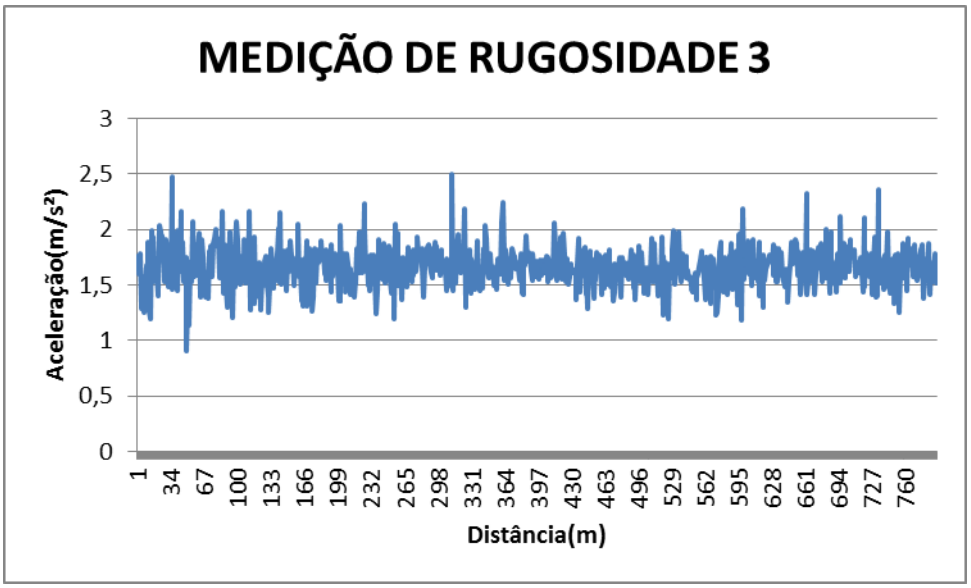

Figura 44: Medição de Rugosidades 3

Figura 45: Dados do Perfil Medido (3)

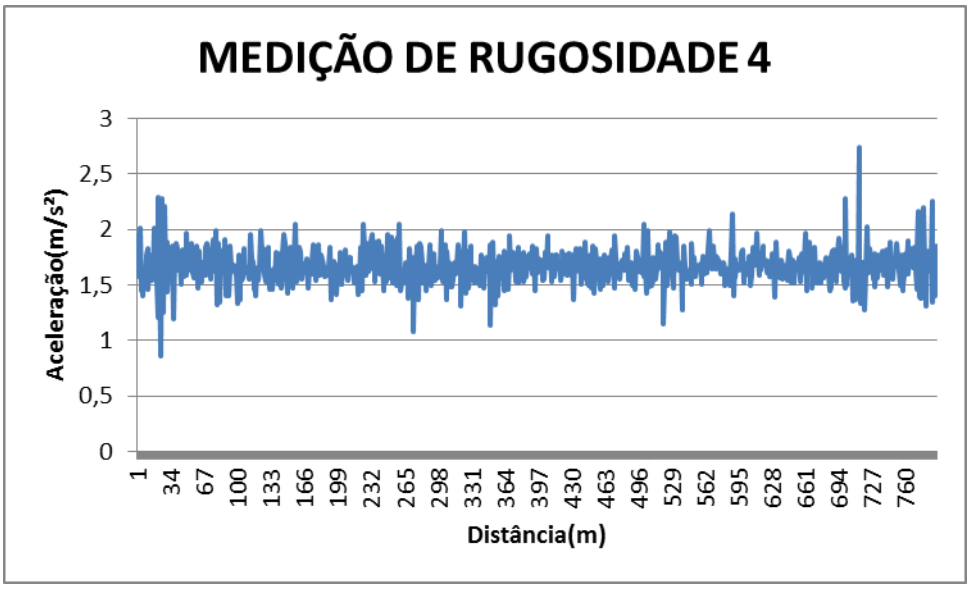

Figura 46: Medição de Rugosidade 4

Figura 47: Dados do Perfil Medido (4)

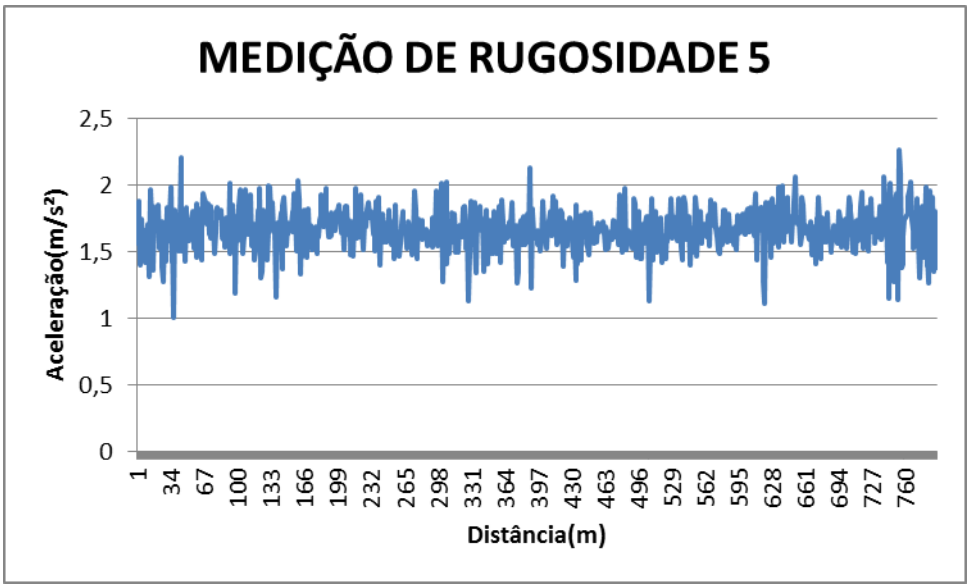

Figura 48: Medição de Rugosidade 5

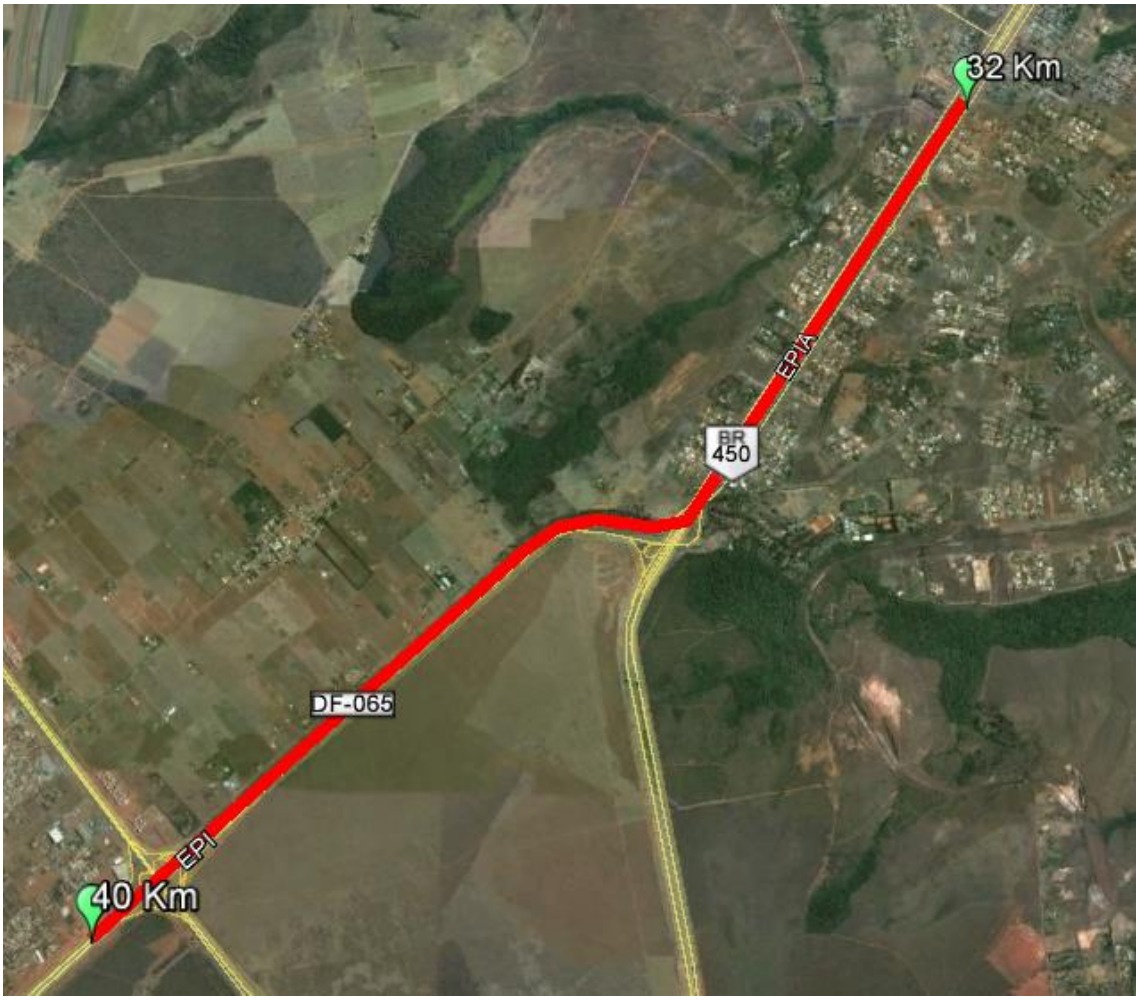

Figura 49: Dados do Perfil Medido (5)

5 CONCLUSÃO

Neste Trabalho de Conclusão de Curso foi descrito a implementação de um sistema de monitoramento de veículo e de superfície de estrada, por meio de um módulo SHIELD GPS /GPRS/ GSM –Arduino Uno- MPU 6050 acoplado em um veículo. Isso possibilitou não só a geolocalização do veículo como também permitiu, por meio do sensor acelerômetro, verificar os níveis de rugosidade da rodovia.

Foi observado nas literaturas estudadas um grande número de soluções para detecção de perturbações nos pavimentos rodoviários. Basicamente as soluções pesquisadas dedicam-se a gerência de mecanismos de compartilhamento de informações *on-line,* como em sites públicos, ou na captação dos índices gerados por sensores de varreduras óticos. Tais procedimentos são custosos para aplicação em larga escala e inconvenientes ao usuário final, pois exigem conhecimento especifico em sua utilização. E esse foi um dos motivos que impulsionaram a ideia de projetar um módulo.

Durante o desenvolvimento deste trabalho a resposta encontrada foi à utilização do sensor MPU-6050 trabalhando em conjunto com o SHIELD GPS/GPRS/GSM para o monitoramento de estradas e rodovias. Constatou-se, por meio de testes, vantajosa a união com um veículo, pois a vibração desses ao passar por algum trecho de perfil rugoso colabora consideravelmente com a captação dos dados para os sensores de aceleração. A vibração, por sua vez, provocada no veículo é efetivamente capturada por um acelerômetro integrado ao módulo. Dado um acelerômetro e um dispositivo GPS, é possível identificar o ponto determinado da estrada onde ocorreram maiores perturbações. Este método, porem, precisa perceber, armazenar e carregar toda a aceleração medida e os dados de GPS para um servidor central para processamento adicional. No entanto, a alta demanda por armazenamento e transmissão de dados continua a ser uma questão desafiadora.

O projeto apresentado ainda está em fase de desenvolvimento, porém demostrou bons resultados, sendo útil nas seguintes aplicações:

- Controle da posição de veículo por solicitação, para segurança de frotas rodoviárias;

- Mapeamento do índice de rugosidade de uma determinada via, contribuído com a identificação de buracos.

Ainda existem muitas melhorias que podem ser feitas a este projeto. Uma delas seria a utilização de múltiplos acelerômetros localizados no veículo. Dado que esse utiliza apenas um acelerômetro e a magnitude de saída é única, tendo em conta a vibração de apenas uma parte do carro. Para uma melhor leitura de saída, um acelerômetro em cada roda iria fornecer melhor informação para determinar a magnitude da vibração.

Outra melhoria seria carregar os dados para uma unidade flash USB. Isto simplificaria ainda mais o projeto. Se for utilizado um *flash drive*, em seguida, um cabo USB e um computador não seriam necessários para fazer o upload dos dados da memória. Com apenas o pressionar de um botão, os dados seriam salvos em um arquivo na unidade *flash*. Vale lembrar que para algumas aplicações o projeto necessita de adaptações. Uma das limitações atuais é o modo de transmissão e armazenamento dos dados coletados. O protótipo utiliza a cabo USB para transmissão das informações coletadas pelo módulo e armazena na memória do computador. Sendo necessária uma otimização também do sistema de transmissão.

BIBLIOGRAFIA

[1]: HEUBERGER, A. et al. Microelectronic Systems: Circuits, Systems and Applications. Germany, Springer, 2012, 388 p.

[2]: SINGH, B.; GUPTA, A. Recent trends in intelligent transportation systems: a review. Journal of Transport Literature, Índia, v. 9, n. 2, p. 1-34, 2015.

[3]: Gráfico 1: FROTA DE VEÍCULOS REGISTRADOS NO DISTRITO FEDERAL, Detran, 2015

[4] NOPIAH, Z. M. et al. Linear Programming: Optimization of Noise and Vibration Model in Passenger Car Cabin, International Journal of Soft Computing And Software Engineering, Malaysia, v. 2, n. 1, p. 1-13, 2012.

[5]: DOUANGPHACHANH, V.; ONEYAMA, H. A Study on the Use of Smartphones for Road Roughness Condition Estimation, Proceedings of the Eastern Asia Society for Transportation Studies, Tóquio, v. 9, p. 1-14, 2002.

[6]: DHAR, A. Traffic and Road Condition Monitoring System. Mumbai, 2008, 26 p.

[7]: DOMMETY, G.; JAIN, R. Potential Networking Applications of Global Positioning Systems. Department of Computer and Information Science. Ohio, 1998, 40 p.

[8]: MOHAMMAD, A. International Journal of Computer Science & Information Technology, Balqa, v. 3, n. 6, p. 1-11, 2011.

[9] Coelho, R. M. S.; Ribeiro, P. F. Sistemas de Posicionamento Global. Lisboa: Universidade Nova de Lisboa, 2006-2007, 17 p.

[10] Kaplan, E. D.; Hegarty, C. J. Understanding GPS: Principles and Applications. Second Edition. Londres: Artech House, 2005, 723 p.

[11]: Buracos proliferam, ainda que a Novacap assegure o recapeamento diário. Disponível em: <http://www.correiobraziliense.com.br/app/noticia/cidades/2015/03/24/interna_cidadesdf,476717/buracos-proliferam-ainda-que-a-novacap-assegure-o- recapeamento-diario.shtml> Acesso em: 24 mar. 2015

[12]: BORGES. et al. Embedded System for Detecting and Georeferencing Holes in Roads. Ministério da Ciência e Tecnologia, Conselho Nacional de Pesquisa, n.11, p.5, 2011.

[14] : GAIER, M. B. Aprendendo a Programar em Arduino. Instituto Federal de Educação Ciência e Tecnologia de Mato Grosso - Campos Cuiabá: PET Auto Net, 2011. 49p.

[15]: VASILJEVIC, G. Apostila de Arduino. Brasil: Autoria própria, 2013. 20p.

[16]: JUSTEN, Á. Curso de Arduino. cursodearuduino.com: Apostila do Aluno, 2014. 36p.

[17]: BEPPU. et al. Introdução ao Kit de Desenvolvimento Arduino. Niterói -RJ: Universidade Federal Fluminense, 2013. 80p.

[18]:DFRobot. Disponível em: <http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:TEL0051)> Acesso em: 15 jun. 2015

[19]: TECH, C. S. SIM908 AT Command Manual_V1.01. DFRobot: SIMCom, 2011. 249p.

[20]: CALACHE, D. C. Caracterização de um Acelerômetro Baseado em Sistemas Microeletromecânicos (MEMS). 2013. 90f. Monografia (Bacharelado em Tecnologia e Ciências) - Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

[21]: FORHAN, N. A. E. GIROSCÓPIOS MEMS. 2010. 84f. Monografia (Bacharelado em Ciência e Tecnologia) - INPE, São José dos Campos, 2010. Disponível em: <<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/01.25.18.42>> Acesso em: 26 jun. 2015

[22]: HALONEN, Timo; MELERO, Javier Romero And Juan. GSM, GPRS and Performance EDGE: Evolution Towards 3G/UMTS. 2. ed. Chichester: John Wiley & Sons Ltd, 2003. 656 p.

[23]: Stan, Arduino Uno and the InvenSense MPU6050 6DOF IMU. Disponível em: <http://42bots.com/tutorials/arduino-uno-and-the-invensense-mpu-6050-6dof-imu/>. Acesso em 03 de junho de 2015.

[25]: DFROBOT, GPS/GPRS/GSM Module V3.0. Disponível em:<http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:TEL0051)> Acesso em 04 de setembro de 2015.

[26]: HACKING, Diy. ARDUINO MPU 6050 – BEST IMU SENSOR. 2015. Disponível em: <http://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial/>. Acesso em: 12 out. 2015.

[27]: HYBRID GPS-GSM LOCALIZATION OF AUTOMOBILE TRACKING SYSTEM: Mohammad A. Al-Khedher. Amman, 6 dez. 2011.

[28]: ARDUINO. MPU-6050 Accelerometer + Gyro. Tutorial. Disponível em: <http://playground.arduino.cc/Main/MPU-6050#short>. Acesso em: 04 jul. 2015.

[29]: INVENSENSE. PS-MPU-6000A-00: MPU-6000 and MPU-6050 Product Specification. 3.2 ed. Boston, 2011. 57 p. Disponível em: <http://www.daedalus.ei.tum.de/attachments/article/57/PS-MPU-6000A.pdf>. Acesso em: 04 jul. 2015.

[30]: KUMAR, G.vijaya; SAI, Dr.y.padma; KUMAR, V.naveen. HAND GESTURE RECOGNITION USING ACCELEROMETER FOR DISABLED. International Journal Of Science: Engineering and Technology Research. Hyderabad, 12 dez. 2014. p. 3478-3481.

[31]: SOUZA, Fábio. Arduino: Interface com acelerômetro e giroscópio. 2015. Disponível em: <http://www.embarcados.com.br/arduino-acelerometro-giroscopio/>. Acesso em: 12 set. 2015.

[32]: BOTS, 42. Arduino Uno and the InvenSense MPU6050 6DOF IMU. 2014. Tutorial. Disponível em: <http://42bots.com/tutorials/arduino-uno-and-the-invensense-mpu-6050-6dof-imu/>. Acesso em: 04 jul. 2015

[33]: National Marine Electronics Association, “NMEA 0183 Standard For Interfacing Marine Electronic Devices,” Version 3.01, January , 2005.

[34]: SAYERS, Michael W.; GILLESPIE, Thomas D.; QUEIROZ, Cesar A. V.. The International Road Roughness Experiment: Establishing Correlation and a Calibration Standard for Measurements. Washuington: The World Bank, 1986. 464 p.

[35]: Tomiyama, K., Kawamura, A., Nakajima, S., Ishida, T., & Jomoto, M. A Mobile Profilometer For Road Surface Monitoring By Use Of Accelerometers. 2012.

[36]: Du, Y., Liu, C., Wu, D., & Jiang, S. (2014). Measurement of International Roughness Index by Using-Axis Accelerometers and GPS. Mathematical Problems in Engineering, 2014.

[37]: REN, Yanjun; WEN, Guanghua; LI, Xiuyun. An SVM Based Algorithm for Road Disease Detection using Accelerometer: Institute of Advanced Engineering and Science (IAES). Telkomnika. Indonésia, p. 5169-5175. 9 set. 2013.

[38]: PROJECTS, Geek Mom. MPU-6050: DMP Data from i2cdevlib. 2015. Disponível em: <http://www.geekmomprojects.com/mpu-6050-dmp-data-from-i2cdevlib/>. Acesso em: 12 out. 2015.

[39]: ODIN, Anthony. Arduino DIY: MPU 6050 Tutorial - Arduino Uno. 2015. Tutorial. Disponível em: <https://www.youtube.com/watch?v=n_f06DhCNNk>. Acesso em: 09 out. 2015.

[40] SYSTEM, S. Rastreamento de veículos via satélite. São Paulo - SP: Disponível em: <http://www.stopsystem.com.br/caminhao-ou-frotas.html> Acesso em 09 out 2015.

[41] MUNDOGEO. Novos satélites e controle do sistema GPS são testados. Disponível em: <http://mundogeo.com/blog/2012/04/19/novos-satelites-e-controle-do-sistema-gps-sao-testados/> Acesso em 09 out 2015.

[42] BARELLA, Rodrigo Maluf. Contribuição para a Avaliação da Irregularidade Longitudinal de Pavimento com Perfilômetros Inerciais. 2008. 362 f.

Tese (Doutorado) - Curso de Engenharia, Engenharia de Transportes, Escola Politécnica da Universidade de São Paulo, São Paulo, 2008.

ANEXOS

I.1 CÓDIGO PARA GERAR DADOS DE POSICIONAMENTO GLOBAL

```
// # Passos:
// # 1. Coloque a chave S1 para o lado  Prog (lado direito)
// # 2. Coloque o interruptor S2 para o lado USB (lado esquerdo)
// # 3.Coloque a chave seletora UART para no meio.
// # 4. Carregar o esboço para a placa Arduino
// # 5. Coloque a chave S1 para o lado  comm (lado esquerdo)
// # 6. RST a placa até o STAT led ficar aceso
void setup() {
// init the driver pins for gsm function
pinMode(3,OUTPUT); // INICIALIZA O PINO DIGITAL COMO SAIDA DE DADOS
pinMode(4,OUTPUT);// INICIALIZA O PINO DIGITAL COMO SAIDA DE DADOS
pinMode(5,OUTPUT);// INICIALIZA O PINO DIGITAL COMO SAIDA DE DADOS
//OUTPUT GSM TIMING
digitalWrite(5,HIGH); //LIGA
delay(1500); // ESPERA 1,5 SEGUNDOS
digitalWrite(5,LOW); //DESLIGA
}
void loop() {
// put your main code here, to run repeatedly:
digitalWrite(3,HIGH);//disable GSM tx,rx
digitalWrite(4,HIGH);//disable GPS tx,rx
```

```
}
```

I.2 CÓDIGO PARA GERAR DADOS DE POSICIONAMENTO GLOBAL E ENVIA-LOS VIA SMS

```
//ADAPTADO DO SITE http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:TEL0051)

#include "gps_gsm_sim908Serial0.h"

void setup () {

gps_init ();    //INICIA PINO GPS

Serial.begin (9600);    //serial0 CONECTARO COMPUTADOR

start_gps ();   //INICIA GPS

}

void loop () {

int stat = gps_get_gga ();  // LÊ DADOS DO GPS, SE RETORNAR 0 OK

if (stat == 0 || stat == 1) {

if (gps_gga_is_fix ()) {

gsm_set_numble ("6194120182"); // MUDAR PARA MEU NUMERO DE CELULAR

gsm_send_message (gps_gga_utc_s ());

gsm_send_message (gps_gga_EW ());

gsm_send_message (gps_gga_NS ());

gsm_send_message (gps_gga_lat_s ());

gsm_send_message (gps_gga_long_s ());

gsm_end_send ();

while (1);

}

}
```

```
switch (stat) {
case 0:
#ifdef DEBUG
Serial.println ("data checksum is ok");
#endif
break;
case 1:
#ifdef DEBUG
Serial.println ("GPGGA ID is error!");
#endif
break;
case 2:
#ifdef DEBUG
Serial.println ("data is error!");
#endif
break;
}

#ifdef DEBUG
Serial.println ("$GPGGA data:");
gps_gga_print ();   //PARA TESTAR
#endif
/*
if (gps_gga_is_fix () == 0) //check if is fix
Serial.println ("can't fix! please go outside!");
```

```
else {
Serial.println ("ok! is fix!");

Serial.println ("gps_gga_utc_hh ()");
Serial.println (gps_gga_utc_hh ());
Serial.println ("gps_gga_utc_mm ()");
Serial.println (gps_gga_utc_mm ());
Serial.println ("gps_gga_utc_ss ()");
Serial.println (gps_gga_utc_ss ());

Serial.println ("gps_gga_NS ()");
Serial.println (gps_gga_NS (), 6);
Serial.println ("gps_gga_EW ()");
Serial.println (gps_gga_EW (), 6);

Serial.println ("gps_gga_lat ()");
Serial.println (gps_gga_lat (), 6);
Serial.println ("gps_gga_long ()");
Serial.println (gps_gga_long (), 6);
Serial.println ("gps_gga_HDOP ()");
Serial.println (gps_gga_HDOP (), 6);
Serial.println ("gps_gga_MSL ()");
Serial.println (gps_gga_MSL (), 6);
Serial.println ();
}
*/
```

```
}
```

I.3 BIBLIOTECA UTILIZADA PARA CONSTRUÇÃO DO CÓDIGO PERMITE A CAPTURA E ENVIO VIA SMS DE DADOS GPS

```
/********************* start of gps_gsm_sim908.h ***********************/

/*

*  by 2013-08-02

*  test on UNO

*  Serial0 to GPS

*

*/


//debug

//#define DEBUG


#include <Arduino.h>

#define gps_enable()    digitalWrite (4, LOW)

#define gps_disable()   digitalWrite (4, HIGH)

#define gsm_enable()    digitalWrite (3, LOW)

#define gsm_disable()   digitalWrite (3, HIGH)

#define GPS_BUF_SIZE 500

#define GGA_NUM 15

#define RMC_NUM 14

//

char *gga_table[GGA_NUM] = {

"Message ID",          //0

"UTC Time",            //1
```

```
"Latitude",             //2
"N/S Indicator",        //3
"Longitude",            //4
"E/W Indicator",        //5
"Position Fix Indicator",   //6
"Satellites Used",      //7
"HDOP",             //8
"MSL Altitude",         //9
"Units(M)",             //10
"Geoid Separation",         //11
"Units",            //12
"Diff.Ref.Station ID",      //13
"Checksum",         //14
};
//
char *gprmc_table[RMC_NUM] = {
"Message ID",           //0
"UTC Time",                 //1
"Status",                   //2
"Latitude",                 //3
"N/S Indicator",            //4
"Langitude",                //5
"E/W Indicator",            //6
"Speed Over Ground",            //7
"Course Over Ground",           //8
"Date",                     //9
```

```
"Magnetic Variation",           //10
"East/West Indicator",          //11
"Mode",                         //12
"Checksum",                     //13
}

//save data from GPS
uint8_t gps_buf[GPS_BUF_SIZE];

//save pointer of gga block
uint8_t* gga_p[GGA_NUM];
uint8_t* gprmc_p[RMC_NUM];

// check sum using xor
uint8_t checksum_xor (uint8_t *array, uint8_t leng) {
uint8_t sum = array[0];
for (uint8_t i=1; i<leng; i++) {
sum ^= array[i];
}
return sum;
}


//
void start_gps () {
digitalWrite (5, HIGH);
```

```
delay (1500);
digitalWrite (5, LOW);
delay (1500);

gsm_enable ();
gps_disable ();

delay (2000);
#ifdef DEBUG
Serial.println ("waiting for GPS! ");
#endif

Serial.println ("AT");
#ifdef DEBUG
Serial.println ("Send AT");
#endif
delay (1000);
Serial.println ("AT+CGPSPWR=1");
#ifdef DEBUG
Serial.println ("Send AT+CGPSPWR=1");
#endif
delay (1000);
Serial.println ("AT+CGPSRST=1");
#ifdef DEBUG
Serial.println ("Send AT+CGPSRST=1");
#endif
```

```
delay (1000);

gsm_disable ();
gps_enable ();

delay (2000);
#ifdef DEBUG
Serial.println ("$GPGGA statement information: ");
#endif
}

// read data to gps_buf[] from GPS
static int gps_read () {
uint32_t start_time = millis ();
while (!Serial.available ()) {
if (millis() - start_time > 1500) {
#ifdef DEBUG
Serial.println ("restart GPS.....");
#endif
start_gps ();
}
}
for (int i=0; i<GPS_BUF_SIZE; i++) {
delay (7);
if (Serial.available ()) {
gps_buf [i] = Serial.read ();
```

```
} else {
#ifdef DEBUG
Serial.print ("read ");
Serial.print (i);
Serial.println (" character");
#endif
return 1;
}
}
#ifdef DEBUG
Serial.println ("error! data is so big!");
#endif
return 0;
}

//test head of gps_buf[] if is "$GPGGA" or not
static int is_GPGGA () {
char gga_id[7] = "$GPGGA";
for (int i=0; i<6; i++)
if (gga_id[i] != gps_buf[i])
return 0;
return 1;
}

//
static uint8_t get_gga_leng () {
```

```
uint8_t l;
for (l=0; l<GPS_BUF_SIZE && gps_buf[l] != 0x0d ; l++);
return l;
}

// build gga_p[] by gps_buf
static void build_gga_p () {
int p,b;
for (p=b=0; p<GGA_NUM && b<GPS_BUF_SIZE; p++,b++) {
gga_p[p] = (gps_buf+b);//
if (gps_buf[b] == ',')
continue;
for (b++; b<GPS_BUF_SIZE && gps_buf[b]!=','; b++);
}
}

//test if fix
int gps_gga_is_fix (void) {
if (gga_p[6][0] == '1')
return 1;
else
return 0;
}

//get gga checksum
static uint8_t gps_gga_checksum () {
```

```
uint8_t sum = 0;
if (gga_p[14][0] != '*')
return 0;
if (gga_p[14][2] >= '0' && gga_p[14][2] <= '9')
sum = gga_p[14][2] - '0';
else
sum = gga_p[14][2] - 'A' + 10;
if (gga_p[14][1] >= '0' && gga_p[14][1] <= '9')
sum += (gga_p[14][1] - '0') * 16;
else
sum += (gga_p[14][1] - 'A' + 10) * 16;
return sum;
}

//check sum of gga
static int checksum_gga () {
uint8_t sum = checksum_xor (gps_buf+1, get_gga_leng ()-4);
return sum - gps_gga_checksum ();
}

// set gga, change ',' to '\0'
static void gps_gga_set_str () {
int i;
for (i=0; gps_buf[i] != 0x0d && i<GPS_BUF_SIZE; i++)
if (gps_buf[i] == ',')
gps_buf[i] = '\0';
```

```
//gps_buf[i] = '\0';
}

//
int gps_get_gga (void) {
int stat = 0;
if (gps_read ()) {
if (is_GPGGA ()) {
build_gga_p (); // build *gga_p[] by gps_buf
 gps_gga_set_str ();
if (checksum_gga () == 0)
stat = 0;
else
stat = 1;
} else
stat = 2;
} else
stat = 3;

return stat;
}


//get UTC second
uint8_t gps_gga_utc_ss () {
return (gga_p[1][4]-'0')*10+gga_p[1][5]-'0';
```

```
}

//get UTC minute
uint8_t gps_gga_utc_mm () {
return (gga_p[1][2]-'0')*10+gga_p[1][3]-'0';
}

//get UTC hour
uint8_t gps_gga_utc_hh () {
return (gga_p[1][0]-'0')*10+gga_p[1][1]-'0';
}

//return UTC Time string, hhmmss
char* gps_gga_utc_s () {
return (char*)gga_p[1];
}

//get latitude
double gps_gga_lat () {
return atof ((char*)gga_p[2]);
}

//get latitude
char* gps_gga_lat_s () {
return (char*)gga_p[2];
}
```

```
//get longitude
double gps_gga_long () {
return atof ((char*)gga_p[4]);
}

//get longitude
char* gps_gga_long_s () {
return (char*)gga_p[4];
}

//get HDOP
double gps_gga_HDOP () {
return atof ((char*)gga_p[8]);
}

//get HDOP
char* gps_gga_HDOP_s () {
return (char*)gga_p[8];
}

//get N/S
char* gps_gga_NS () {
return (char*)gga_p[3];
/*
if (gga_p[3][0] == '\0')
```

```
return '0';
else if (gga_p[3][0] == 'N' || gga_p[3][0] == 'S')
return gga_p[3][0];
else
return '?';
*/
}

//get E/W
char* gps_gga_EW () {
return (char*)gga_p[5];
/*
if (gga_p[5][0] == '\0')
return '0';
else if (gga_p[5][0] == 'E' || gga_p[5][0] == 'W')
return gga_p[5][0];
else
return '?';
*/
}

//
double gps_gga_MSL () {
return atof ((char*)gga_p[9]);
}
```

```
//
char* gps_gga_MSL_s () {
return (char*)gga_p[9];
}

//get gpggpa Geoid Separation
double gps_gga_geoid_sep () {
return atof ((char*)gga_p[11]);
}

//get gpggpa Geoid Separation
char* gps_gga_geoid_sep_s () {
return (char*)gga_p[11];
}

#ifdef DEBUG
//
void gps_gga_print () {
for (int i=0; i<GPS_BUF_SIZE && gps_buf[i]!=0xd; i++)
Serial.print ((char)gps_buf[i]);
Serial.println ();
}
#endif

/*
void send_string (char* numble, char*string) {
```

```
char num_buf[25];
sprintf (num_buf, "AT+CMGS=\"%s\"", numble);
gsm_enable ();
gps_disable ();
delay (2000);
Serial.println ("AT");
delay (200);
Serial.println ("AT");
delay (200);
Serial.println ("AT+CMGF=1");
delay (200);
Serial.println (num_buf);
delay (200);
Serial.println (string);
Serial.write (26);
}
*/

// set mobile numble
void gsm_set_numble (char *numble) {
char num_buf[25];
sprintf (num_buf, "AT+CMGS=\"%s\"", numble);
gsm_enable ();
gps_disable ();
delay (2000);
Serial.println ("AT");
```

```
delay (200);
Serial.println ("AT");
delay (200);
Serial.println ("AT+CMGF=1");
delay (200);
Serial.println (num_buf);
delay (200);
}

// send message to mobile
void gsm_send_message (char *message) {
Serial.println (message);
}
//
void gsm_end_send () {
Serial.write (26);
delay (200);
gsm_disable ();
gps_enable ();
delay (2000);
}

//
void gps_init () {
pinMode (3, OUTPUT);
pinMode (4, OUTPUT);
```

```
pinMode (5, OUTPUT);

}

/********************* end of gps_gsm_sim908.h **********************/
```

I.4 CALIBRAÇÃO DO SENSOR MPU-6050 UTILIZANDO A IDE DO ARDUINO

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//       2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//          2012-06-20 - improved FIFO overflow handling and simplified read process
//          2012-06-19 - completely rearranged DMP initialization code and simplification
//          2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
```

// 2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING

// 2012-06-05 - add gravity-compensated initial reference frame acceleration output

// - add 3D math helper file to DMP6 example sketch

// - add Euler output and Yaw/Pitch/Roll output formats

// 2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)

// 2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250

// 2012-05-30 - basic DMP initialization working

/* ==

I2Cdev device library code is placed under the MIT license

Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

==========

*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files

// for both classes must be in the include path of your project

#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"

//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation

// is used in I2Cdev.h

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

  #include "Wire.h"

#endif

```
        // class default I2C address is 0x68
        // specific I2C addresses may be passed as a parameter here
        // AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
        // AD0 high = 0x69
        MPU6050 mpu;
        //MPU6050 mpu(0x69); // <-- use for AD0 high

        /* =========================================================================
           NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
           depends on the MPU-6050's INT pin being connected to the Arduino's
           external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
           digital I/O pin 2.
         * ========================================================================= */

        /* =========================================================================
           NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
           when using Serial.write(buf, len). The Teapot output uses this method.
           The solution requires a modification to the Arduino USBAPI.h file, which
           is fortunately simple, but annoying. This will be fixed in the next IDE
           release. For more info, see these links:
```

```
    http://arduino.cc/forum/index.php/topic,109987.0.html
    http://code.google.com/p/arduino/issues/detail?id=958
 *
=========================================================================== */

    // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
    // quaternion components in a [w, x, y, z] format (not best for parsing
    // on a remote host such as Processing or something though)
    //#define OUTPUT_READABLE_QUATERNION

    // uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
    // (in degrees) calculated from the quaternions coming from the FIFO.
    // Note that Euler angles suffer from gimbal lock (for more info, see
    // http://en.wikipedia.org/wiki/Gimbal_lock)
    //#define OUTPUT_READABLE_EULER

    // uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
    // pitch/roll angles (in degrees) calculated from the quaternions coming
    // from the FIFO. Note this also requires gravity vector calculations.
    // Also note that yaw/pitch/roll angles suffer from gimbal lock (for
    // more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
```

```
//#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT



#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;
```

```
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };
```

```
        //
================================================================
        // ===               INTERRUPT DETECTION ROUTINE                ===
        //
================================================================

        volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has
gone high
        void dmpDataReady() {
            mpuInterrupt = true;
        }




        //
================================================================
        // ===                      INITIAL SETUP                       ===
        //
================================================================

        void setup() {
            // join I2C bus (I2Cdev library doesn't do this automatically)
            #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
                Wire.begin();
                TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
            #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
                Fastwire::setup(400, true);
```

```
#endif

// initialize serial communication
// (115200 chosen because it is required for Teapot Demo output, but it's
// really up to you depending on your project)
Serial.begin(115200);
while (!Serial); // wait for Leonardo enumeration, others continue immediately

// NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
// Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
// the baud timing being too misaligned with processor ticks. You must use
// 38400 or slower in these cases, or use some kind of external separate
// crystal solution for the UART timer.

// initialize device
Serial.println(F("Initializing I2C devices..."));
mpu.initialize();

// verify connection
Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

// wait for ready
```

```
        Serial.println(F("\nSend any character to begin DMP programming and demo: "));
        while (Serial.available() && Serial.read()); // empty buffer
        while (!Serial.available());                 // wait for data
        while (Serial.available() && Serial.read()); // empty buffer again

        // load and configure the DMP
        Serial.println(F("Initializing DMP..."));
        devStatus = mpu.dmpInitialize();

        // supply your own gyro offsets here, scaled for min sensitivity
        mpu.setXGyroOffset(220);
        mpu.setYGyroOffset(76);
        mpu.setZGyroOffset(-85);
        mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

        // make sure it worked (returns 0 if so)
        if (devStatus == 0) {
            // turn on the DMP, now that it's ready
            Serial.println(F("Enabling DMP..."));
            mpu.setDMPEnabled(true);

            // enable Arduino interrupt detection
            Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
            attachInterrupt(0, dmpDataReady, RISING);
```

```
            mpuIntStatus = mpu.getIntStatus();

            // set our DMP Ready flag so the main loop() function knows it's okay to
use it
            Serial.println(F("DMP ready! Waiting for first interrupt..."));
            dmpReady = true;

            // get expected DMP packet size for later comparison
            packetSize = mpu.dmpGetFIFOPacketSize();
        } else {
            // ERROR!
            // 1 = initial memory load failed
            // 2 = DMP configuration updates failed
            // (if it's going to break, usually the code will be 1)
            Serial.print(F("DMP Initialization failed (code "));
            Serial.print(devStatus);
            Serial.println(F(")"));
        }

        // configure LED for output
        pinMode(LED_PIN, OUTPUT);
    }
```

```
        //
================================================================
        // ===                MAIN PROGRAM LOOP                ===
        //
================================================================

        void loop() {
            // if programming failed, don't try to do anything
            if (!dmpReady) return;

            // wait for MPU interrupt or extra packet(s) available
            while (!mpuInterrupt && fifoCount < packetSize) {
                // other program behavior stuff here
                // .
                // .
                // .
                // if you are really paranoid you can frequently test in between other
                // stuff to see if mpuInterrupt is true, and if so, "break;" from the
                // while() loop to immediately process the MPU data
                // .
                // .
                // .
            }

            // reset interrupt flag and get INT_STATUS byte
            mpuInterrupt = false;
```

```
        mpuIntStatus = mpu.getIntStatus();

        // get current FIFO count
        fifoCount = mpu.getFIFOCount();

        // check for overflow (this should never happen unless our code is too inefficient)
        if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
            // reset so we can continue cleanly
            mpu.resetFIFO();
            Serial.println(F("FIFO overflow!"));

        // otherwise, check for DMP data ready interrupt (this should happen frequently)
        } else if (mpuIntStatus & 0x02) {
            // wait for correct available data length, should be a VERY short wait
            while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

            // read a packet from FIFO
            mpu.getFIFOBytes(fifoBuffer, packetSize);

            // track FIFO count here in case there is > 1 packet available
            // (this lets us immediately read more without waiting for an interrupt)
            fifoCount -= packetSize;

            #ifdef OUTPUT_READABLE_QUATERNION
```

```
    // display quaternion values in easy matrix form: w x y z
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    Serial.print("quat\t");
    Serial.print(q.w);
    Serial.print("\t");
    Serial.print(q.x);
    Serial.print("\t");
    Serial.print(q.y);
    Serial.print("\t");
    Serial.println(q.z);
#endif

#ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
```

```
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(ypr[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
```

```
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t");
    Serial.print(aaWorld.x);
    Serial.print("\t");
    Serial.print(aaWorld.y);
    Serial.print("\t");
    Serial.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0];
    teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4];
    teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8];
    teapotPacket[7] = fifoBuffer[9];
    teapotPacket[8] = fifoBuffer[12];
    teapotPacket[9] = fifoBuffer[13];
    Serial.write(teapotPacket, 14);
```

```
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

I.5 CALIBRAÇÃO DO SENSOR MPU-6050 UTILIZANDO A IDE DO PROCESSING

```
// I2C device class (I2Cdev) demonstration Processing sketch for MPU6050 DMP output
// 6/20/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     2012-06-20 - initial release

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

==========

*/

```
import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;

// NOTE: requires ToxicLibs to be installed in order to run properly.
// 1. Download from http://toxiclibs.org/downloads
// 2. Extract into [userdir]/Processing/libraries
//    (location may be different on Mac/Linux)
// 3. Run and bask in awesomeness

ToxiclibsSupport gfx;

Serial port;                         // The serial port
char[] teapotPacket = new char[14];  // InvenSense Teapot packet
int serialCount = 0;                 // current packet byte position
int synced = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];
```

```
void setup() {
    // 300px square viewport using OpenGL rendering
    size(300, 300, OPENGL);
    gfx = new ToxiclibsSupport(this);

    // setup lights and antialiasing
    lights();
    smooth();

    // display serial port list for debugging/clarity
    println(Serial.list());

    // get the first available port (use EITHER this OR the specific port code below)
    //String portName = Serial.list()[0];

    // get a specific serial port (use EITHER this OR the first-available code above)
    String portName = "COM4";

    // open the serial port
    port = new Serial(this, portName, 115200);

    // send single character to trigger DMP init/start
    // (expected by MPU6050_DMP6 example Arduino sketch)
    port.write('r');
```

```
}

void draw() {
    if (millis() - interval > 1000) {
        // resend single character to trigger DMP init/start
        // in case the MPU is halted/reset while applet is running
        port.write('r');
        interval = millis();
    }

    // black background
    background(0);

    // translate everything to the middle of the viewport
    pushMatrix();
    translate(width / 2, height / 2);

    // 3-step rotation from yaw/pitch/roll angles (gimbal lock!)
    // ...and other weirdness I haven't figured out yet
    //rotateY(-ypr[0]);
    //rotateZ(-ypr[1]);
    //rotateX(-ypr[2]);

    // toxiclibs direct angle/axis rotation from quaternion (NO gimbal lock!)
    // (axis order [1, 3, 2] and inversion [-1, +1, +1] is a consequence of
    // different coordinate system orientation assumptions between Processing
```

```
            // and InvenSense DMP)
            float[] axis = quat.toAxisAngle();
            rotate(axis[0], -axis[1], axis[3], axis[2]);

            // draw main body in red
            fill(255, 0, 0, 200);
            box(10, 10, 200);

            // draw front-facing tip in blue
            fill(0, 0, 255, 200);
            pushMatrix();
            translate(0, 0, -120);
            rotateX(PI/2);
            drawCylinder(0, 20, 20, 8);
            popMatrix();

            // draw wings and tail fin in green
            fill(0, 255, 0, 200);
            beginShape(TRIANGLES);
            vertex(-100,  2, 30); vertex(0,  2, -80); vertex(100,  2, 30);  // wing top layer
            vertex(-100, -2, 30); vertex(0, -2, -80); vertex(100, -2, 30);  // wing bottom
layer
            vertex(-2, 0, 98); vertex(-2, -30, 98); vertex(-2, 0, 70);  // tail left layer
            vertex( 2, 0, 98); vertex( 2, -30, 98); vertex( 2, 0, 70);  // tail right layer
            endShape();
            beginShape(QUADS);
```

```
        vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -
80);
        vertex( 100, 2, 30); vertex( 100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -
80);
        vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(100, -2,  30); vertex(100, 2,
30);
        vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2, -30, 98); vertex(-2, -30, 98);
        vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
        vertex(-2, -30, 98); vertex(2, -30, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
        endShape();

        popMatrix();
    }

    void serialEvent(Serial port) {
        interval = millis();
        while (port.available() > 0) {
            int ch = port.read();

            if (synced == 0 && ch != '$') return;   // initial synchronization - also used
to resync/realign if needed
            synced = 1;
            print ((char)ch);

            if ((serialCount == 1 && ch != 2)
                || (serialCount == 12 && ch != '\r')
                || (serialCount == 13 && ch != '\n'))  {
```

```
    serialCount = 0;
    synced = 0;
    return;
}

if (serialCount > 0 || ch == '$') {
    teapotPacket[serialCount++] = (char)ch;
    if (serialCount == 14) {
        serialCount = 0; // restart packet byte position

        // get quaternion from data packet
        q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
        q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
        q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
        q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
        for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];

        // set our toxilibs quaternion to new data
        quat.set(q[0], q[1], q[2], q[3]);

        /*
        // below calculations unnecessary for orientation only using toxilibs

        // calculate gravity vector
        gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
        gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
```

```
                gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];

                // calculate Euler angles
                euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
                euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1], 2*q[0]*q[0] + 2*q[3]*q[3] - 1);

                // calculate yaw/pitch/roll angles
                ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] + gravity[2]*gravity[2]));
                ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] + gravity[2]*gravity[2]));

                // output various components for debugging
                //println("q:\t" + round(q[0]*100.0f)/100.0f + "\t" + round(q[1]*100.0f)/100.0f + "\t" + round(q[2]*100.0f)/100.0f + "\t" + round(q[3]*100.0f)/100.0f);
                //println("euler:\t" + euler[0]*180.0f/PI + "\t" + euler[1]*180.0f/PI + "\t" + euler[2]*180.0f/PI);
                //println("ypr:\t" + ypr[0]*180.0f/PI + "\t" + ypr[1]*180.0f/PI + "\t" + ypr[2]*180.0f/PI);
                */
            }
        }
    }
```

```
}

void drawCylinder(float topRadius, float bottomRadius, float tall, int sides) {
    float angle = 0;
    float angleIncrement = TWO_PI / sides;
    beginShape(QUAD_STRIP);
    for (int i = 0; i < sides + 1; ++i) {
        vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
        vertex(bottomRadius*cos(angle), tall, bottomRadius*sin(angle));
        angle += angleIncrement;
    }
    endShape();

    // If it is not a cone, draw the circular top cap
    if (topRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, 0, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(topRadius * cos(angle), 0, topRadius * sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }
```

```
    // If it is not a cone, draw the circular bottom cap
    if (bottomRadius != 0) {
      angle = 0;
      beginShape(TRIANGLE_FAN);

      // Center point
      vertex(0, tall, 0);
      for (int i = 0; i < sides + 1; i++) {
        vertex(bottomRadius * cos(angle), tall, bottomRadius * sin(angle));
        angle += angleIncrement;
      }
      endShape();
    }}
```

I.6 AQUISIÇÃO DE DADOS DO ACELERÔMETRO

```
//Programa : Teste MPU-6050
//Alteracoes e adaptacoes : Arduino e Cia
//
//Baseado no programa original de JohnChi
//Carrega a biblioteca Wire
#include<Wire.h>
//Endereco I2C do MPU6050
const int MPU=0x68;
```

```
//Variaveis para armazenar valores dos sensores
int AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

void setup()
{
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);

  //Inicializa o MPU-6050
  Wire.write(0);
  Wire.endTransmission(true);
}

void loop()
{
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  //Solicita os dados do sensor
  Wire.requestFrom(MPU,14,true);

//Armazena o valor dos sensores nas variaveis correspondentes
  AcX=Wire.read()<<8|Wire.read();  //0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
```

```
    AcY=Wire.read()<<8|Wire.read();  //0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
    AcZ=Wire.read()<<8|Wire.read();  //0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
    Tmp=Wire.read()<<8|Wire.read();  //0x41 (TEMP_OUT_H) & 0x42
(TEMP_OUT_L)
    GyX=Wire.read()<<8|Wire.read();  //0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
    GyY=Wire.read()<<8|Wire.read();  //0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
    GyZ=Wire.read()<<8|Wire.read();  //0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)
    //Mostra os valores na serial
    Serial.print("Acel. X = "); Serial.print(AcX);
    Serial.print(" | Y = "); Serial.print(AcY);
    Serial.print(" | Z = "); Serial.print(AcZ);
    Serial.print(" | Gir. X = "); Serial.print(GyX);
    Serial.print(" | Y = "); Serial.print(GyY);
    Serial.print(" | Z = "); Serial.print(GyZ);
    Serial.print(" | Temp = "); Serial.println(Tmp/340.00+36.53);

    //Aguarda 300 ms e reinicia o processo
    delay(300);
  }
```

A. DATASHEET/ESQUEMA ELÉTRICO

A.1 ARDUINO UNO

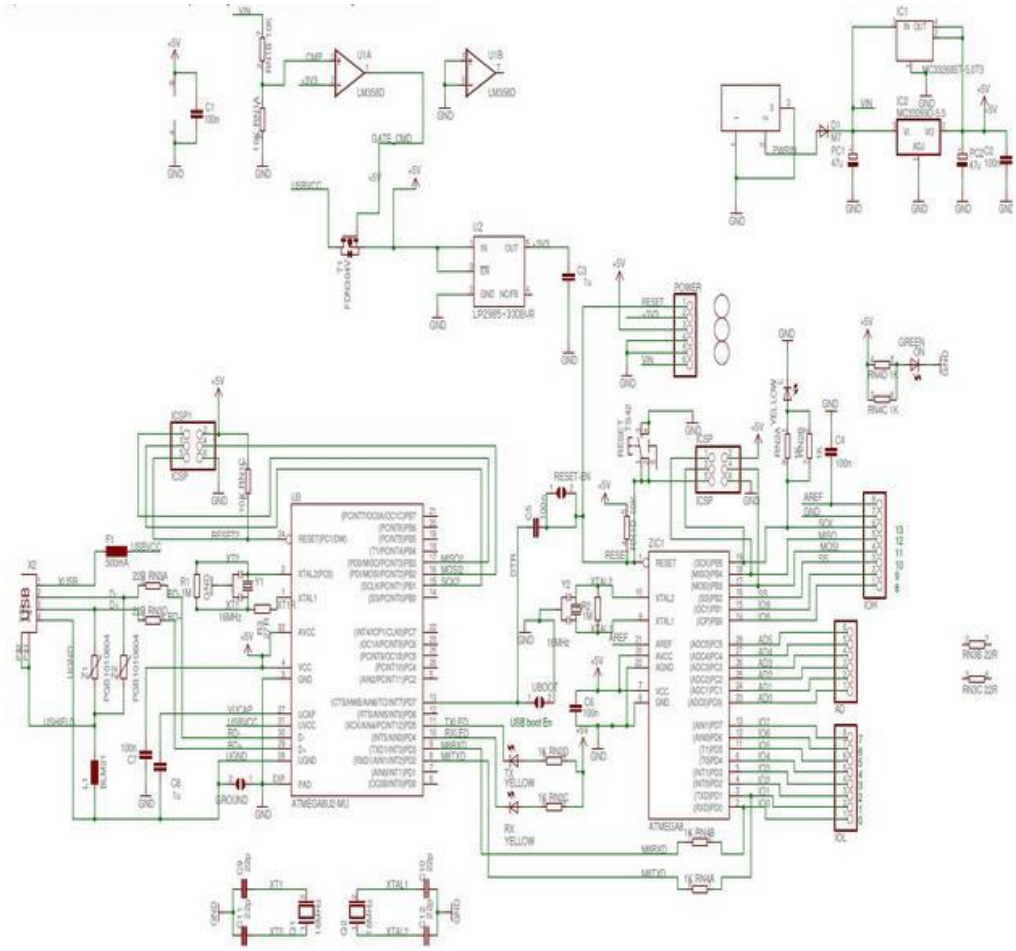

Figura 50: Esquemático Arduino

A.2 SHIELD GPS/GPRS/GSM

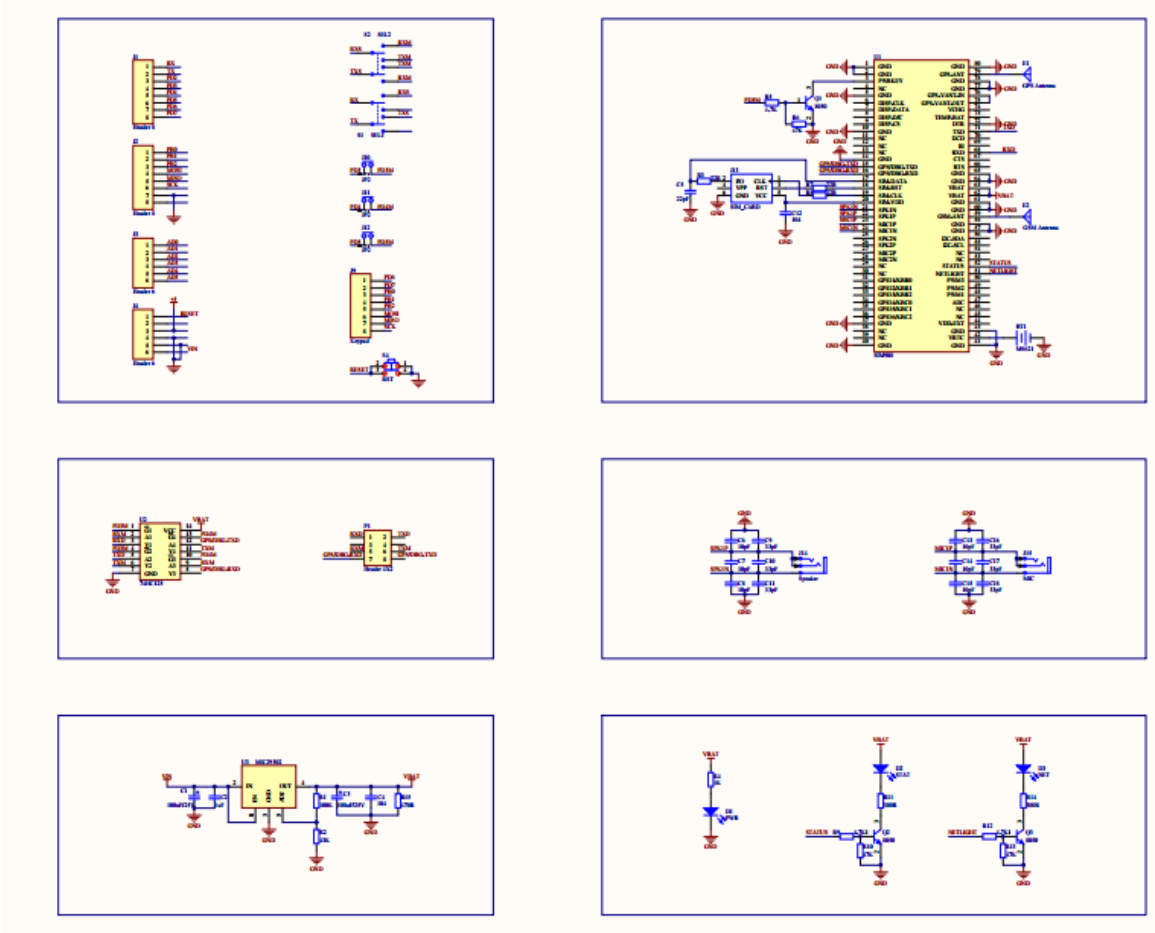

Figura 51: Esquemático Shield GPS/GPRS/GSM

A.3 MPU-6050

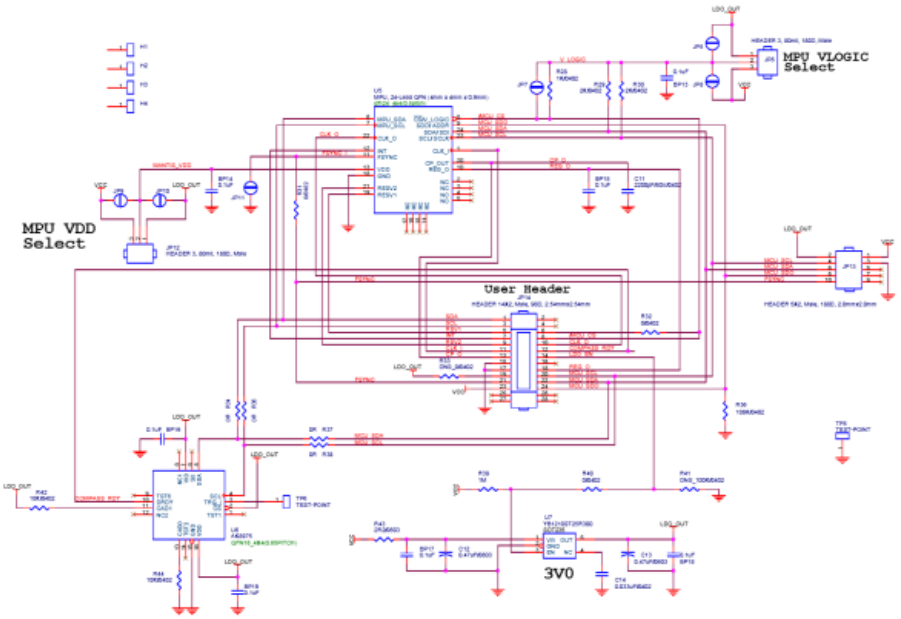

Figura 52: Esquemático do Sensor MPU-6050