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

Projeto de um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem

Autor: Jennifer Gladys Pereira Cavalcante
Orientador: Prof. Dr. Gerardo Antonio Idrobo Pizo

Brasília, DF
2021

Jennifer Gladys Pereira Cavalcante

Projeto de um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem

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).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Gerardo Antonio Idrobo Pizo

Brasília, DF

2021

Jennifer Gladys Pereira Cavalcante

Projeto de um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem/ Jennifer Gladys Pereira Cavalcante. – Brasília, DF, 2021-

75 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Gerardo Antonio Idrobo Pizo

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2021.

1. caixa preta para carros. I. Prof. Dr. Gerardo Antonio Idrobo Pizo. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Projeto de um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem

CDU 02:141:005.6

Jennifer Gladys Pereira Cavalcante

Projeto de um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem

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).

Prof. Dr. Gerardo Antonio Idrobo Pizo
Orientador

Prof. Dr. Rudi Henri Van Els
Convidado 1

Prof. Dr. Daniel M. Muñoz Arboleda
Convidado 2

Brasília, DF
2021

Este trabalho é dedicado aos meus pais e ao meu irmão, que estiveram comigo e não mediram esforços para que eu estudasse e chegasse nessa etapa. Obrigada por acreditarem e por estarem comigo sempre.

Agradecimentos

Agradeço primeiramente a Jeová, por toda a estabilidade para poder enfrentar todas as dificuldades encontradas até aqui. Aos meus pais e irmão, que foram primordiais em todos os aspectos durante minha graduação, dando-me apoio e não deixando que houvesse uma possibilidade de desistência. Às minhas amigas Lilian e Luana, que me acompanham desde o ensino médio e sempre acreditaram que seria possível, que riram e choraram comigo, aos meus amigos Antônio Lucas, Mateus, Pedro Guilherme, Yasmin, Maria Cristina, Victor Batalha, Amauri Jr. e todos os colegas, familiares que foram de forma direta ou indiretamente, fundamentais para que eu chegasse até aqui. Meu orientador Gerardo, por toda paciência em todos esses anos e, principalmente agora no desenvolvimento deste projeto, aos outros professores que contribuíram para o meu aprendizado. À Universidade de Brasília pela oportunidade de estudo de qualidade.

"Não abandone um sonho só porque vai demorar a acontecer.
O tempo vai passar do mesmo jeito."
(H. Jackson Brown)

Resumo

A caixa-preta de um avião é parte fundamental para entender o que levou aquele acidente a acontecer e como ele ocorreu. Com ela é possível obter dados de momentos antes do acidente, gravando o áudio da cabine, onde se capta o microfone do piloto, copiloto e todo o som do ambiente, além de uma outra que memoriza os parâmetros do voo, como velocidade do avião, posição dos manetes e o momento em que determinados botões foram acionados. Com base na ideia da caixa-preta presente em aviões, existem equipamentos similares voltados para carros, com o intuito de diminuir os acidentes automobilísticos, e identificar os responsáveis de eventuais acidentes, visto que em muitos deles não há sobreviventes. Existem alguns carros com esse dispositivo, mas são voltados para um determinado público, que possuem utilitários de luxo. O objetivo deste trabalho é o desenvolvimento de um sistema de registro de dados automotivos de baixo custo que possa ser parte integrante de qualquer veículo e não apenas de uma pequena parcela. Será feito o registro de dados automotivos através de sensores já presentes viabilizando o menor custo, visto que não será necessário fazer alguma alteração em sua parte eletrônica do veículo. Esse sistema fará o registro de dados do sensor de temperatura, sensor de rotação do motor e sensor de velocidade, buscando encontrar uma forma de utilizar esses dados de maneira a ajudar a entender como o evento ocorreu. Além dos dados coletados através do OBDII, foi utilizado um sistema empregando microcontroladores e um módulo GPS para coleta de dados de velocidade, direção e localização, como forma de registro menos invasiva dos sistemas para coleta dos dados. Um sistema não exclui o outro e ambos serão utilizados para o registros. Os dados obtidos com o sistema externo, serão armazenados em um banco de dados externo via conexão *Wi-Fi*.

Palavras-chaves: Eletrônica Embarcada, Sistema Automotivo,Baixo Custo, ESP32, GPS-neo6MV2, Arduíno Uno.

Abstract

The black box of an airplane is a fundamental part of understanding what caused that accident to happen and how it happened. With it, it is possible to obtain data from moments before the accident, recording the cabin audio, where the pilot and copilot microphone and all the ambient sound are captured, as well as another one that memorizes the flight parameters such as airplane speed, position of the levers and the moment when certain buttons were activated. Based on the idea of the black box present in planes, there is similar equipment aimed at cars, with the aim of reducing automobile accidents, and identifying those responsible for any accidents, since in many of them there are no survivors. There are some cars with this device, but they are aimed at a certain audience, which have luxury SUVs. The objective of this work is the development of a low cost automotive data recording system that can be an integral part of any vehicle and not just a small portion. The registration of automotive data will be done through sensors already present, enabling the lowest cost, since it will not be necessary to make any changes to your vehicle's electronics. This system will record data from the temperature sensor, engine speed sensor and speed sensor, seeking to find a way to use this data in order to help understand how the event occurred. In addition to the data collected through OBDII, a system employing microcontrollers and a GPS module was used to collect speed, direction and location data, as a less invasive way of recording the systems for data collection. One system does not exclude the other and both will be used for registration. The data obtained with the external system will be stored in an external database via Wi-Fi connection.

Key-words: Embedded Electronics, Automotive System, Low Cost, ESP32, GPS-neo6MV2, Arduino Uno.

Lista de abreviaturas e siglas

ABS	*Antilock Braking System*
ACC	*Adaptive Cruise Control*
DER	Departamento de Estradas de Rodagem
DNIT	Departamento Nacional de Infraestrutura de Transportes
ECU	Unidade de Controle Eletrônico
ESP	*Eletronic Stability Program*
GRU	Aeroporto Internacional de Guarulhos
OBD	*On Board Diagnostics*
OMS	Organização Mundial da Saúde
RPM	*Revolutions per minute*
SVS	Secretaria de Vigilância em saúde
IoT	Internet das Coisas
PWM	*Pulse Width Modulation*
USB	*Universal Serial Bus*
GPS	Sistema de Posicionamento Global
JSON	*JavaScript Object Notation*
UART	*Universal Asynchronous Receiver / Transmitter*

Sumário

Introdução

Segundo dados da Secretaria de Vigilância em Saúde (SVS) acidentes de trânsitos estão entre as dez principais causas de mortes no Brasil (1). A Organização Mundial da Saúde (OMS) ainda diz que acidentes de trânsito matam cada vez mais pessoas em todo o planeta, com cerca de 1,35 milhão de óbitos registrados em 2018. Ressaltando assim a importância de se reforçar as condições de segurança dos veículos(2). As causas desses acidentes são várias, mas em sua maioria trata-se de falhas humanas e situações em que o condutor, coloca-se em perigo e também ao próximo. Para tentar diminuir esses índices, o Departamento Nacional de Infraestrutura de Transportes (DNIT), responsável por vias administradas pela União, e o Departamento de Estradas de Rodagem (DER), responsável por vias estaduais e municipais, têm buscado investir em instalações de radares com o intuito de diminuir os acidentes, que em períodos de férias aumentam significativamente, devido ao aumento do tráfego, tendo como principal fator, na maioria dos acidentes, o abuso da velocidade (3). Há também o monitoramento das vias por câmeras e até por *drones*, que visam fiscalizar os motoristas, podendo interceptá-los em casos de má conduta.

As montadoras também fazem a sua parte, investindo cada vez mais em segurança, seja com *airbags*, que antes estavam disponíveis apenas para motorista e passageiro dianteiro, e atualmente já é possível encontrar carros onde há esse dispositivo para passageiros do banco traseiro ou com freios *ABS*, que auxiliam em uma melhor frenagem sem que haja o travamento das rodas. Foram citados dois exemplos de métodos de segurança que tem feito a diferença na vida de motoristas e passageiros, porém acidentes continuam frequentes.

Objetivando a segurança, fabricantes de automóveis têm se empenhado mais, investindo em modelos que se assemelham a caixas-pretas, que funcionam através do módulo do *airbag*. Assim que o capô começa a deformar, o sensor emite o sinal para o computador de bordo abrir o *airbag* dando início ao registro de dados. Exemplos de dados monitorados pelo módulo são a velocidade nos últimos segundos antes do impacto, a curva de desaceleração, e a situação do afivelamento do cinto de segurança. Dependendo da informação recebida, o *airbag* deve inflar com maior ou menor intensidade. Esses módulos vêm instalados no console central e as informações obtidas variam de acordo com cada montadora e as especificações desejadas. Esse modelo de caixa-preta hoje está disponível apenas para poucos modelos de carros de luxo. De acordo com Jacson Birai(4), há relatos da caixa preta em automóveis e fez-se presente em 1974, como um protótipo experimental em mais de 1.000 carros. Mas para sua implementação e para que carros passassem a sair de fábricas com tal equipamento, levou mais de 20 anos.

Aspirando contribuir com a redução dos acidentes, com a compreensão dos fatores que levaram ao acidente ocorrer e, da situação durante o impacto, este trabalho visa desenvolver um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem, cujo objetivo é coletar e armazenar dados durante a viagem, não tendo como intenção a construção de uma caixa-preta em si. Há trabalhos nesse âmbito, como o trabalho do Jacson Birai (4) cujo projeto estuda o desenvolvimento e os aspectos físicos de uma caixa-preta automotiva, que guarda dados da velocidade, ângulo do volante e pedais acionados. Seguindo esse meio de estudo, na esfera do registro e armazenamento em nuvem, intencionando a compreensão das circunstâncias da colisão, com foco nos dados da velocidade do automóvel, localização em tempo real via GPS e tempo de condução.

Definição do problema

O estudo de um sistema de registro de dados automotivos de baixo custo, pretende desenvolver um protótipo que seja viável para qualquer carro, utilizando um sistema eletrônico que não tenha alto valor agregado.

O intuito deste projeto é fazer o registro dos dados de velocidade, localização, tempo e de alguns sensores presentes no automóvel tais como RPM e líquido de arrefecimento. Dessa forma, a finalidade é guardar os dados para que posteriormente agentes especializados possam usá-los para elucidar possíveis causas do acidente, como falha humana, imprudência ou falha mecânica.

Os dados coletados referentes à velocidade, à localização e, ao sentido do trajeto, serão enviados para um banco de dados na nuvem podendo ser acessados por celulares com acesso à internet ou através de um computador.

Objetivos Gerais

Realizar o registro de dados automotivos como auxílio do *scanner* ELM327 (9) com o qual pode-se obter dados dos sensores de temperatura do líquido de arrefecimento e rotação do motor. Combinado com aquisição de dados obtidos através do módulo GPS gy-GPS6mv2 (12), esses registros serão armazenados em um banco de dados na nuvem.

Objetivos Específicos

- Apresentar base teórica para aquisição dos dados necessários para o registro.
- Apresentar uma ferramenta funcional capaz de realizar a captação de dados.
- Apresentar uma base de dados viável para o registro dos dados.
- Apresentar os resultados levantados por intermédio do sistema e sua viabilização.

1 Revisão Bibliográfica

Nesta seção será abordada a mecânica básica do automóvel, alguns sensores presentes no carro, além dos que foram utilizados para o registro de dados, visto que a maioria trabalha em conjunto, protocolo de leitura desses sensores e itens de segurança que são obrigatórios de fábrica.

1.1 Sistemas de segurança presentes em um automóvel

Atualmente é comum ver carros autômatos que são considerados sinônimos de segurança. Todavia assim que tais carros surgiram, seus faróis eram itens opcionais. Surgindo em 1898, eram feitos de acetileno, uma chama resistente a chuva e vento, não podendo ser de filamentos elétricos, visto que as condições das pistas naquela época eram precárias, rompendo os filamentos. Os faróis mais próximos do que conhecemos hoje, só começaram a aparecer em 1908.

O item mais associado à segurança é o cinto de segurança que é, responsável por evitar que o condutor e seus passageiros em caso de colisão, sejam arremessados ou colidam com o painel. Outro item muito comum é o *airbag*, e apesar de só ter sido considerado item obrigatório de fábrica em 2014, trata-se de um sistema de proteção passiva, pois atua imediatamente após um acidente, visando reduzir o impacto dos passageiros (5). Há também outros métodos de segurança presentes nos automóveis, como o *Adaptive Cruise Control* , controle de cruzeiro adaptativo ou piloto automático. Nessa função o motorista pode ajustar a que velocidade o veículo deve se locomover, sem a necessidade de manter pressionado o pedal do acelerador. Carros com a função *ACC*, podem ser automaticamente desligados e os freios *ABS* acionados, caso o sensor do sistema instalado à frente do carro detecte uma frenagem e a aproximação de um outro veículo (5).

Juntamente com o *ACC*, o freio *ABS* (*Antilock Braking System*), que é nativo de fábrica, é um dos sistemas de segurança ativa existentes em um veículo, um sistema de freio antitravamento. Em conjunto com os freios *ABS*, o sistema *ESP* (*Eletronic Stability Program*), que compõe um conjunto de sensores é, um sensor instalado no volante de direção e indica para onde o veículo deve ser direcionado; outro sensor fica localizado no chassi do veículo, sendo responsável por indicar as forças de movimentação atuantes no veículo (forças de giro e de aceleração lateral e longitudinal). Com os dados, o sistema consegue identificar as diferenças entre a direção solicitada pelo motorista e a real direção do carro (5).

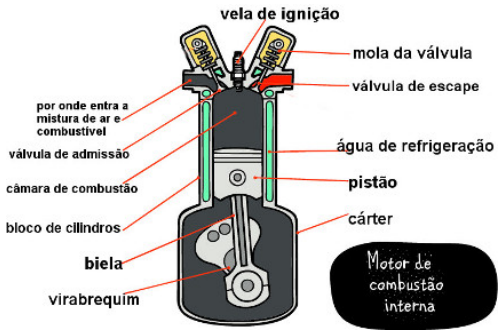
Dados os registros de componentes de segurança em um veículo, vale notar que eles

não estão amplamente distribuídos em todos os automóveis e a maioria dos dispositivos são para proteger os motoristas em casos de acidentes, mas são poucos voltados para prevenção.

1.2 Motor

O motor é a fonte de energia do carro. Ele converte a energia da combustão proveniente da queima da gasolina em energia mecânica, o que faz o carro se movimentar. Quando ocorre a partida do motor de um veículo, alimenta-se o chamado motor de partida, um motor elétrico, utilizado para iniciar o ciclo de trabalho do motor à combustão. Com o movimento do motor, o alternador passa a ser operado e a gerar energia, que é utilizada para alimentar os sistemas eletroeletrônicos do veículo.

Figura 1 – Motor de Combustão interna



Fonte: (6)

- Motor de ciclo Otto: Utiliza a energia da centelha elétrica da vela de ignição para dar início a reação de combustão. Nos motores de quatro tempos ocorre a mistura de ar e combustível, já no de dois tempos, ocorre a mistura de ar, combustível e óleo lubrificante (7, p.2).
- Motor de ciclo Diesel: "Os motores do ciclo diesel ou motores de ignição por compressão utilizam o aumento da temperatura devido a compressão de uma massa de ar para dar início a reação de combustão. Somente ar é admito (7, p.6)".

1.2.1 Sensor de rotação do motor

O sensor de RPM é o responsável por informações como rotação instantânea do virabrequim e a posição do mesmo. Informações necessárias para que a unidade de comando do motor saiba o momento exato da injeção eletrônica começar o seu trabalho, além do tempo, tanto para motores de ciclo Otto quanto Diesel. Trata-se de um sensor vital para o funcionamento do motor, sem essa informação o motor não funciona.

Figura 2 – Sensor RPM

Fonte: (8)

1.2.2 Sensor de temperatura

Um dos principais sistemas de arrefecimento do carro, sendo sua função analisar a temperatura do motor e do fluido em circulação enquanto está ocorrendo a combustão. Essa informação é transmitida para o módulo de controle eletrônico do veículo. É formado por um termistor cerâmico, que apresenta variação em sua resistência elétrica quando submetido a mudanças de temperatura, assim quando o calor do sistema aumenta, a resistência do sensor diminui. Com alteração da resistência do sensor, ocorre variação na tensão, que permite a identificação da temperatura exata do motor.

Figura 3 – Sensor de temperatura de arrefecimento

Fonte: (9)

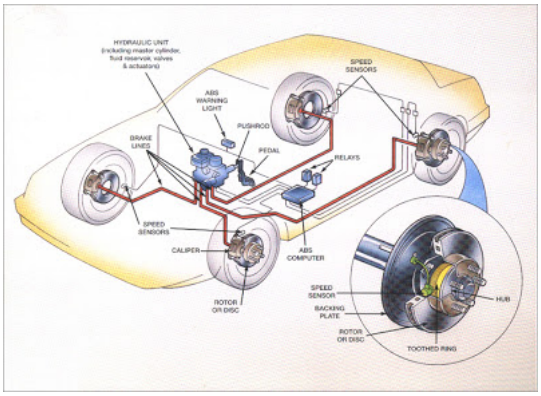
1.3 Outros Sensores presentes em um veículo

Nos últimos anos, os automóveis vêm passando por constantes mudanças, com a necessidade de seguranças, conforto e estética, algo que não era imaginado nos séculos

passados. Para que essas características fossem incorporadas, foi necessário que os veículos passassem por uma revolução automotiva e ocorresse também a inserção de diversos módulos eletrônicos no controle do carro. Dos controles existentes, alguns serão de grande importância para a implementação da Caixa-Preta. "*A injeção eletrônica é um sistema de alimentação de combustível e de gerenciamento eletrônico do motor à combustão de um automóvel (4)*". A injeção eletrônica é a encarregada pela economia de combustível, pois faz com que o motor trabalhe em condições adequadas, e também por fazer a leitura de vários sensores presentes no carro. Os sensores são dispositivos eletrônicos que monitoram as várias grandezas físicas e químicas de um carro, transformando em informação para os módulos eletrônicos responsáveis por cada sistemas embarcados. Cada sensor trabalha como um coletor de dados, tendo cada um comparado com padrões já estabelecidos. De acordo com o resultado, será gerado avisos, podendo ser sonoros, luminosos no painel do carro ou uma gravação na central eletrônica (10).

Além dos já citados sensores de temperatura de líquido de arrefecimento e de rotação do motor, o carro ainda conta com outros sensores importantes que irão colaborar no desenvolvimento do estudo.

Figura 4 – Sistema de freios ABS e seus componentes



Fonte: (11)

Sensor de freios ABS: Responsável por evitar o travamento das rodas em uma frenagem mais intensa, evitando o descontrole do veículo.

Sensor de velocidade: Fornece um sinal em forma de onda que é enviado para a central da injeção eletrônica.

Sonda lambda: Envia um sinal elétrico à injeção eletrônica do veículo, possibilitando a o controle de combustível que será enviado para o motor, por indicar a presença de oxigênio nos gases de escape.

Sensor de temperatura do ar do coletor: Fornece à central a temperatura do ar que está entrando no motor. Junto com o sensor de pressão, a central consegue calcu-

lar a massa admitido pelo motor e assim determinar a quantidade de combustível necessária para a combustão completa.

Sensor do airbag: São acelerômetros que medem a desaceleração do veículo e junto com o módulo eletrônico que irá processar as informações recebidas pelo sensor que enviará o comando para o acionamento das bolsas infláveis.

Sensor de posição do acelerador: Informa a central a posição instantânea do pedal, permitindo a central identificar a potência em que o condutor está requerendo do motor.

Sensor de posição do volante: A função do sensor é medir a posição do volante através do ângulo de rotação, sendo possível determinar as direções enviadas ao comando de posicionamento das rodas.

Com os sensores presentes no veículo é possível construir a caixa preta, reduzindo custos, visto que só será necessário fazer as leituras dos mesmos. Para o estudo será feito a leitura de sensores presentes no motor: sensor de RPM e sensor de temperatura do líquido de arrefecimento. Além do uso de sistema composto por microcontroladores, para auxílio de obtenção de dados referentes a localização, velocidade, sentido.

1.4 Computador de Bordo

Computador de bordo presente na maioria dos carros disponíveis atualmente no mercado brasileiro, conta com essa tecnologia. Muitos não entendem seu funcionamento, mas trata-se de um grande aliado do motorista por conter informações tais como dados de consumo, autonomia do tanque disponível, quilometragem, temperatura do motor ou se o carro apresenta alguma anomalia. Por estar conectado aos sensores do carro e a sua central eletrônica, o computador de bordo é capaz de captar dados sensíveis e importantes para o carro, sem que o motorista se preocupe em tirar as mãos do volante. O computador de bordo permite, através do sistema operacional do fabricante, que o usuário instale aplicativos de desenvolvedores para melhorar sua experiência, como GPS, aplicativos para reprodução de músicas, entre outros.

1.5 Sensores utilizados para leitura dos dados e registro em nuvem

A leitura dos dados ocorrerá através dos dados binários recebidos pelos sensores do motor relacionado ao RPM e à temperatura. Com o advento da tecnologia no mercado automobilístico é possível encontrar meios de comunicação que buscam facilitar o monitoramento e tornar o carro mais inteligente. O caso da rede *CAN bus*, uma forma

de comunicação dos equipamentos entre e de modo rápido, que podem atuar e controlar com segurança determinadas tarefas do sistema (4).

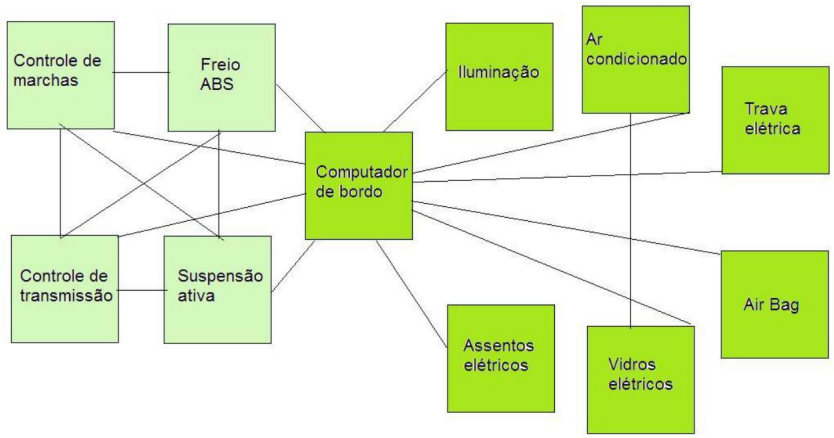
1.6 Rede CAN Bus

O *CAN bus* ou barramento *Controller Area Network* foi disponibilizado na década de 80 pela empresa alemã *Bosch*, inicialmente desenvolvida para implementação em ônibus e caminhões e hoje amplamente incorporada na indústria automobilística, náutica e outras. O *CAN* é um protocolo de comunicação serial síncrono (5).

Cada módulo recebe a mensagem lançada ao barramento em que os *bits* comunicam-se sem haver o problema de perda. Os módulos conseguem trabalhar na função de multimestre, na qual todos eles podem trabalhar como mestres ou escravos, além de ser possível enviar mensagens para todos os módulos existentes na rede. Um ponto importante nesse protocolo é a possibilidade dos módulos verificarem o estado de barramento de outro módulo, sendo possível saber se o tal módulo está enviando ou não mensagens com prioridade. Caso seja percebido, o módulo que tem a mensagem de menor prioridade tem sua transmissão encerrada e o de maior prioridade continua enviando a sua mensagem.

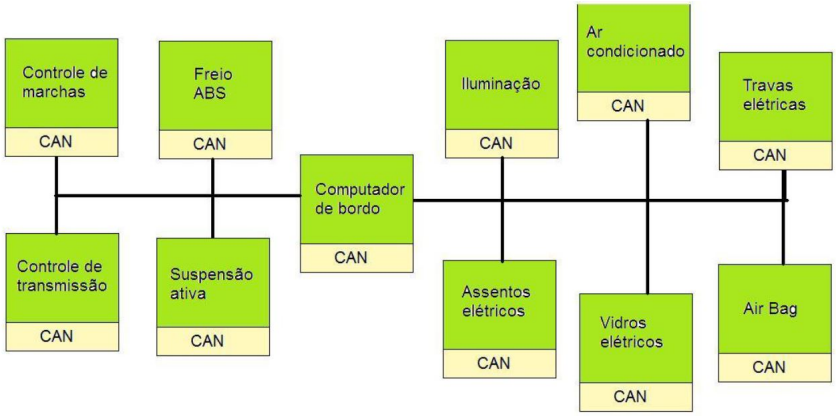
Essa flexibilidade permite que vários sistemas sejam aplicados quando for necessária a comunicação com outros equipamentos, atribuindo agilidade ao processo. Além disso, ocorre a diminuição da quantidade de fios no veículo, visto que por causa da complexidade dos sistemas de controle e com a necessidade de trocas de informações rápidas eram necessários cada vez mais cabos ligados ao chicote do carro, tendo ainda o aumento do custo para fazer essas ligações.

Figura 5 – Sistema Convencional de comunicação entre os sensores



Fonte: (4, p.25)

Figura 6 – Sistema Convencional de comunicação entre os sensores



Fonte: (4, p.25)

Como mostrado nas imagens acima (figura 5 e figura 6), no sistema convencional é necessário que todos os sensores estejam conectados entre si, diferente do sistema *CAN* que necessita apenas que estejam conectados a uma central.

A caixa-preta teria acesso a essas informações, sem a necessidade de novas conexões com os sensores, além de diminuir a quantidade de cabos e aumentar a velocidade de obtenção dos dados.

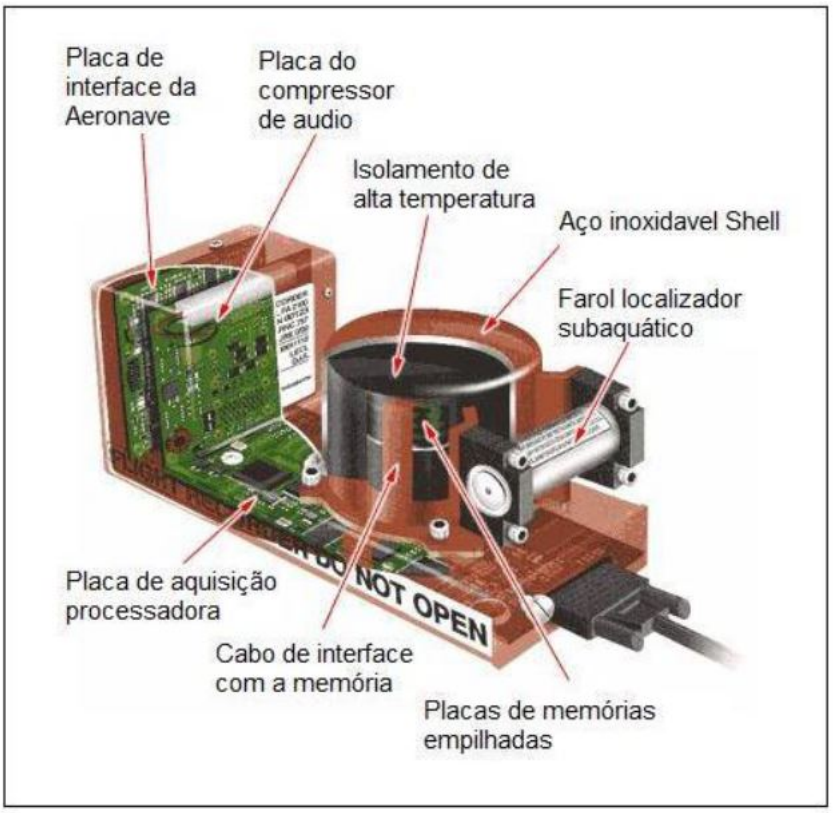
1.7 Caixa-Preta em Aviões

Para o estudo de uma caixa-preta em automóveis, é necessário compreender como funciona uma caixa-preta de aviões, visto que é a partir dela que surgiu a ideia de implementar uma em um carro. Ela é constituída por dois dispositivos de gravação, um registra apenas o áudio da cabine e o outro registra dados do voo. Trata-se de um dispositivo ultra-resistente composto por titânio e aço, além de uma espécie que se assemelha a uma esponja responsável por proteger todos os componentes internos do calor.

Pesando apenas 4,5 quilos, é capaz de suportar um impacto de mais de 3 mil vezes esse peso. É resistente por uma hora a uma temperatura de 1.100 graus celsius e a 10 horas a uma temperatura de 260 graus celsius. Sua posição no avião é tão importante quanto a sua blindagem para conservação dos seus dados. A caixa-preta fica localizada na cauda do avião, que comprovadamente é normalmente o último local a sofrer um impacto da queda.

Em casos de submersão, a caixa preta é resistente a uma pressão de 6 mil metros de profundidade, assim que ocorre o contato com a água, é disparo um sensor localizado ao lado de fora que emite um alarme sonoro que pode ser ouvido a mais de 4 mil metros (12).

Figura 7 – Caixa Preta do Avião



Fonte: (4, p.27)

1.7.1 Análise dos Dados da Caixa-Preta

Assim que as caixas pretas são recuperadas, elas são levadas para laboratórios especializados para que seus dados gravados sejam analisados para que seja entender o que levou a ocorrer acidente, visto que ali contém todos os momentos que antecederam a queda.

Partindo do princípio de que a caixa-preta foi encontrada em perfeito estado e seus componentes internos não sofreram nenhuma avaria, é utilizado um sistema que funciona como um *scanner*, que irá fazer a leitura e extrair os dados da caixa preta.

Em caso de danos internos, retira-se as memórias internas e com o auxílio de um cabo de interface com memória é instalado (4).

O leitor de dados da caixa preta possui um software que facilita a recuperação dos dados. Com as informações, uma equipe especializada é chamada para interpretar os dados.

Para o modelo proposto nesse trabalho será usado um *scanner* para leitura dos sensores presentes no veículo, além de um sistema composto por dois microcontroladores que combinados irão captar dados do módulo GPS e enviar para um banco de dados na nuvem, para evitar que em caso de acidentes, ocorra a perda dos dados. O registro e o

envio de dados será explicado no capítulo da metodologia, em que serão apresentados os componentes e como se integram.

1.7.2 Modelos Existentes

Durante as pesquisas realizadas para o desenvolvimento deste trabalho não foi encontrado nenhum produto que se assemelha à ideia proposta. Os produtos que foram encontrados integram uma proposta de caixa preta, mas sem a resistência do material encontrado na caixa-preta presente em aviões. É empregado apenas o conceito de registro de dados, sendo esse um registro local, correndo o risco de perda desses dados, em caso de danos mais graves ao acessório.

Além do modelo Super *Box* de caixa preta citada anteriormente, fabricada pela *IAPA* em 2009 (4), no mercado ainda é possível encontrar o modelo fabricado pela *Thinkware*, uma grande empresa de tecnologia sul-coreana, que teve início de comercialização no Brasil em 2014.

Dos modelos oferecidos pela *Thinkware*, a *Clair2* é um modelo intermediário, com câmera frontal de 2MP. memória de 16GB, com o opcional de GPS externo. O *FXD900* é um modelo mais robusto, com câmera frontal *Full* HD, e uma câmera traseira HD, com memória de 32GB, antena GPS embutida (13). A caixa-preta é de fácil instalação, sendo acoplada atrás do espelho retrovisor. Ela é acionada assim que o carro é ligado e também quando ocorre impactos, independentemente se o carro estiver em movimento ou parado (13).

1.7.3 Considerações Finais

Torna-se necessária a implementação da caixa-preta em carros, tendo em vista a existência de um amplo mercado. Além disso, mesmo que o CONTRAN (Conselho Nacional de Trânsito) tenha publicado uma resolução em que dispõe sobre as exigências dos automóveis e, entre tais exigências se encontre o “Gravador de Acidentes de Trânsitos”, este item não é explicado nem detalhado e apenas dá a entender que os veículos devam vir equipados com uma caixa-preta para armazenamento de dados do veículo (14).

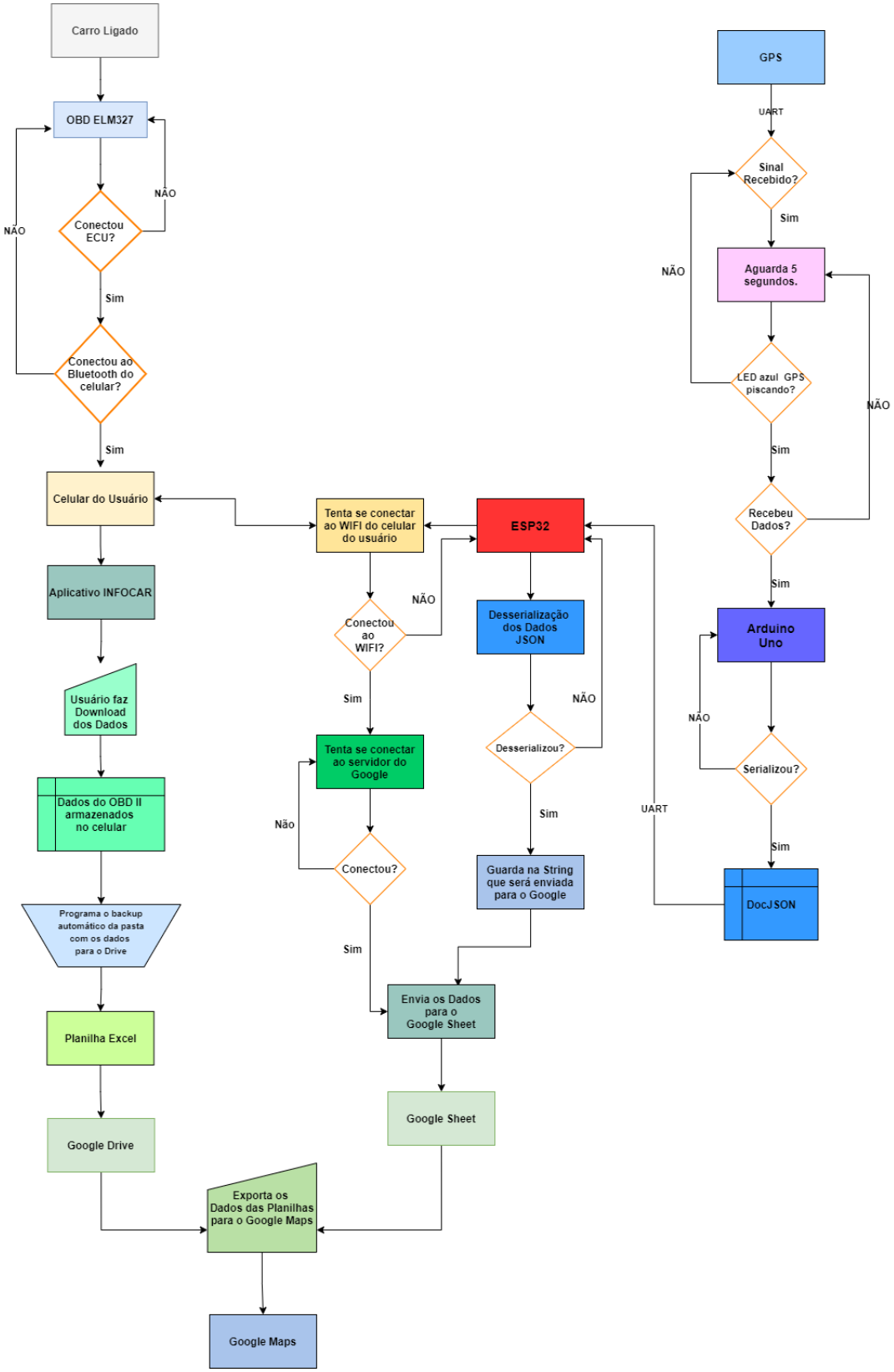
2 Metodologia

A primeira etapa do projeto consiste em um levantamento bibliográfico dos recursos que serão utilizados ao longo do projeto, como: informações para uso do *scanner* e alocação dos registros local e remotamente com auxílio de microcontroladores. A segunda parte consiste no procedimento realizado para o levantamento dos dados utilizados para o registro, bem como o procedimento para envio de informações coletadas através dos microcontroladores, combinados com o módulo GPS.

2.1 Leitura e Registro de Dados

A realização do projeto de um sistema de registro de dados automotivos de baixo custo com armazenamento em nuvem tem como objetivo fazer as leituras dos sensores presentes no carro e seu registro. Para o mesmo será utilizado um *scanner* de diagnóstico veicular ELM327, igual ao da figura 9, onde será possível obter os dados do líquido de arrefecimento do motor, do sensor RPM e velocidade, início e final do trajeto. Com o auxílio do aplicativo INFOCAR, presente no celular, onde esse se comunica com o *scanner* via conexão *Bluetooth*, serão realizados os registros que permitirão ser feitas as análises necessárias referentes aos sensores, bem como o acesso às informações e seu lançamento em forma de planilha. Para o auxílio do registro de informações, serão utilizados dois microcontroladores, Arduino Uno (13), *ESP32* (14) para coleta de dados com auxílio do módulo GPS. Esse sistema irá fornecer dados de localização, data, hora, velocidade e terá os dados armazenados em um banco de dados na nuvem.

Figura 8 – Fluxograma Completo do Projeto



Fonte: Autor

2.1.1 OBDII - Diagnóstico de Bordo

O *scanner* da imagem 9 trata-se de um OBD cuja sigla significa (*On Board Diagnostics*) ou apenas diagnóstico de bordo. Esse padrão foi criado por volta dos anos 90 pela indústria automotiva, com o intuito de padronizar a comunicação de leitura e transmissão de dados entre computadores e a central eletrônica do carro. Em 2010 no Brasil passou a ser adotada a segunda geração, adotando a conexão OBDII (15).

Figura 9 – *Scanner* Diagnóstico Veicular

Fonte: Autor

A necessidade de uma padronização surgiu da complexidade para acessar as informações da central eletrônica. Os profissionais cobravam muito caro, visto que cada fabricante poderia adotar o protocolo que bem entendesse(16).

Antes da padronização, eram comuns 4 protocolos, sendo eles:

- Ford - SAE J1850 PWM
- General Motors - SAE J1850 VPW
- Chrysler, carros europeus e asiáticos - ISO 9141-2
- Maioria dos carros europeus - ISO 14230 KWP2000

Entre eles variam mapa de pinos, qual pino é responsável por transmitir ou receber determinada informação, frequência de informações da rede e velocidade de conexão. Com carros cada vez mais modernos e, por conseguinte, um maior volume de dados armazenados nas ECU's, viu-se a necessidade de padronização para acesso a essas informações(16).

Com a padronização, a segunda geração do OBDII conta com cinco [5] protocolos de comunicação (17), sendo eles:

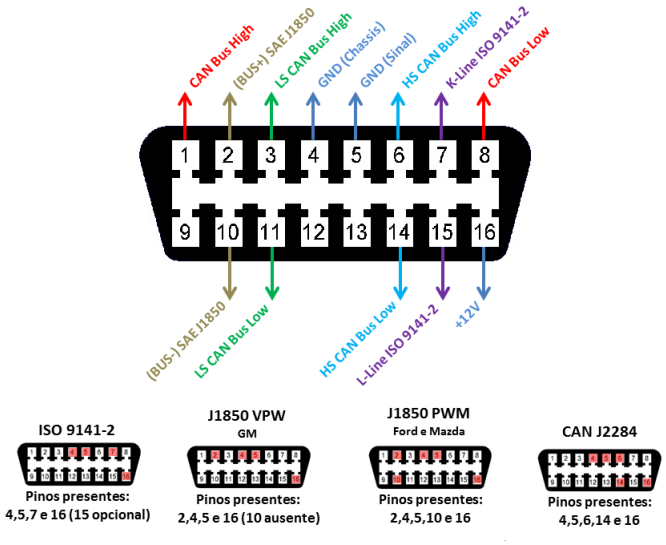
- **CAN** - ISO15765-4
- **KPW2000** - ISO14230-4

- ISO9141-2
- SAE J1850 VPW
- SAE J1850 PWM

Apesar da padronização, esta foi parcial, sendo assim a quantidade de informações que pode ser lida através do *scanner* vai depender do carro e do equipamento utilizado (17).

O OBDII conta com 16 pinos, sendo pinos específicos para cada protocolo, como mostrado na figura 10. Através do *scanner* é possível obter informações como: velocidade do veículo, nível de combustível, status da ignição, consumo de combustível, hodômetro, além de informar qual protocolo o automóvel utiliza. Lembrando que apenas carros com fabricação após 2010, são compatíveis com esse tipo de *scanner*, por ser uma versão mais simples e barata.

Figura 10 – Padrão do Conector

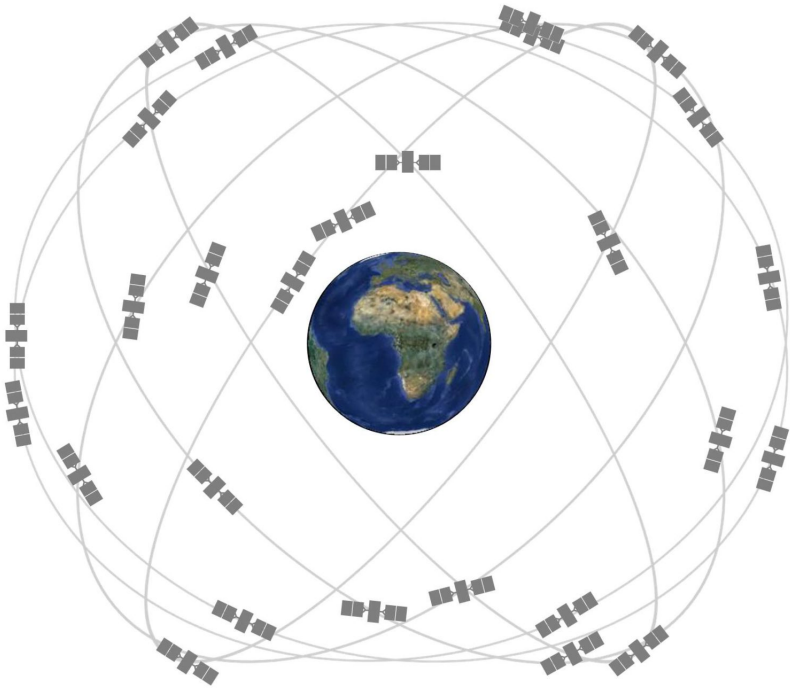


Fonte: (17)

2.1.2 GPS

O GPS (Sistema de Posicionamento Global), é um sistema de radionavegação que foi desenvolvido pelo Departamento de Defesa dos Estados Unidos da América, com o propósito de ser um sistema de navegação do exército americano (18) e acabou tornando-se um sistema de toda uma indústria de equipamentos e serviços de posicionamento global (19).

Figura 11 – Constelação GPS



Fonte: (19)

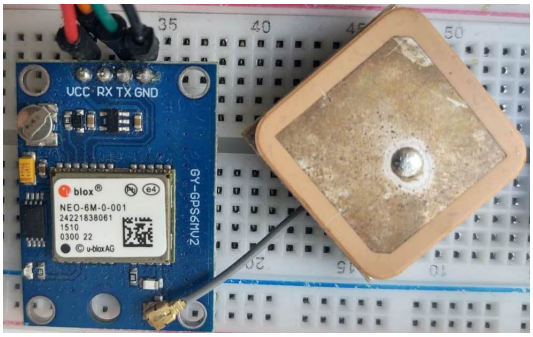
Sendo administrado pelo *50th Space Wing*, uma das áreas da Força Aérea dos Estados Unidos, o sistema GPS é composto por uma constelação de satélites (11) no início dos anos 90, 24 satélites eram operados e os demais funcionavam como *backups* em casos de falhas (20). Atualmente um novo bloco agregou o sistema e este é composto por 36 satélites, equipados com refletores a laser, que podem ser rastreados de forma independente. O sistema ainda conta com um bloco 2R-M, que não está disponível para civis, sendo apenas de uso militar (21). O arranjo dos satélites é feito de forma que exista 6 planos orbitais espaçados de forma igualitária circulando a Terra. Cada plano contém quatro *slots* ocupados por um satélite(19), formando um arranjo com 24 *slots*, como pode ser visto na figura 11. Os planos são programados para que pelo menos 4 satélites possam ser observados a qualquer momento do dia e de qualquer localidade(18). Cada satélite orbita a Terra quase duas vezes por dia a uma altitude de aproximadamente 20.200 km acima da superfície da Terra (20). O GPS tem como objetivo contribuir em atividades como navegação, levantamentos geodésicos e topográficos. Está programado para fornecer coordenadas bi ou tridimensionais de pontos no terreno, velocidade e direção

de deslocamento entre os pontos. Esse complexo opera ininterruptamente, independente das condições climáticas, ainda que possam provocar alguma interferência na qualidade dos resultados (18).

2.1.2.1 GY-GPS6MV2

Para obtenção dos dados de latitude, longitude, velocidade e sentido, será utilizado o módulo GPS modelo GY-GPS6MV2, mostrado na figura 12.

Figura 12 – GY-GPS6MV2



Fonte: Autor

Esse módulo funciona com uma tensão de trabalho de 3,3V 5V, uma interface serial de 3,3V, taxa de transmissão padrão de 9600 bps (22). Trata-se de um dispositivo bastante preciso, podendo rastrear até 22 satélites em 50 canais, com seu alto nível de sensibilidade, sendo um dos mais altos da indústria, com rastreamento de -161 dB, consumindo apenas 45mA de corrente de alimentação (22). Possuindo ainda uma capacidade de até 5 atualizações de localização por segundo com uma precisão de posição horizontal de 2,5m (23). Com seu regulador de tensão embutido, o módulo pode ser ligado diretamente ao 5V do Arduíno Uno, microcontrolador adotado para recebimento dos dados do GPS.

2.1.3 Microcontroladores

O microcontrolador é constituído por um único circuito integrado que reúne um núcleo de processador, periféricos de entrada e saída, memórias voláteis e não voláteis. Muitos inclusive se assemelham a computadores muito pequenos, com capacidade de realizar tarefas de maneira eficaz (24).

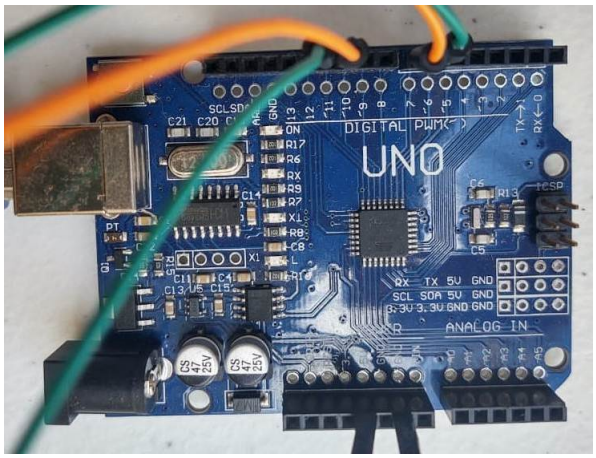
Com o crescimento do mercado voltado para a Internet das Coisas (IoT), tem feito dos microcontroladores peças importantes, devido a alta demanda, custo acessível, além dos seus sistemas computacionais compactos (24). Os microcontroladores devidos suas características, permitem a conexão entre outros microcontroladores, circuitos complementares e periféricos em seus pinos de entrada e saída, permitindo assim a personalização do sistema, ao mesmo tempo que proporciona o controle, conforme a programação utilizada

dentro do microcontrolador é adaptada conforme a necessidade do projeto e seu circuito empregado (25).

2.1.3.1 Arduino Uno

Para receber os dados obtidos pelo módulo GY-GPS6MV2 (12), será utilizado o Arduino Uno (13), um microcontrolador baseado no ATmega328, possuindo 14 pinos digitais de entrada/saída, dos quais 6 podem ser usados como saída PWM (*Pulse Width Modulation*), 6 entradas analógicas (26).

Figura 13 – Arduíno Uno



Fonte: Autor

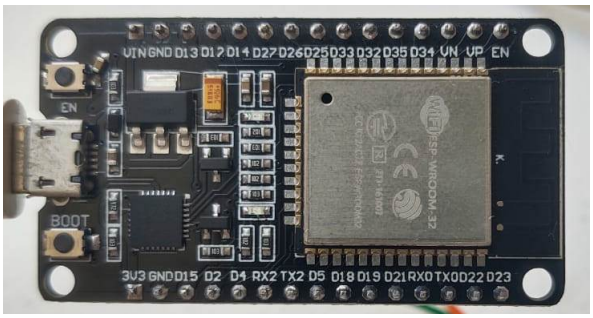
O Arduino Uno pode ser alimentado por uma fonte externa, suportando entre 7V-12V, sendo mais adequado entre 7V-9V por quesitos de segurança, ou através da porta USB(*Universal Serial Bus*) mesma porta onde ocorre a programação do microcontrolador (26). Com todas as suas características, trata-se de uma plataforma de *Hardware* e *Software* livre, possibilitando sua ampla implementação em projetos IoT, visto que é possível criar e modificar as bibliotecas para ele existentes conforme a necessidade do projeto, além da facilidade de combinação com outros microcontroladores e módulos, mostrando-se simples e acessível, ao mesmo tempo que não deixa a desejar para usuários mais experientes e exigentes (25).

2.1.3.2 ESP32

ESP32 é um microcontrolador de baixo custo e consumo energético, integrado com *WI-FI* e *Bluetooth* híbrido (clássico e BLE). Apresenta um sistema de processador *Dual Core*, múltiplos sensores embutidos, memória SRAM de 520Kb, memória *flash* de 16Mb, 240Mhz de *clock*, tensão operacional de 2,3V-3,6V, facilitando a implementação de projetos IoT, os tornando simples e compactos (27). O ESP32 representado na figura 14, possui 30 pinos GPIOS (*General Ports Input/Output*), terminais de entrada e saída,

tendo a possibilidade de atribuir através do *software* a finalidade do pino, podendo ser: UART, I²C,SPI, PWM, entre outros (27).

Figura 14 – DOIT ESP32 DEVKIT V1



Fonte: Autor

O *ESP32* mostra-se um *hardware* poderoso, com foco no desenvolvimento IoT, porém com fácil programação, já que o mesmo pode ser utilizado na interface de programação do Arduino, gerando praticidade e garantindo um imenso suporte ao desenvolvimento, com bibliotecas prontas, exemplos, fóruns de dúvidas.

2.2 Ambiente de Desenvolvimento

O Arduino IDE 15 é o *software* gratuito que permite programar o microcontrolador Arduino qualquer modelo da placa que esteja disponível e outras placas compatíveis com o ambiente. Além de desenvolver os programas, através dele é possível gravar os códigos no microcontrolador. Por meio do IDE, quando ocorre erros de programação, seja no código, ou por qualquer outro motivo, o ambiente gera código de erros, além de mostrar, se for o caso, qual linha está gerando o problema. O *software* é compatível com diversos sistemas operacionais, demonstrando assim uma grande versatilidade (28).

Figura 15 – Arduino IDE

Fonte: (28)

O *software* Arduino IDE (15) conta com um ambiente totalmente em português e de fácil navegação, possibilitando que aqueles que não dominam o inglês possam utilizar o programa sem dificuldades. Além de contar com todas as funcionalidades separadas por funções, possibilitando o acesso de forma direta.

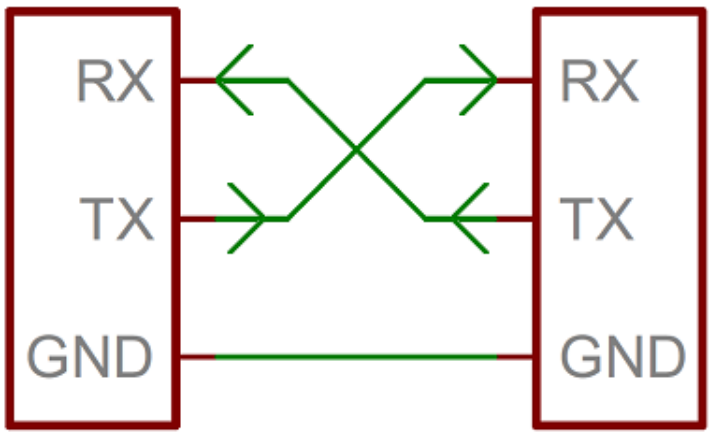
2.2.0.1 ArduinoJSON

"ArduinoJSON foi criado pelo Benoît Blanchon, um desenvolvedor C++ (29). ArduinoJson é uma forma fácil de se gerar um documento JSON e extrair informações, com poucas linhas de código, e de forma eficiente. Foi programado para ser utilizado em pequenos microcontroladores, como o Arduíno. Mas além de compatível com o Arduíno, é compatível com as placas *ESP32*, podendo inclusive ser utilizada no computador (29). A vantagem do ArduinoJson é que, por usar menos memória RAM, ele evita que cópias de arquivos sejam feitas desnecessariamente, poupando que a mesma informação seja copiada várias vezes na memória (29).

2.3 Protocolo de Comunicação Serial UART

Protocolos de comunicação são amplamente utilizados em muitas áreas voltadas para tecnologia, possibilitando a condução de informações entre dispositivos. Por serem utilizados em diversas áreas, diferentes protocolos foram desenvolvidos ao longo dos anos. O protocolo de comunicação UART (*Universal Asynchronous Receiver/Transmitter*) tem como vantagem a simplicidade e a possibilidade ser *full-duplex* (cada fio possui um fluxo de dados em uma direção), dessa forma o dispositivo pode transmitir e receber dados ao mesmo tempo (30). A transmissão de dados pelo protocolo funciona *bit* a *bit*, onde o pino de transmissão (TX), envia o pacote para o receptor (RX), no qual irá interpretar cada *bit* recebido. Cada pacote conta com 1 *start bit* que é o responsável por indicar o início da mensagem, e 1 ou 2 *stop bits*, que sinalizam o final da mensagem (31).

Figura 16 – Barramento Serial



Fonte: (30)

Como ilustrado na figura 16, para cada pino TX deve existir um pino RX e a conexão entre eles deve ser de forma alternada.

2.4 Banco de Dados Remoto

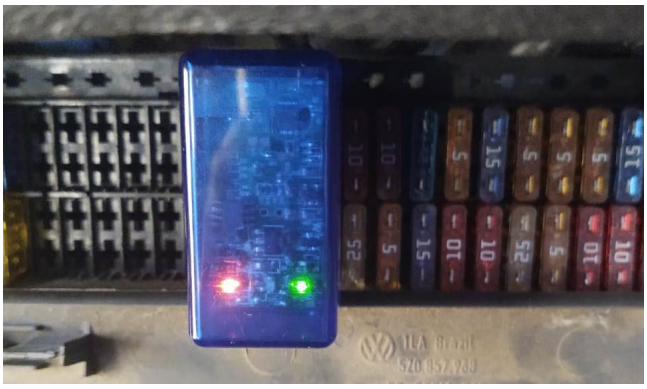
O banco de dados é a junção de dados que pode variar entre registro de pessoas, lugares, coisas, que precisam ser armazenados para segurança ou verificação futura (32). Com o crescimento da popularidade dos projetos IoT, é muito comum encontrar nuvens IoT disponíveis para armazenar e monitorar dados de vários sensores. Tendo os dados armazenados em uma nuvem, esses podem ser usados para análises de melhorias no sistema, e em eventuais perdas do *hardware*, não se perde todo o trabalho, em caso de monitoramentos com imagens, ou localizadores via GPS. Sua aplicação para registro de dados é vasta, podendo ser aplicadas em laboratórios, centros de pesquisas, entre outros (33).

2.5 Implementação

Para a primeira parte de implementação do projeto foi utilizado o OBDII (9) conectado ao carro como mostrado na figura 17. Visto que o OBDII ELM327(9) não manteve uma conexão constante e longa o suficiente com o*ESP32* via *Bluetooth*, para que fosse possível receber os dados necessários, viu-se a necessidade de utilizar um aplicativo já disponível para o *scanner* para coleta dos dados.

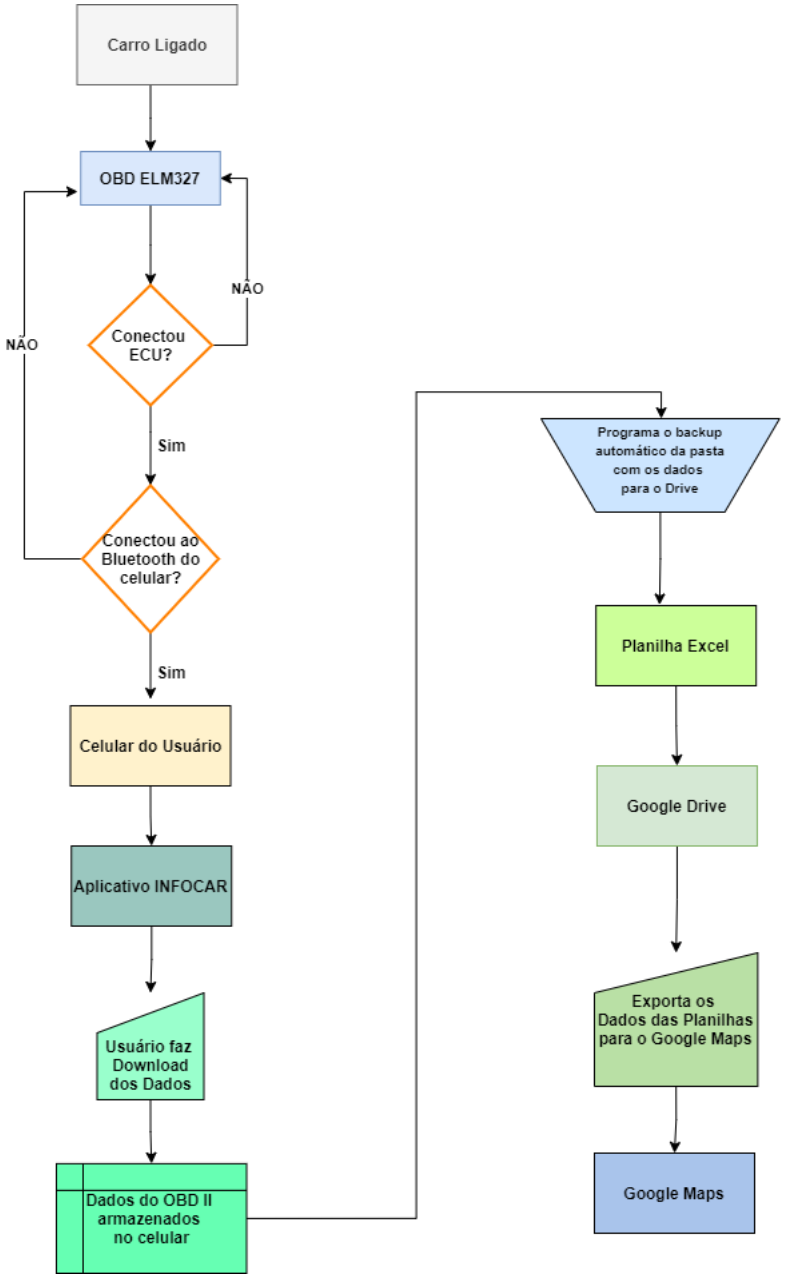
A imagem 18 mostra como funciona a conexão do OBDII com o automóvel. O carro é ligado, o *scanner* tenta se conectar com o celular do usuário através do *Bluetooth*, ao passo que tenta uma conexão com a central de controle do carro. A partir do momento que as conexões são estabelecidas, o próprio aplicativo encarrega-se de guardar os dados fornecidos pela central de comando. Esses dados ficarão armazenados no celular do usuário.

Figura 17 – Scanner ELM327 conectado ao carro



Fonte: Autor

Figura 18 – Fluxograma do sistema utilizando o *scanner* ELM327



Fonte: Autor

Utilizou-se um aparelho *smartphone* com auxílio do aplicativo INFOCAR, disponível na loja de aplicativos. Através dele foi possível manter uma conexão estável durante todos os percursos realizados para coleta de dados. Como mostrado na figura 19, é possível ter acesso ao monitoramento do carro, registro de condução, painel de controle e diagnóstico do veículo. O aplicativo além de armazenar os dados, permite que o usuário faça o *download* dos dados em formato de planilha.

Para ter acesso a esses dados, primeiro é necessário fazer o registro, que consiste no e-mail do usuário, modelo do carro e ano. Com o registro feito, é necessário que o usuário permita que o *scanner* conecte-se ao celular via *Bluetooth*. O OBDII(17) é

Figura 19 – Interface do aplicativo INFOCAR

Fonte: Autor

acoplado a central do carro, a conexão com o automóvel é feita assim que a ignição for ligada. O armazenamento no celular através do aplicativo, no modelo de *scanner* em questão, somente é possível via *Bluetooth*, mas para outros modelos, existe a possibilidade de conexão via *WI-FI*.

Como mostrado na figura 20, já nessa tela é possível obter alguns dados referentes ao registro de condução. Ele atribui pontuações para condução segura, que envolve frear muito rapidamente e soltar o pedal do freio com rispidez. Ao se fazer o *download* do arquivo, é possível obter mais dados, como horário de início e final da condução, local de chegada, RPM máximo e médio, máxima velocidade, a média da velocidade, temperatura do óleo de arrefecimento, eficiência do combustível e quantidade de desacelerações rápidas que ocorreram durante o trajeto. O local de chegada foi ocultado por segurança.

Figura 20 – Registro de condução do aplicativo INFOCAR

Fonte: Autor

Ao fazer o *download* dos dados do aplicativo, através da opção "*download* do registro do veículo" fornecida pelo desenvolvedor, os dados são guardados no celular do usuário. Para utilizar os registros, utilizou-se da opção de *backup* automático da pasta do celular onde os registros foram armazenados, utilizando o *Google Drive*, ferramenta de arquivos em nuvem disponibilizada pela empresa *Google*. O *backup* foi utilizado para que os registros fossem acessados de forma fácil e rápida.

Podemos visualizar isso nas tabelas transcritas 1 e 2.

Tabela 1 – Parte 1 - Tabela dados OBDII

Hora de início da condução	Hora de Fim da Condução	Tempo de condução (mm: ss)	Local de chegada
24 de março de 2021 18:21:32	24 de março de 2021 18:35:32	14: 0	
26 de março de 2021 14:58:43	26 de março de 2021 15:16:13	17:30	
29 de março de 2021 19:06:44	29 de março de 2021 19:09:28	2:44	
08 de maio de 2021 18:39:30	08 de maio de 2021 18:44:13	4:43	
09 de maio de 2021 14:53:48	09 de maio de 2021 14:57:03	3:15	
09 de maio de 2021 15:03:59	09 de maio de 2021 15:08:31	4:32	

Fonte: Autor

Tabela 2 – Parte 2 - Tabela dados OBDII

distancia	Máxima velocidade	média de velocidade	RPM máximo	RPM médio	Temperatura máxima do refrigeramento
2,39km	59,0km / h	20,05km / h	3068 rpm	1569,04 rpm	72,0 ° C
9,40km	78,0km / h	32,19km / h	3723 rpm	1935,18 rpm	94,0 ° C
1,35km	55,0km / h	27,19km / h	3205 rpm	1834,16 rpm	78,0 ° C
1,99 km	59,0km / h	25,27km / h	3573 rpm	1764,04 rpm	65,0 ° C
1,13km	45,0km / h	20,75km / h	2669 rpm	1577,92 rpm	49,0 ° C
2,13km	43,0km / h	28,15km / h	2591 rpm	1779,82rpm	68,0 ° C

Fonte: Autor

2.5.1 Obtenção de dados com módulo GPS

Para registro básico dos dados do veículo, além dos obtidos pelo aplicativo, que ficam armazenados no celular do usuário, pensou na possibilidade de outro armazenamento, visto que o *scanner* (9) não se adequou como esperado ao microcontrolador *ESP32* (14), não permitindo uma conexão estável. Fez-se uso do microcontrolador Arduíno Uno (13) em conjunto com o módulo GPS gy-neo6mv2 (12) para obtenção de dados da localização, velocidade, sentido, data e hora (21).

A conexão dos terminais pode ser vista na tabela 3.

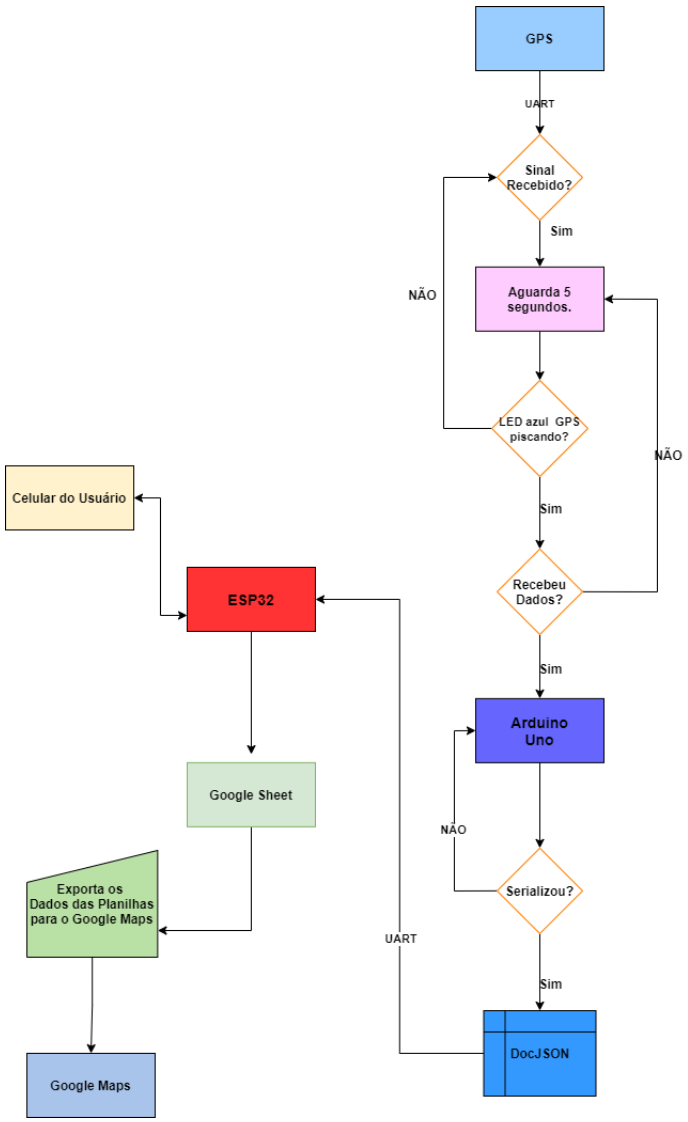
Tabela 3 – Conexão Arduíno Uno com módulo GPS

Arduíno Uno	GPS
GPIO 6 (RX)	TX
GPIO 7 (TX)	RX
5V	VCC
GND	GND

Fonte: Autor

Como mostrado na tabela 3, alguns pinos do Arduíno (13) foram definidos como TX (transmissor) e RX (receptor), conforme o protocolo de comunicação UART, e conec-

Figura 21 – Fluxograma do sistema composto pelo Arduíno Uno



Fonte: Autor

tados de forma alternada com o módulo GPS (12), ou seja, TX do Arduino com o RX do GPS, RX do Arduino com TX do módulo, possibilitando dessa forma a comunicação entre o microcontrolador e o módulo para envio e recepção de dados. Utilizando o *software* Arduino IDE (15) como interface de programação, visto que o mesmo possibilita a integração dos módulos sem problemas através de bibliotecas de *softwares* livres, como a utilizada *TinyGPS* (34).

Como o chip *ublox 6m* integrado ao módulo gy-GPS6mv2 (12) entrega dados "crus"(22), da forma como o GPS envia para o Arduino, utiliza-se a biblioteca *TinyGPS* para tratar essas *strings* e receber dados que possam ser lidos sem dificuldades pelo usuário.

Para o recebimento de dados pelo GPS (12), primeiro é necessário que ele triangule, ou seja, estabeleça conexão com os satélites, para poder obter os dados requeridos.

Figura 22 – Dados obtidos pelo GPS sem refinação

```
$GPGLL,,,,,210402.00,V,N*4F
$GPRMC,210403.00,V,,,,,,,150521,,,N*7B
$GPVTG,,,,,,,,,N*30
$GPGGA,210403.00,,,,,0,00,99.99,,,,,,*62
$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30
$GPGSV,1,1,01,13,,,09*73
$GPGLL,,,,,210403.00,V,N*4E
$GPRMC,210404.00,V,,,,,,,150521,,,N*7C
$GPVTG,,,,,,,,,N*30
$GPGGA,210404.00,,,,,0,00,99.99,,,,,,*65
```

Fonte: Autor

O código 23 mostra a programação para o recebimento dos dados. A função *while* é a instrução responsável por identificar se há conexão. O *loop* infinito vai executar a função continuamente verificando se há conexão com os satélites para o recebimento das informações. A programação foi feita para que assim que o primeiro dado for recebido, ele irá atualizar a cada 5 segundos (23).

Figura 23 – Código GPS via Arduíno

```
34 void loop() {
35   bool recebido = false;
36   static unsigned long delayPrint;
37
38   while (serial1.available()) {
39      char cIn = serial1.read();
40      recebido = (gps1.encode(cIn) || recebido);  //Verifica até receber o primeiro sinal dos satelites
41   }
42
43   if ( (recebido) && ((millis() - delayPrint) > 5000) ) {  //Mostra apenas após receber o primeiro sinal.
44      delayPrint = millis();
45
```

Fonte: Autor

Com o auxílio da biblioteca *TinyGPS*, utilizando comandos estabelecidos, foi possível obter os dados de localização, velocidade, sentido, data e hora, sendo possível saber quantos satélites foram encontrados e altitude, mas esses não foram utilizados.

O recebimento das coordenadas de latitude e longitude, que são linhas usadas para indicar a localização de qualquer ponto na Terra, são obtidas utilizando o comando ***gps1.f_get_position***. GPS1 foi o objeto criado para associar a biblioteca *TinyGPS* para receber os dados do módulo, podendo ser o nome que o programador achar mais adequado ao seu projeto. A função *f._get_position* retorna as coordenadas relacionadas aos pontos referentes a localização: latitude e latitude (24).

Figura 24 – Código GPS via Arduíno

```
gps1.f_get_position(&latitude, &longitude /*&idadeInfo*/);   //O método f_get_position é mais

if (latitude != TinyGPS::GPS_INVALID_F_ANGLE) {
   Serial.print("Latitude: ");
   Serial.println(latitude, 6);  //Mostra a latitude com a precisão de 6 dígitos decimais
}

if (longitude != TinyGPS::GPS_INVALID_F_ANGLE) {
   Serial.print("Longitude: ");
   Serial.println(longitude, 6);  //Mostra a longitude com a precisão de 6 dígitos decimais
}
```

Fonte: Autor

Através do módulo é possível se obter os dados de data e hora, porém o módulo vem de acordo com o meridiano de Greenwich, o que difere em 3 horas a mais em relação ao horário oficial do Brasil. Para obtenção do horário oficial, ou seja, o horário de Brasília, foi necessário fazer uma correção manual, para que a hora saísse correta. Os dados de data e hora puderam ser obtidos utilizando o comando ***gps1.crack_datetime*** (25).

Figura 25 – Código GPS via Arduíno

```
//Dia e Hora

gps1.crack_datetime(&ano, &mes, &dia, &hora, &minuto, &segundo, &centesimo, &idadeInf);
hora = hora - 3; //corrigindo manualmente o horario do meridiano de greenwich para o brasil.

Serial.print("Data (GMT): ");
Serial.print(dia);
Serial.print("/");
Serial.print(mes);
Serial.print("/");
Serial.println(ano);

Serial.print("Horario (UCT): ");
Serial.print(hora);
Serial.print(":");
Serial.print(minuto);
Serial.print(":");
Serial.println(segundo);
```

Fonte: Autor

A direção do trajeto, com uma precisão de 0.5 grau (22), pode ser obtido utilizando comando ***gps1.course()***, sendo necessário fazer a conversão manual para graus. Os dados de velocidade podem ser obtidos utilizando o comando ***gps1.f_speed_kmph()***, nesse caso para receber os dados em Km/h, existindo outras possibilidades, como "nós"e "m/s". Os dados da velocidade possuem uma precisão de aproximadamente 0,03 km/h ou 0,1 m/s conforme especifica o fabricante (22). Os primeiros dados puderam ser visualizados utilizando o monitor serial do Arduino IDE (15) como mostrado na figura 26.

Figura 26 – Monitor Serial Arduino IDE

```
19:05:27.563 -> ----------------------------------------
19:05:27.563 -> Latitude: -15.874
19:05:27.596 -> Longitude: -47.97
19:05:27.596 -> Data (GMT): 15/5/2021
19:05:27.596 -> Horario (UCT): 19:5:28
19:05:27.631 -> Velocidade (km/h): 0.20
19:05:27.631 -> Sentido (grau): 185.38
19:05:33.565 -> ----------------------------------------
```

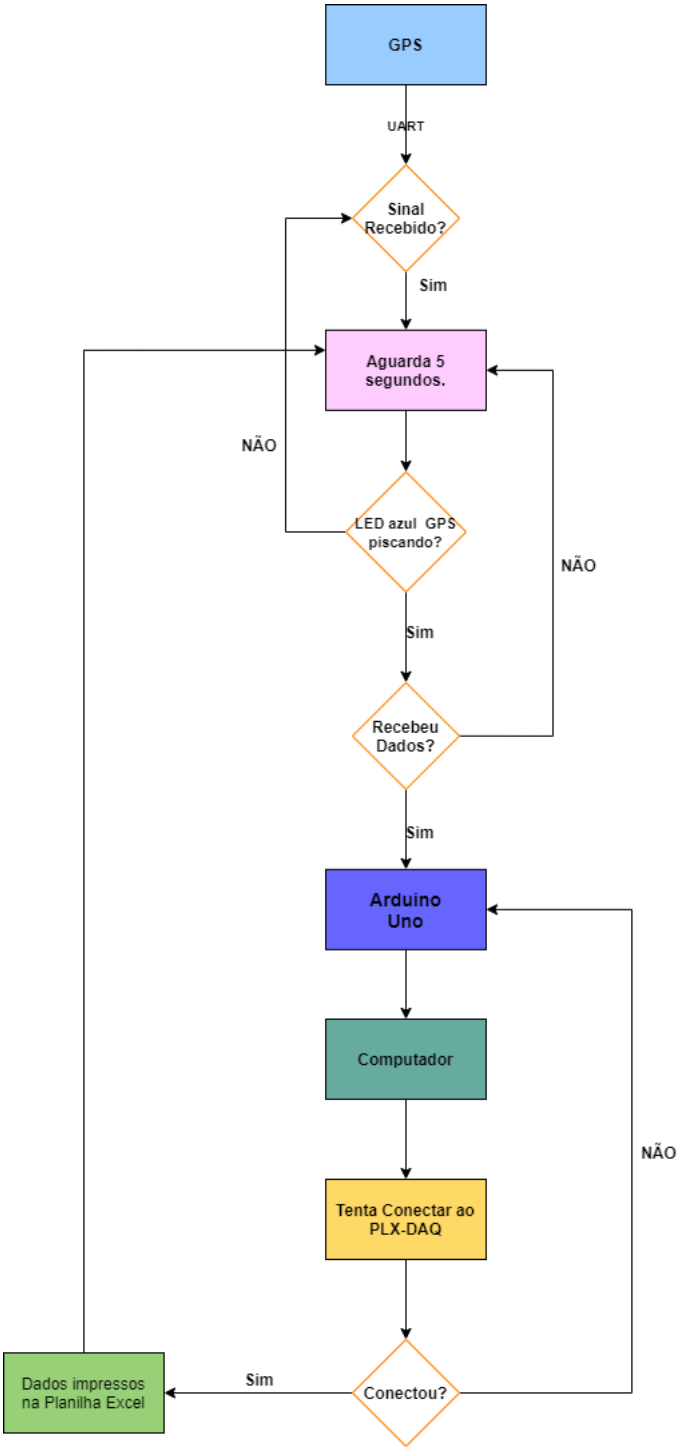
Fonte: Autor

A partir dos dados obtidos através do OBDII por meio do aplicativo INFOCAR, fez-se uso de tabelas, onde foi possível saber se os dados obtidos através do GPS e do *scanner* combinavam entre si(27).

Utilizando o Arduino Uno, em vez de mostrar os dados no monitor serial, como foi feito (26), foi gerado uma tabela com esses registros.

Para a implementação da tabela com os dados recebidos pelo GPS, foi utilizado o *software* PLX-DAQ, uma ferramenta complementar para aquisição de dados para o

Figura 27 – Diagrama Arduino Uno comunicação Excel

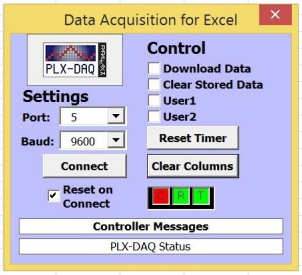


Fonte: Autor

Microsoft Excel, que propiciou a implementação da tabela. Ele funcionou como o monitor serial do Arduino, com a diferença de que os dados ficam gravados e podem ser subscritos, desde que haja o comando no código no Arduino IDE (15).

Figura 28 – Planilha PLX-DAQ - Dados GPS

A	B	C	D	E	F	G
17:35:14	**22/04/2021**	**Latitude**	**Longitude**	**Altitude**	**Velocidade**	**Sentido**
17:35:15	22/04/2021	-15,8736	-47,974702	1045,4	1,22	297,81
17:35:16	22/04/2021	-15,8736	-47,974703	1045,4	1,65	297,81
17:35:18	22/04/2021	-15,8736	-47,974707	1045,4	3,04	297,81
17:35:19	22/04/2021	-15,8736	-47,974704	1045,4	1,74	297,81
17:35:21	22/04/2021	-15,8736	-47,974701	1045,4	1,89	297,81
17:35:22	22/04/2021	-15,8736	-47,974702	1045,4	0,72	297,81
17:35:24	22/04/2021	-15,8736	-47,974700	1045,4	0,28	297,81
17:35:25	22/04/2021	-15,8736	-47,974699	1045,4	0,3	297,81
17:35:31	22/04/2021	-15,8736	-47,974696	1045,4	0,81	297,81
17:35:32	22/04/2021	-15,8736	-47,974695	1045,4	0,65	297,81
17:35:34	22/04/2021	-15,8736	-47,974693	1045,4	1,15	297,81
17:35:35	22/04/2021	-15,8736	-47,974692	1045,4	0,94	297,81
17:35:37	22/04/2021	-15,8736	-47,974688	1045,4	0,81	297,81
17:35:38	22/04/2021	-15,8736	-47,974686	1045,4	0,09	297,81
17:35:40	22/04/2021	-15,8736	-47,974685	1045,4	0,78	297,81
17:35:41	22/04/2021	-15,8736	-47,974683	1045,4	0,7	297,81
17:35:43	22/04/2021	-15,8736	-47,974680	1045,4	1,89	297,81
17:35:44	22/04/2021	-15,8736	-47,974678	1045,4	1,48	297,81
17:35:46	22/04/2021	-15,8736	-47,974676	1045,4	1,44	297,81
17:35:47	22/04/2021	-15,8736	-47,974675	1045,4	1,28	297,81
17:35:49	22/04/2021	-15,8736	-47,974673	1045,4	0,93	297,81
17:35:50	22/04/2021	-15,8736	-47,974672	1045,4	1,52	297,81
17:35:52	22/04/2021	-15,8736	-47,974671	1045,4	1,07	297,81
17:35:53	22/04/2021	-15,8736	-47,974669	1045,4	1,48	297,81
17:35:55	22/04/2021	-15,8736	-47,974667	1045,4	1,15	297,81
17:35:56	22/04/2021	-15,8736	-47,974666	1045,4	0,67	297,81
17:35:58	22/04/2021	-15,8736	-47,974664	1045,4	5,91	297,81



Fonte: Autor

Através da figura 28 é possível verificar a planilha após o recebimento dos dados. Para que a planilha fosse gerada, como mostrado (28), utilizou-se de comandos, para que os dados fossem implementados.

Figura 29 – Código para gerar planilha

```
18
19 int linha = 0;
20 int LABEL = 1;
21
22 void setup() {
23    serial1.begin(9600);
24    Serial.begin(9600);
25    Serial.println("CLEARDATA");
26
27    Serial.println("LABEL, TIME, DATE, satelites, Latitude, Longitude, Altitude, Velocidade, Sentido");
28 }
29
```

Fonte: Autor

Conforme mostrado na figura 29, é utilizado a função *LABEL* para que as colunas sejam criadas, correspondendo aos dados que serão recebidos. Os dados vão sendo impressos na tabela conforme mostrado na figura 30, onde as vírgulas ","funcionam como colunas, ou seja, o próxima informação a ser impressa, irá para próxima coluna.

Figura 30 – Código para gerar planilha

```
109  linha++;
110
111  Serial.print("DATA,TIME,DATE,");
112  Serial.print(satelites);
113  Serial.print(",");
114  Serial.print(latitude,6);
115  Serial.print(",");
116  Serial.print(longitude,6);
117  Serial.print(",");
118  Serial.print(altitudeGPS);
119  Serial.print(",");
120  Serial.print(velocidade,2);
121  Serial.print(",");
122  Serial.println(float(sentido) / 100, 2);
123
124   if(linha > 300){
125      linha = 0;
126      Serial.println("ROW,SET,2");
127    }
128  }
```

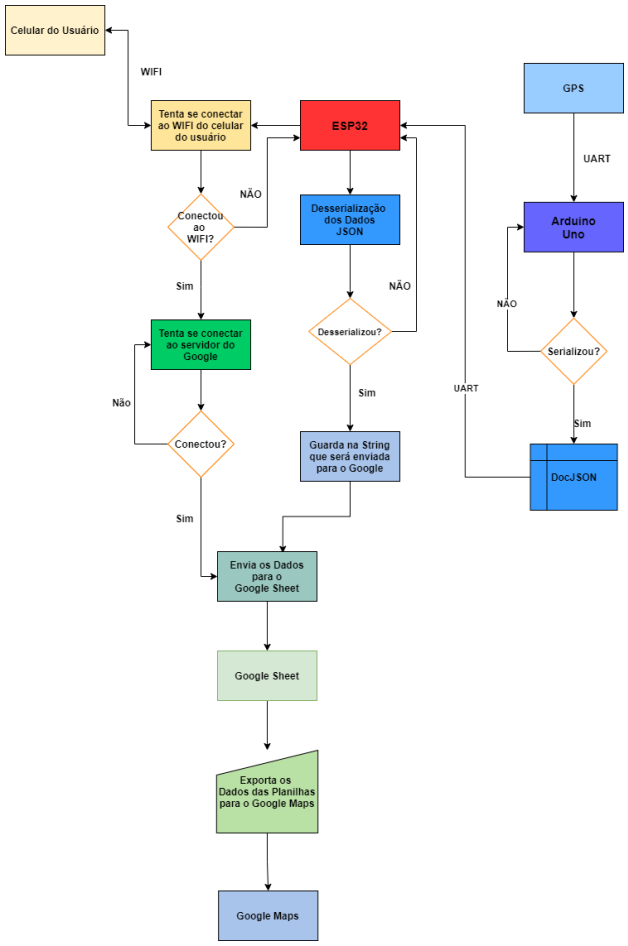
Fonte: Autor

Foi criado uma condição dentro do *loop* para que, quando passasse da linha 300, ele subscrevesse os dados a partir da linha 2, visto que a primeira linha compõe os nomes dos dados a serem recebidos. Os dados de data e hora, para implementação, optou-se por utilizar os fornecidos pelo computador, por questão de estética, sendo gerados através dos comandos *TIME, DATE*.

Para que os dados passem a ser impressos na planilha como um monitor serial é necessário que haja habilitação, visto que quando se abre a planilha do *software* PLX-DAQ, ele é composto por macros, os quais precisam ser habilitados. No código, antes de chamar o primeiro dado a ser gravado na planilha, utiliza-se o comando **DATA**, que é responsável por saber o caminho onde será gravado os dados na planilha. Atribuindo o comando, na planilha, quando se habilita os macros, aparece a caixa de comando, como pode ser vista na figura 28, essa caixa de controle, será habilitada conforme os comandos utilizados para rodar o código no Arduino, devendo ser a mesma porta de programação, ou seja a porta USB utilizada para programar o Arduino e o tempo utilizado, no caso *baud rate* de 9600. Como a planilha funciona como uma espécie de monitor serial, o monitor serial do Arduino IDE (15) não funciona enquanto a planilha está habilitada.

2.5.2 Comunicação do Arduino Uno com o *ESP32*

Figura 31 – Fluxograma do sistema com a parte do ESP32



Fonte: Autor

Com a obtenção dos dados utilizando o microcontrolador Arduino Uno (13) e com o módulo GPS (12), utilizou-se do *ESP32* (14) como escravo, através da comunicação UART (16), para o envio das informações para o banco de dados, devido a falta de conexão com a internet na placa Arduino.

Para que não houvesse perda dos dados obtidos, utilizou a linguagem de programação ArduinoJSON para criar documentos, para cada variável respectivamente. A implementação do documento pode ser visto na figura 32. É criado um espaço na memória RAM para alocação do documento na pilha. Foi atribuído valores para cada documento, no caso, foram guardados os valores recebidos pelo módulo GPS(12).

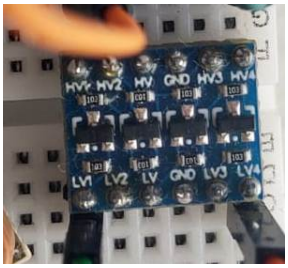
Através da comunicação UART entre o *ESP32* e o Arduino Uno, foi criado um *link serial* entre os pinos de comunicação, esse *link* funciona como um fio. Esse *link* chamado pelo comando ***serializeJson*** é o responsável por encaminhar os dados para *ESP32*. A conexão do Arduino Uno com o *ESP32* não pode ser feita de forma direta, visto que trabalham com alimentação distinta. Foi necessário fazer o uso do módulo conversor de nível lógico como mostrado na figura 33.

Figura 32 – Gerando Documento JSON

```
88
89      // criando documento JSON
90    StaticJsonDocument<200> doc;
91
92
93    doc["latitude"] = latitude;
94    doc["longitude"] = longitude;
95    doc["velocidade"] = velocidade;
96    doc["sentido"] = sentido;
97    doc["dia"] = dia;
98    doc["mes"] = mes;
99    doc["ano"] = ano;
100   doc["hora"] = hora;
101   doc["minuto"] = minuto;
102   doc["segundo"] = segundo;
```

Fonte: Autor

Figura 33 – Conversor Nível Lógico 5V/3,3V



Fonte: Autor

A conexão do Arduino com o *ESP32* pode ser vista na tabela 4.

Tabela 4 – Conexão Arduino Uno - ESP32

Conexão Pinos Arduíno - Esp32			
Arduíno	Conversor		ESP32
10	HV1	LV1	15
11	HV4	LV4	13

Fonte: Autor

A recepção dos dados pelo *ESP32* é feita através de um *linkserial*, que é o responsável pela conexão UART entre os microcontroladores. Esse *linkserial* trabalha a uma taxa baixíssima, para o projeto foi escolhido 4800, a menor disponível no ambiente de desenvolvimento, para evitar erros de comunicação e perda de dados entre as transferências.

Essa desserialização foi feita de modo que os dados recebidos fossem guardados nas variáveis correspondentes as recebidas pelo GPS de modo a facilitar a integralização com o banco de Dados. A figura 35 mostra como ficam os dados após ocorrida a desserialização do documento JSON(32) enviado pelo Arduino, vista pelo monitor serial.

Figura 34 – Desserialização do documento Json

```
34
35  if(linkSerial.available()){
36    StaticJsonDocument<300> doc;
37    DeserializationError err = deserializeJson(doc, linkSerial);
38
39    if(err == DeserializationError::Ok){
40      //Se tudo certo, printa a leitura dos valores recebidos do arduino.
41         auto latitude = doc["latitude"].as<float>();
42         auto longitude = doc["longitude"].as<float>();
43         auto velocidade = doc["velocidade"].as<float>();
44         auto sentido = doc["sentido"].as<float>();
45
```

Fonte: Autor

Figura 35 – Desserialização do documento Json

```
.9:02:34.140 -> ---------------
.9:02:34.140 -> Latitude -15.87
.9:02:34.140 -> Logitude -47.97
.9:02:34.140 -> Velocidade 0.44
.9:02:34.140 -> Sentido 174.11
.9:02:34.140 -> Data: 15/5/2021
.9:02:34.140 -> Horario: 19:2:34
.9:02:40.131 -> ---------------
```

Fonte: Autor

2.5.3 Registro no Banco de Dados

O registro de dados em um banco de dados foi escolhido devido a garantia de que, mesmo que haja um acidente e o sistema fique comprometido, até o momento da perda de sinal, os dados estarão sendo mantidos em um ambiente acessível. A implementação utilizando o PLX-DAQ (27) serviu como ensaio, simulando um banco de dados interno no computador e mostrando como o sistema se comportava, sendo registrado em planilhas. A partir dessa simulação que se mostrou viável e segura, optou-se por utilizar o *Google Sheet*, a planilha do Google, uma plataforma gratuita, de fácil comunicação e que possui como espaço tudo o que estiver disponível no *Drive*, a pasta de arquivos na nuvem.

A comunicação do *ESP32* com o *Google Sheet* (31) se deu através da criação de um formulário da própria plataforma do Google. Com a criação das perguntas, que correspondem às colunas da planilha e as respostas ocorrem conforme há o recebimento dos dados pelo GPS. Esse formulário é associado a uma planilha existente, criada previamente para servir como banco de dados.

Assim que o formulário é criado, para cada pergunta é gerado uma espécie de endereço, chamado de *&entry* acompanhado por uma sequência de números. Cada *&entry* gerado irá corresponder a uma coluna distinta, sendo possível assim, associar para qual coluna irá ser guardado determinado dado do GPS. Por exemplo os dados pertencentes a latitude, a coluna da latitude, os dados de longitude, para a coluna da longitude e assim para todos os outros dados requisitados.

Figura 36 – Endereço do formulário para comunicação

```
client.setInsecure();
if (client.connect("docs.google.com", 443) == 1){
   //Atribuimos a String auxiliar na nova String que sera enviada
    String toSend = textFix + "&entry.5[redacted]="+ latitude + "&entry.12[redacted]=" + longitude + "&entry.15[redacted]=" + velocidade + '
    toSend += "&submit=Submit HTTP/1.1";//Completamos o metodo GET para nosso formulario.
    client.println(toSend);//Enviamos o GET ao servidor-
    client.println("Host: docs.google.com");//-
    client.println();//-
    client.stop();//Encerramos a conexao com o servidor
    Serial.println("Dados enviados.");//Mostra no monitor que foi enviado
}
else{

    Serial.println("Erro ao se conectar");//Se nao for possivel conectar no servidor, ira avisar no monitor.
```

Fonte: Autor

Como mostrado na figura 36 é criado uma *string* responsável por auxiliar no envio das informações. A ela é relacionado o endereço obtido através do código fonte do formulário, o *&entry*, juntamente ao dado que deve ser gravado naquele endereço, ou seja, a qual coluna corresponde a informação a ser gravada.

Figura 37 – Endereço do formulário

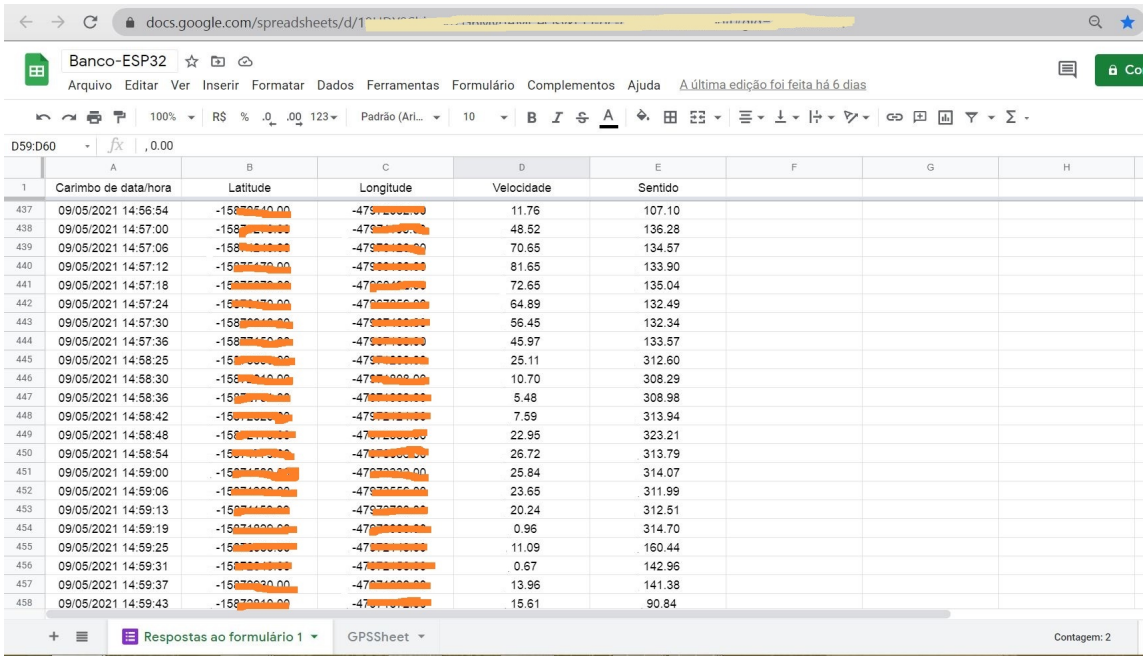
```
13
14 String textFix = "GET //forms/d/e/1FAIpQLSe4V[redacted]mfQjnt-Q/formResponse?ifq&entry.503[redacted]=&entry.129[redacted]=&entry.157
15 //Essa String sera uma auxiliar contendo o link utilizado pelo GET
```

Fonte: Autor

Uma *string textfix* (37) é responsável por guardar o endereço do formulário que é associado à planilha do *Google*. Nela contém o endereço completo do formulário e o conjunto de perguntas responsáveis pela formação das colunas. Ela é criada para facilitar e não haver a necessidade de ficar escrevendo a todo momento, como mostrado na figura 37.

A comunicação e o posterior envio dos dados é feito através da comunicação do *ESP32* com a conexão *WI-FI*. Para implementação do projeto, usou-se a conexão de internet fornecida pelo telefone celular. A partir da conexão *WI-FI*, ocorre a conexão com o servidor da *Google* para que seja possível o envio das informações e o seu registro na planilha, como pode ser visto na figura 38. O acesso à planilha é feito de forma compartilhada, não sendo possível alteração dos dados, apenas leitura e seu *download*.

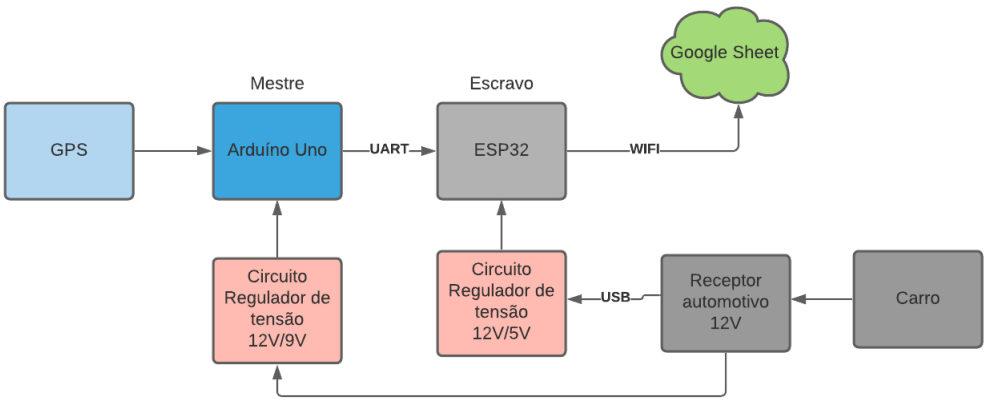
Figura 38 – Dados Armazenados no Google Planilha



Fonte: Autor

2.5.4 Alimentação do Sistema

Figura 39 – Alimentação do Sistema Eletrônico



Fonte: Autor

O sistema eletrônico foi alimentado(39) através do receptor presente nos automóveis, que até o início dos anos 2000 era conhecido como acendedor de cigarros, mas com a evolução da tecnologia, passou a ser utilizado como forma de carregar eletrônicos como aparelhos celulares. Através dessa entrada, utilizando um adaptador USB, que fornece uma tensão de 12V, foi feita a alimentação dos componentes. Para que não ocorresse danos ao sistema, visto que o Arduíno Uno aceita como tensão 9V e o *ESP32* uma tensão de 3,6V foi necessário um circuito regulador de tensão, que teve como finalidade a manutenção da tensão de saída, fornecendo a tensão necessária para os dispositivos.

2.6 Protocolo Experimental

Com o intuito de realizar testes no sistema eletrônico, quanto à capacidade de se obter dados para registro, tempo necessário para captação de dados, referente ao GPS, sem que houvesse sobrecarga na quantidade de dados e assim perda de informações, fez-se necessário testes em rua.

O primeiro teste consistiu no comportamento do OBDII com a central do carro, e verificou-se a possibilidade de se obter os dados desejados através da central de comando do carro. Concluindo o teste, através de voltas por locais em Brasília-DF, notou-se que ocorria o registro e que era possível acessá-lo através do celular com o *backup* dos dados no *Google Drive*, sendo esses dados registrados em planilha, passou-se ao teste utilizando o sistema eletrônico, para registro de velocidade e localização.

Inicialmente o teste utilizando o sistema eletrônico foi feito utilizando apenas o Arduino Uno, computador e GPS, tendo os dados transferidos para uma planilha do *Microsoft Excel*, com o auxílio do software PLX-DAQ. Através desse teste, viu-se a necessidade do aumento do tempo de *loop* que estava programado em 1 segundo. Quando utilizava o tempo de 1 segundo para triangulação do GPS e assim começar o registro de dados, notou-se que a planilha tinha seus dados sobrescritos muito rapidamente, preenchendo assim as linhas da planilha e não tendo pontos com tanta importância para o trabalho, perdendo início do trajeto e guardando apenas parte até a chegada, mesmo utilizando-se 1000 linhas para registro.

Uma possibilidade testada foi retirar de dentro do *loop* referente a triangulação do GPS, a função para escrita dos dados na planilha, colocando que apenas a cada 10 segundo os dados deveriam ser escritos na planilha. Com isso, notou-se que valores estavam sendo perdidos. Os pontos ficavam muito espaçados e quando gerou um mapa desses dados, os alfinetes marcavam pontos que não correspondiam ao trajeto feito e sim a pontos no raio do trajeto. Quando houve a mudança da triangulação para 5 segundos, voltando a função para impressão dos dados para dentro do *loop*, houve um melhor espaçamento das informações. Quando o mapa foi gerado, quase não houve perda no trajeto, sendo apenas 2 alfinetes que correspondiam ao raio do trajeto realizado. O mapa com dados gerados pela planilha com o auxílio do PLX-DAQ, foi gerado de forma manual, apenas para visualização do trajeto.

Após obter resposta satisfatória dos testes com o Arduino, fez-se testes com o sistema eletrônico completo, ainda sendo alimentado pelo computador. Para envio dos dados para o banco de dados em nuvem, precisou do compartilhamento da rede do celular com o *ESP32*. Para que não houvesse problemas de conexão entre celular e *ESP32*, foi necessário criar uma função fora do *Setup* e do *loop*, para que fosse possível se conectar ao *WI-FI* e essa ser chamada no *Setup*. A conexão com a internet utilizando o *ESP32*

precisou um tempo de 2 segundos, menos que isso ele não funcionava. E quando colocava no *loop*, onde ocorre a desserialização dos dados vindos do Arduino e o envio para o banco de dados, passava direto pela função responsável pela conexão com a internet. Assim, não conseguia conexão com o servidor do *Google* para o envio dos dados para registro.

Para registro, foi feito um percurso em uma região administrativa do Distrito Federal, apenas em uma avenida, no espaço de 1 km e depois fazendo o retorno para sua origem. Durante o trajeto, também foram testadas variações de velocidade, visto que o percurso é composto por lombadas (quebra-molas). Com o teste foi possível verificar a funcionalidade do projeto, tendo como resultado os dados guardados na planilha do *Google*, utilizada como banco de dados.

Outros testes foram feitos utilizando o OBDII e o sistema eletrônico, para combinação dos resultados. O sistema eletrônico foi alimentado através do receptor do carro, utilizando um conversor de tensão, devido ao fato do *ESP32* ser alimentado a 3,6V e o Arduino Uno 9V. O trajeto para registro dos dados para o conjunto foi feito em um trajeto maior, utilizando mais de uma via de acesso na mesma região administrativa de Brasília, tendo redução de velocidade e paradas para o que o OBDII pudesse fazer mais de um registro. Enquanto o GPS colhia dados sem interrupção, já que estava sendo alimentado constantemente pela bateria do carro, ele continuava enviando dados para o banco de dados. O OBDII finaliza um registro sempre que ocorre o desligamento do carro, por isso foram feitas paradas durante o trajeto, para que houvesse esses dados para usar em conjunto ao GPS, do contrário o ponto de partida e o de chegada, detectado pelo OBDII naquela ocasião, seriam os mesmos. Tendo apenas registro de distância percorrida, quilometragem, tempo de deslocamento, entre outros, pertencentes ao *scanner*. Com as paradas, a intenção foi mostrar todo o comportamento durante a condução, sendo o OBDII e o sistema eletrônico um complemento de informações um do outro.

2.7 Considerações Finais

Nesta seção foi abordada a forma do recebimento dos dados através do *scanner* ELM327 (9) e a implementação do sistema utilizando o módulo GPS combinado com o Arduíno Uno. Abordou-se também a comunicação entre o Arduíno Uno e o *ESP32*, além da comunicação e o registro com o banco de dados. Na próxima seção serão apresentados os resultados obtidos, a comparação com os dados gerados pelo OBDII(9), além da visualização dos registro com o auxílio do *Google Maps*(12).

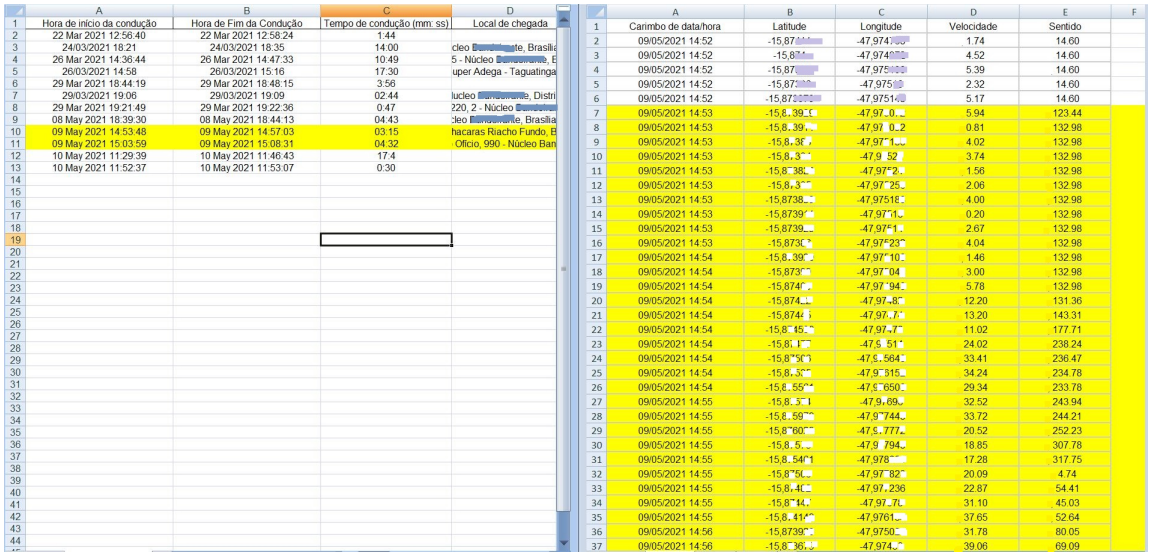
3 Resultados e Discussões

Nesta seção serão apresentados os resultados finais, comparações entre os dois sistemas e validação dos resultados.

3.1 Registro dos Dados do OBDII e do GPS

Os dados obtidos pelo OBDII (9), assim como os obtidos com o sistema composto pelo GPS(12), podem ser vistos na figura 40.

Figura 40 – Dados obtidos pelo OBDII e GPS



	A	B	C	D
1	Hora de início da condução	Hora de Fim da Condução	Tempo de condução (mm: ss)	Local de chegada
2	22 Mar 2021 12:56:40	22 Mar 2021 12:58:24	1:44	
3	24/03/2021 18:21	24/03/2021 18:35	14:00	cleo [redacted]te, Brasília
4	26 Mar 2021 14:36:44	26 Mar 2021 14:47:33	10:49	5 - Núcleo [redacted], E
5	26/03/2021 14:58	26/03/2021 15:16	17:30	uper Adega - Taguatinga
6	29 Mar 2021 18:44:19	29 Mar 2021 18:48:15	3:56	
7	29/03/2021 19:06	29/03/2021 19:09	02:44	lucleo [redacted], Distri
8	29 Mar 2021 19:21:49	29 Mar 2021 19:22:36	0:47	220, 2 - Núcleo [redacted]
9	08 May 2021 18:39:30	08 May 2021 18:44:13	04:43	cleo [redacted]te, Brasília
10	09 May 2021 14:53:48	09 May 2021 14:57:03	03:15	hacaras Riacho Fundo, B
11	09 May 2021 15:03:59	09 May 2021 15:08:31	04:32	Ofício, 990 - Núcleo Ban
12	10 May 2021 11:29:39	10 May 2021 11:46:43	17:4	
13	10 May 2021 11:52:37	10 May 2021 11:53:07	0:30	

	A	B	C	D	E	F
1	Carimbo de data/hora	Latitude	Longitude	Velocidade	Sentido	
2	09/05/2021 14:52	-15,87[redacted]	-47,974[redacted]	1.74	14.60	
3	09/05/2021 14:52	-15,87[redacted]	-47,974[redacted]	4.52	14.60	
4	09/05/2021 14:52	-15,87[redacted]	-47,975[redacted]	5.39	14.60	
5	09/05/2021 14:52	-15,87[redacted]	-47,975[redacted]	2.32	14.60	
6	09/05/2021 14:52	-15,873[redacted]	-47,9751[redacted]	5.17	14.60	
7	09/05/2021 14:53	-15,8[redacted]	-47,97[redacted]	5.94	123.44	
8	09/05/2021 14:53	-15,8[redacted]	-47,97[redacted]	0.81	132.98	
9	09/05/2021 14:53	-15,8[redacted]	-47,97[redacted]	4.02	132.98	
10	09/05/2021 14:53	-15,8[redacted]	-47,9[redacted]	3.74	132.98	
11	09/05/2021 14:53	-15,8[redacted]	-47,97[redacted]	1.56	132.98	
12	09/05/2021 14:53	-15,8[redacted]	-47,97[redacted]	2.06	132.98	
13	09/05/2021 14:53	-15,8738[redacted]	-47,97518[redacted]	4.00	132.98	
14	09/05/2021 14:53	-15,8739[redacted]	-47,97[redacted]	0.20	132.98	
15	09/05/2021 14:53	-15,8739[redacted]	-47,97[redacted]	2.67	132.98	
16	09/05/2021 14:53	-15,873[redacted]	-47,97[redacted]	4.04	132.98	
17	09/05/2021 14:54	-15,8[redacted]	-47,97[redacted]	1.46	132.98	
18	09/05/2021 14:54	-15,873[redacted]	-47,97[redacted]	3.00	132.98	
19	09/05/2021 14:54	-15,874[redacted]	-47,97[redacted]	5.78	132.98	
20	09/05/2021 14:54	-15,874[redacted]	-47,97[redacted]	12.20	131.36	
21	09/05/2021 14:54	-15,874[redacted]	-47,97[redacted]	13.20	143.31	
22	09/05/2021 14:54	-15,8[redacted]	-47,97[redacted]	11.02	177.71	
23	09/05/2021 14:54	-15,8[redacted]	-47,9[redacted]	24.02	238.24	
24	09/05/2021 14:54	-15,8[redacted]	-47,9[redacted]	33.41	236.47	
25	09/05/2021 14:54	-15,8[redacted]	-47,9[redacted]	34.24	234.78	
26	09/05/2021 14:54	-15,8[redacted]	-47,9[redacted]	29.34	233.78	
27	09/05/2021 14:55	-15,8[redacted]	-47,9[redacted]	32.52	243.94	
28	09/05/2021 14:55	-15,8[redacted]	-47,9[redacted]	33.72	244.21	
29	09/05/2021 14:55	-15,8[redacted]	-47,9[redacted]	20.52	252.23	
30	09/05/2021 14:55	-15,8[redacted]	-47,9[redacted]	18.85	307.78	
31	09/05/2021 14:55	-15,8[redacted]	-47,978[redacted]	17.28	317.75	
32	09/05/2021 14:55	-15,8[redacted]	-47,97[redacted]	20.09	4.74	
33	09/05/2021 14:55	-15,8[redacted]	-47,97[redacted]	22.87	54.41	
34	09/05/2021 14:55	-15,8[redacted]	-47,97[redacted]	31.10	45.03	
35	09/05/2021 14:55	-15,8[redacted]	-47,9761[redacted]	37.65	52.64	
36	09/05/2021 14:56	-15,8739[redacted]	-47,9750[redacted]	31.78	80.05	
37	09/05/2021 14:56	-15,8[redacted]	-47,974[redacted]	39.06	69.09	

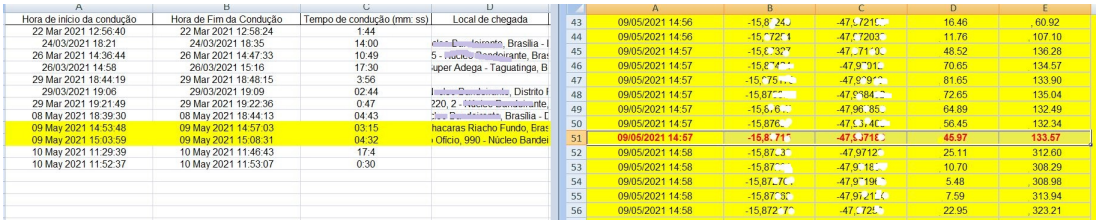
Fonte: Autor

O *scanner* acusou dois registros de chegada, obtidos quando houve desligamento do automóvel, visto que quando se desliga a ignição do carro, o OBDII perde a conexão com a ECU, mantendo apenas a conexão com o celular. Assim ele entende que pode se desconectar e há o registro do local de parada do carro, compartilhado com os dados de localização do celular. O sistema composto pelo GPS(12) continua registrando ininterruptamente e só vai parar de registrar caso perca a conexão com satélites ou alimentação. Ele mantém mais pontos de registros, visto que vai atualizando a cada 5 segundos, tempo necessário para que ele comece a receber os dados dos satélites após a triangulação e possa se comunicar com o servidor para registro dos dados no banco de dados. Sendo assim, a cada 5 segundos, ele imprime os dados de localização recebidos na planilha do Google.

Com a figura 41 é possível ver o comparativo dos dados obtidos utilizando o OBDII e todos os pontos adquiridos através do GPS, havendo um destaque para a velocidade máxima apresentada pelo *scanner*, podendo destacar que as velocidades combinam. Vale

ressaltar que entre o velocímetro presente nos carros há uma margem de tolerância de aproximadamente 3%, e a acurácia do GPS de acordo com o fabricante tem uma precisão de 0,1 m/s(22).

Figura 41 – Dados obtidos pelo OBDII e GPS



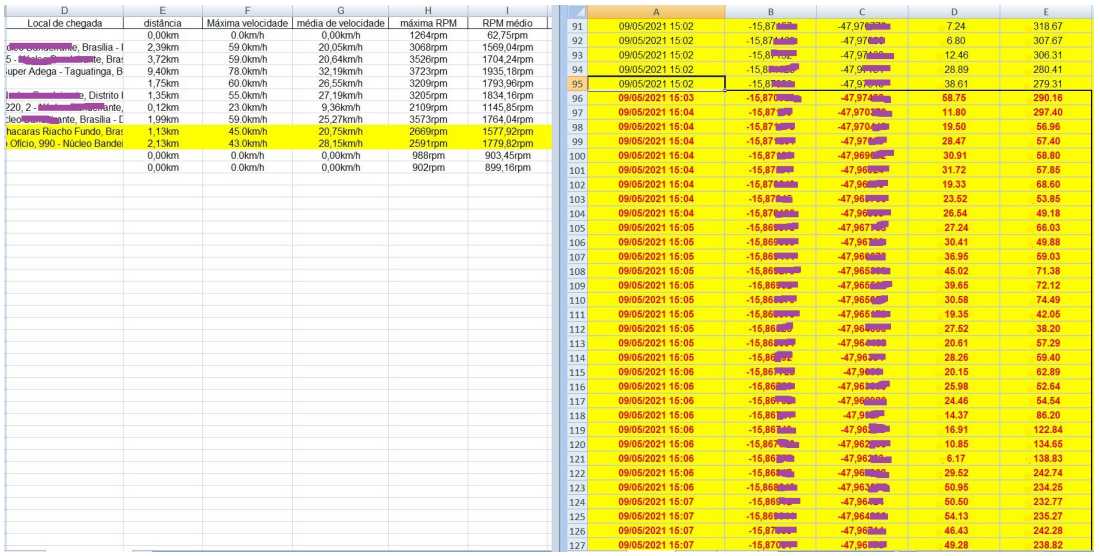
Hora de início da condução	Hora de Fim da Condução	Tempo de condução (mm: ss)	Local de chegada
22 Mar 2021 12:56:40	22 Mar 2021 12:58:24	1:44	
24/03/2021 18:21	24/03/2021 18:35	14:00	[illegible], Brasília - I
26 Mar 2021 14:36:44	26 Mar 2021 14:47:33	10:49	5 - [illegible], Bras
26/03/2021 14:58	26/03/2021 15:16	17:30	uper Adega - Taguatinga, B
29 Mar 2021 18:44:19	29 Mar 2021 18:48:15	3:56	
29/03/2021 19:06	29/03/2021 19:09	02:44	[illegible], Distrito I
29 Mar 2021 19:21:49	29 Mar 2021 19:22:36	0:47	220, 2 - [illegible]
08 May 2021 18:39:30	08 May 2021 18:44:13	04:43	[illegible], Brasília - [
09 May 2021 14:53:48	09 May 2021 14:57:03	03:15	hacaras Riacho Fundo, Bras
09 May 2021 15:03:59	09 May 2021 15:08:31	04:32	Ofício, 990 - Núcleo Bandei
10 May 2021 11:29:39	10 May 2021 11:46:43	17:4	
10 May 2021 11:52:37	10 May 2021 11:53:07	0:30	

	A	B	C	D	E
43	09/05/2021 14:56	-15,87[illegible]	-47,[illegible]21[illegible]	16.46	, 60.92
44	09/05/2021 14:56	-15,[illegible]72[illegible]	-47,[illegible]1203	11.76	107.10
45	09/05/2021 14:57	-15,[illegible]32[illegible]	-47,[illegible]71[illegible]9[illegible]	48.52	136.28
46	09/05/2021 14:57	-15,87[illegible]	-47,9[illegible]91[illegible]	70.65	134.57
47	09/05/2021 14:57	-15,[illegible]75[illegible]	-47,9[illegible]9[illegible]	81.65	133.90
48	09/05/2021 14:57	-15,87[illegible]	-47,9[illegible]84[illegible]	72.65	135.04
49	09/05/2021 14:57	-15,8[illegible]6[illegible]	-47,96[illegible]85[illegible]	64.89	132.49
50	09/05/2021 14:57	-15,876[illegible]	-47,[illegible]3[illegible]4[illegible]	56.45	132.34
51	09/05/2021 14:57	-15,8[illegible]71[illegible]	-47,[illegible]71[illegible]	45.97	133.57
52	09/05/2021 14:58	-15,87[illegible]3[illegible]	-47,9712[illegible]	25.11	312.60
53	09/05/2021 14:58	-15,87[illegible]	-47,9[illegible]18[illegible]	10.70	308.29
54	09/05/2021 14:58	-15,87[illegible]70[illegible]	-47,9[illegible]19[illegible]	5.48	308.98
55	09/05/2021 14:58	-15,87[illegible]8[illegible]	-47,97[illegible]21[illegible]	7.59	313.94
56	09/05/2021 14:58	-15,872[illegible]7[illegible]	-47,[illegible]725[illegible]	22.95	323.21

Fonte: Autor

Por meio da figura 42, vemos que a diferença entre as velocidades obtidas pelo GPS e a mostrada como velocidade máxima registrada pelo OBDII fica mais evidente.

Figura 42 – Dados obtidos pelo OBDII e GPS



Local de chegada	distância	Máxima velocidade	média de velocidade	máxima RPM	RPM médio
	0,00km	0.0km/h	0,00km/h	1264rpm	62,75rpm
[illegible]e, Brasília - I	2,39km	59.0km/h	20,05km/h	3068rpm	1569,04rpm
5 - [illegible]te, Bras	3,72km	59.0km/h	20,64km/h	3526rpm	1704,24rpm
uper Adega - Taguatinga, B	9,40km	78.0km/h	32,19km/h	3723rpm	1935,18rpm
	1,75km	60.0km/h	26,55km/h	3209rpm	1793,96rpm
[illegible]e, Distrito I	1,35km	55.0km/h	27,19km/h	3205rpm	1834,16rpm
220, 2 - [illegible]nte,	0,12km	23.0km/h	9,36km/h	2109rpm	1145,85rpm
[illegible]nte, Brasília - [	1,99km	59.0km/h	25,27km/h	3573rpm	1764,04rpm
hacaras Riacho Fundo, Bras	1,13km	45.0km/h	20,75km/h	2669rpm	1577,92rpm
Ofício, 990 - Núcleo Bandei	2,13km	43.0km/h	28,15km/h	2591rpm	1779,82rpm
	0,00km	0.0km/h	0,00km/h	988rpm	903,45rpm
	0,00km	0.0km/h	0,00km/h	902rpm	899,16rpm

	A	B	C	D	E
91	09/05/2021 15:02	-15,87[illegible]	-47,97[illegible]	7.24	318.67
92	09/05/2021 15:02	-15,87[illegible]	-47,97[illegible]	6.80	307.67
93	09/05/2021 15:02	-15,87[illegible]	-47,97[illegible]	12.46	306.31
94	09/05/2021 15:02	-15,8[illegible]	-47,9[illegible]	28.89	280.41
95	09/05/2021 15:02	-15,8[illegible]	-47,9[illegible]	38.61	279.31
96	09/05/2021 15:03	-15,870[illegible]	-47,974[illegible]	58.75	290.16
97	09/05/2021 15:04	-15,87[illegible]	-47,970[illegible]	11.80	297.40
98	09/05/2021 15:04	-15,871[illegible]	-47,970[illegible]	19.50	56.96
99	09/05/2021 15:04	-15,87[illegible]	-47,97[illegible]	28.47	57.40
100	09/05/2021 15:04	-15,87[illegible]	-47,969[illegible]	30.91	58.80
101	09/05/2021 15:04	-15,87[illegible]	-47,96[illegible]	31.72	57.85
102	09/05/2021 15:04	-15,87[illegible]	-47,96[illegible]	19.33	68.60
103	09/05/2021 15:04	-15,87[illegible]	-47,96[illegible]	23.52	53.85
104	09/05/2021 15:04	-15,87[illegible]	-47,96[illegible]	26.54	49.18
105	09/05/2021 15:05	-15,86[illegible]	-47,967[illegible]	27.24	66.03
106	09/05/2021 15:05	-15,869[illegible]	-47,96[illegible]	30.41	49.88
107	09/05/2021 15:05	-15,86[illegible]	-47,96[illegible]	36.95	59.03
108	09/05/2021 15:05	-15,86[illegible]	-47,965[illegible]	45.02	71.38
109	09/05/2021 15:05	-15,869[illegible]	-47,965[illegible]	39.65	72.12
110	09/05/2021 15:05	-15,86[illegible]	-47,965[illegible]	30.58	74.49
111	09/05/2021 15:05	-15,86[illegible]	-47,965[illegible]	19.35	42.05
112	09/05/2021 15:05	-15,86[illegible]	-47,96[illegible]	27.52	38.20
113	09/05/2021 15:05	-15,86[illegible]	-47,964[illegible]	20.61	57.29
114	09/05/2021 15:05	-15,86[illegible]	-47,96[illegible]	28.26	59.40
115	09/05/2021 15:06	-15,86[illegible]	-47,96[illegible]	20.15	62.89
116	09/05/2021 15:06	-15,86[illegible]	-47,963[illegible]	25.98	52.64
117	09/05/2021 15:06	-15,867[illegible]	-47,96[illegible]	24.46	54.54
118	09/05/2021 15:06	-15,86[illegible]	-47,96[illegible]	14.37	86.20
119	09/05/2021 15:06	-15,86[illegible]	-47,96[illegible]	16.91	122.84
120	09/05/2021 15:06	-15,867[illegible]	-47,962[illegible]	10.85	134.65
121	09/05/2021 15:06	-15,86[illegible]	-47,962[illegible]	6.17	138.83
122	09/05/2021 15:06	-15,86[illegible]	-47,96[illegible]	29.52	242.74
123	09/05/2021 15:06	-15,868[illegible]	-47,963[illegible]	50.95	234.25
124	09/05/2021 15:07	-15,869[illegible]	-47,96[illegible]	50.50	232.77
125	09/05/2021 15:07	-15,86[illegible]	-47,964[illegible]	54.13	235.27
126	09/05/2021 15:07	-15,87[illegible]	-47,96[illegible]	46.43	242.28
127	09/05/2021 15:07	-15,870[illegible]	-47,96[illegible]	49.28	238.82

Fonte: Autor

Através dos dados obtidos foi possível gerar um mapa com o auxílio do *Google Maps* onde é possível notar visualmente a diferença entre os dados obtidos apenas com o OBDII para o quesito localização e quando combinado com os dados obtidos com GPS.

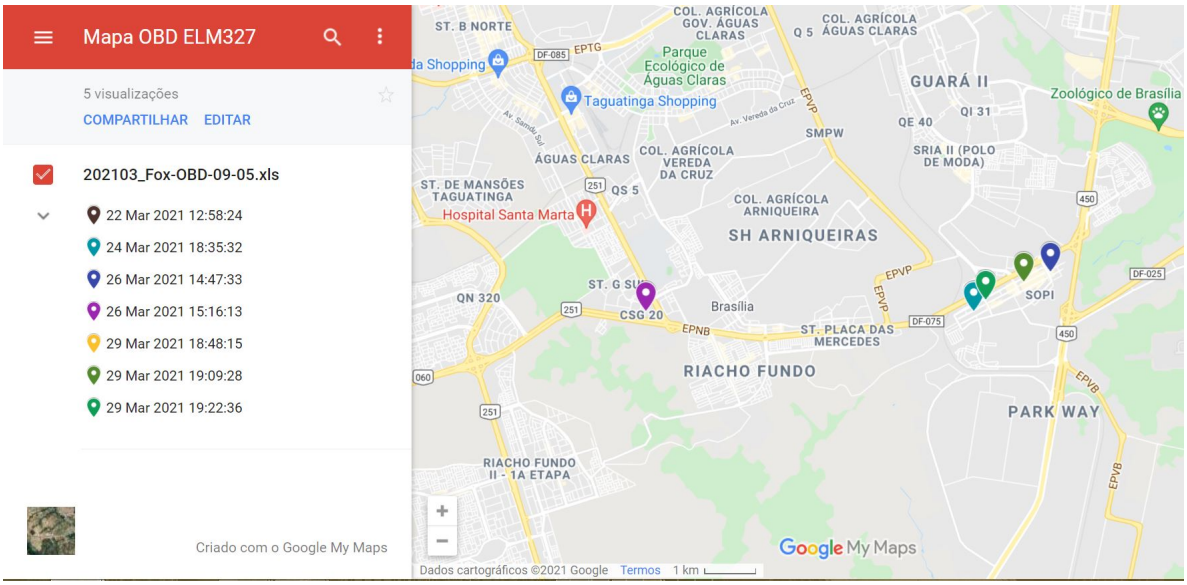
Para gerar o mapa com o auxílio do *Google Maps*, através de uma planilha salva no Google Drive ou no próprio computador, é necessário que se faça o *login* na conta *Google* junto ao *Google Maps*. Através da opção "Criar um Novo Mapa", é possível gerar um mapa com uma planilha ou várias planilhas, criando assim camadas com informações, cada planilha corresponderá a uma camada de informação, se assim for o caso. Para gerar o mapa utilizando os dados obtidos através do OBDII e GPS, utilizou a planilha que havia sido salva no *Google Drive*, por meio do *backup* da pasta do celular referente aos dados do OBDII. Com a planilha gerada com os dados do GPS, foi necessário apenas associar a

planilha ao mapa, devido ao fato da planilha já ser pertencente à plataforma. Na planilha gerada pelo GPS, sempre que esse é ligado, novos dados serão gravados, por tanto, novos alfinetes automaticamente surgirão no mapa. Quanto aos dados obtidos através do OBDII, é necessário que um novo arquivo referente às informações a essa camada seja associado, de forma manual. Visto que o mapa gerado neste trabalho, tem apenas o intuito de mostrar os dados visualmente e não para análise dos dados, portanto não houve a preocupação quanto a parte de automatização para que o mapa fosse gerado. Apenas o registro das informações em nuvem.

O mapa mostrado na figura 43 mostra os dados relacionados ao endereço obtido, quando o carro é estacionado, não sendo possível saber o caminho adotado até chegar ao destino sinalizado. Cada ponto marcado no mapa mostra, conforme as datas registradas, locais onde o carro fez alguma parada.

A figura 44 mostra o mapa obtido através dos registros adquiridos com o auxílio do GPS(12). É possível visualizar todo o trajeto percorrido e o local onde possui uma maior concentração de pontos, mostra locais onde foram feitas paradas. No mapa ficam visíveis 3 pontos com maiores concentrações de alfinetes de localização, corroborando para afirmação.

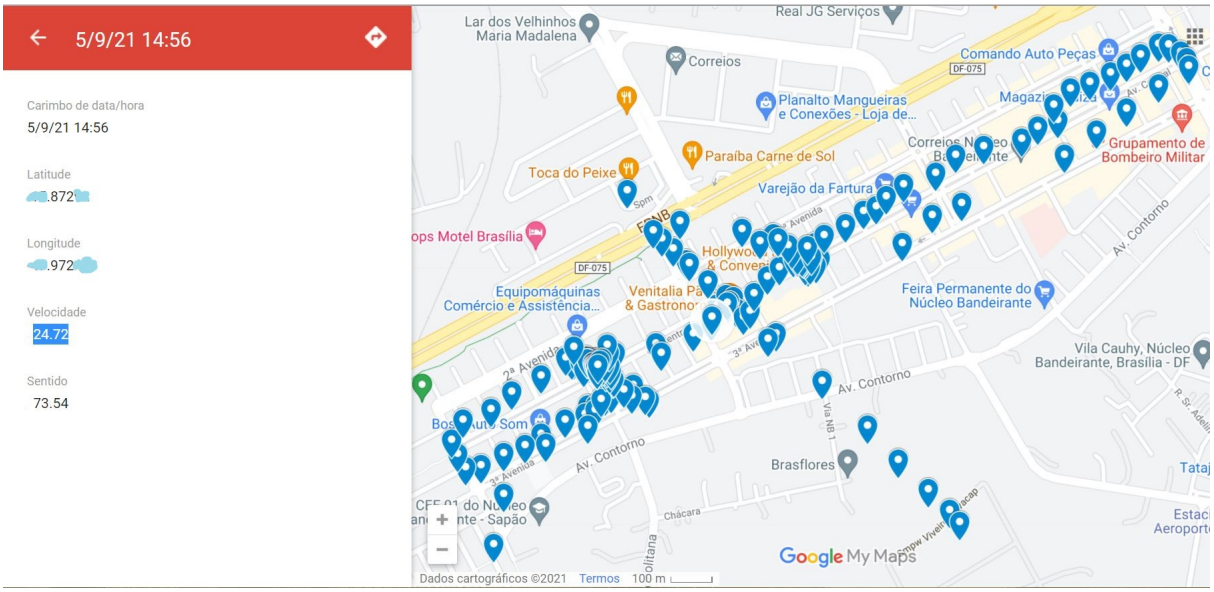
Figura 43 – Mapa obtido com o registro do ELM327



Fonte: Autor

Quando combinado os dois mapas, além da combinação de informações que apenas o *scanner* pode fornecer, como dados da temperatura do óleo de arrefecimento do motor, dados do RPM, velocidade máxima registrada, há a integração com maior precisão de dados referentes a localização do automóvel e uma maior confiabilidade do valores da velocidade durante todo o trajeto e não apenas uma média e uma máxima aproximada.

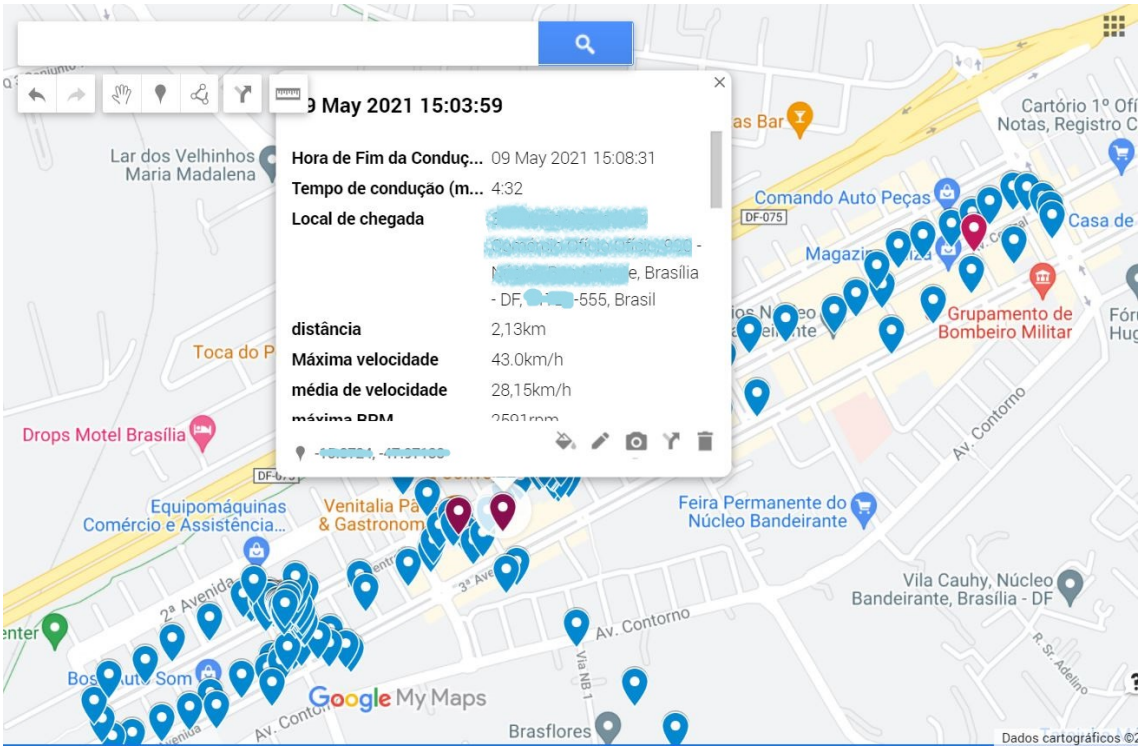
Figura 44 – Mapa obtido com o registro do GPS



Fonte: Autor

A figura 45 mostra o resultado da integração dos dois mapas, sendo os alfinetes em roxo correspondentes aos dados obtidos com o *scanner* enquanto os azuis, em maior número, correspondem aos dados obtidos com o GPS. Quando se analisa ponto a ponto, é possível ter acesso às informações guardadas nas planilhas geradas, fornecendo dados obtidos naquele ponto. Para ter acesso aos dados referentes àquele ponto, basta um clique e tudo o que tiver sido armazenado nas planilhas aparecerão. Com o auxílio de todos os pontos gerados pelo GPS, é possível acompanhar além de todo o trajeto até a chegada da marcação final obtida pelo *scanner* como saber em cada ponto sua velocidade contribuindo para perícias futuras. Na imagem 45 é possível notar que foi selecionado um ponto no mapa e obtido as informações referentes àquele local. No caso o ponto selecionado foi resultado de dados obtidos pelo OBDII, mas qualquer que fosse o alfinete selecionado, obteria-se-ia informações. Ressaltando que alfinetes relacionados ao OBDII, só surgem quando ocorre a parada do carro.

Figura 45 – Mapa combinado com os registros OBD e GPS



Fonte: Autor

3.2 Considerações Finais

Nesta seção foram apresentados os resultados e o registro deles através de tabelas geradas por ambos os sistemas. A tabelas foram combinadas para auxílio de análises futuras, mostrando como um sistema não exclui o outro. Os resultados obtidos foram apresentados a efeito visual em formato de mapas separados e combinando as informações.

4 Conclusão

O objetivo deste trabalho foi gerar uma ferramenta que possa registrar informações capazes de auxiliar peritos a analisar acontecimentos que possam colaborar para a solução de casos envolvendo acidentes, principalmente quando não há sobreviventes. A falta de uma testemunha pode ser crucial para determinar causas. Quando se há dados como os coletados nos testes, é possível estabelecer quais fatores foram determinantes para o ocorrido.

Com a integração dos dados captados através do GPS e armazenados em um banco de dados em nuvem facilita o acesso e sua posterior análise, visto que o mesmo pode ser acessado mesmo em casos em que o sistema tenha sofrido danos. O registro estará sendo gravado e armazenado até o momento em que se perde a conexão.

Os dados mesmo que sejam gerados com coordenadas geográficas, o *Google Maps* permite que esses dados sejam transformados, tornando-se visual, como foi mostrado nas figuras 43, 44, 45. Os dados foram registrados em forma de tabela, utilizando-se as coordenadas geográficas entregue e com auxílio do mapa é possível visualizar os locais por onde o carro passou, assim tendo acesso às informações pertinentes relacionadas ao trajeto além de localização, como a velocidade, o sentido do trajeto, se houve invasão de pista, mostrado após a obtenção do mapa, tendo a possibilidade de gerar o trajeto, do ponto de partida ao de chegada, somado ao *scanner* é possível obter dados da mecânica do carro, que só seriam obtidos após perícia do veículo.

4.1 Dificuldades Encontradas

O trabalho foi desenvolvido durante a pandemia de 2020 que se estendeu ao ano de 2021, que por si já é uma dificuldade. Além disso, descobriu-se a incapacidade da conexão do *scanner* ELM327(9) com o *ESP32*(14), sendo que o intuito era que esses dados também estivessem disponíveis no banco de dados de forma automática sem que houvesse a necessidade de *download* manual por meio do aplicativo, para que só então os dados estivessem disponíveis de forma remota. A falta de acesso aos laboratórios, impediu que pudesse experimentar a viabilidade de uma conexão serial entre o *scanner* e o *ESP32*, assim podendo encaminhar os dados para o banco de dados em conjunto com os dados obtidos pelo GPS. Outra dificuldade se deu por causa da triangulação do GPS, que em dias nublados demorava mais de meia hora, e ainda assim não havia garantia de estabilidade.

4.2 Trabalhos Futuros

Para trabalhos futuros, o próximo passo seria a conexão serial do *ESP32* com o *scanner*, além do uso do GPS, combinar o rádio LORA, para cobertura quando não há sinal de internet. E a implementação de um sistema em que haja o registro de cada caixa-preta, com o acesso ao banco de dados a qual pertence, possibilitando o acesso para análise dos dados.

4.3 Considerações Finais

O projeto se mostrou viável aos objetivos esperados, apesar de algumas intercorrências, provando o seu valor através dos resultados obtidos e mostrados com o auxílio de planilhas e mapas.

Referências

1 VIáRIA, O. N. de S. *90% dos acidentes são causados por falhas humanas*. 2019. Disponível em: <https://www.onsv.org.br/90-dos-acidentes-sao-causados-por-falhas-humanas-alerta-observatorio/>. Acesso em: 3 Out. 2019. Citado na página 12.

2 PRESSE, F.; G1. *OMS DIVULGA RELATÓRIO SOBRE MORTES NO TR NSITO E SUGERE REDUÇÃO DE VELOCIDADES EM ÁREAS URBANAS*. 2018. Disponível em: <https://g1.globo.com/carros/noticia/2018/12/07/oms-divulga-relatorio-sobre-mortes-no-transito-e-sugere-reducao-de-velocidade-em-areas-urbanas.ghtml>. Acesso em: 7 Out. 2019. Citado na página 12.

3 SVS. *10 PRINCIPAIS CAUSA DE MORTALIDADE NO BRASIL*. 2019. Disponível em: <http://svs.aids.gov.br/dantps/centrais-de-conteudos/paineis-de-monitoramento/mortalidade/gbd-brasil/principais-causas/>. Acesso em: 3 Out. 2019. Citado na página 12.

4 BIRAI, J. Caixa preta em automóveis. Universidade São Francisco, 2011. Citado 7 vezes nas páginas 12, 13, 18, 20, 21, 22 e 23.

5 GUIMARAES, A. *Eletrônica Embarcada Automotiva*. [S.l.]: Editora Érica LTDA, 2007. v. 1. Citado 2 vezes nas páginas 15 e 20.

6 FOGAçA, J. R. V. *Funcionamento do Motor de Combustão Interna*. 2021. Disponível em: <https://mundoeducacao.uol.com.br/quimica/funcionamento-motor-combustao-interna.htm>. Acesso em: 10 maio. 2021. Citado na página 16.

7 VARELLA, C. A. A. *PRINCÍPIOS DE FUNCIONAMENTO DOS MOTORES DE COMBUSTÃO INTERNA*. Disponível em: <http://www.ufrrj.br/institutos/it/deng/varella/Downloads/IT154_motores_e_tratores/Aulas/principios_de_funcionamento.pdf>. Acesso em: 6 Nov. 2019. Citado na página 16.

8 THOMSON mte. *Sensor de Rotação, Fase e Velocidade*. 2021. Disponível em: <https://www.mte-thomson.com.br/produtosmte/sensor-de-rotacao/>. Acesso em: 10 maio. 2021. Citado na página 17.

9 ATPARTS. *Sensor de temperatura de arrefecimento*. 2021. Disponível em: <ww.authentichparts.com.br/sensor-de-temperatura-do-liquido-do-arrefecimento-da-injecao-eletronica>. Acesso em: 10 maio. 2021. Citado na página 17.

10 NAKATA. *ENTENDA COMO OS SENSORES PARA CARROS FUNCIONAM NA PRÁTICA*. 2019. Disponível em: <https://blog.nakata.com.br/entenda-como-os-sensores-para-carros-funcionam-na-pratica/>. Acesso em: 10 Nov. 2019. Citado na página 18.

11 RODACARPNEUS. *Sistema de freio e seus componentes*. 2021. Disponível em: <https://rodacarpneus.com.br/sistema-de-freio-e-seus-componentes/>. Acesso em: 10 maio. 2021. Citado na página 18.

12 GRU. *Como funciona a caixa-preta de um avião?* 2016. Disponível em: <http://www.aeroportoguarulhos.net/noticias/mundo/como-funciona-a-caixa-preta-de-um-aviao>. Acesso em: 19 Out. 2019. Citado na página 21.

13 CZERWONKA, M. *Caixa preta para automóveis começa a ser vendida no Brasil*. 2014. Disponível em: <https://www.portaldotransito.com.br/noticias/caixa-preta-para-automoveis-comeca-a-ser-vendida-no-brasil-2/>. Acesso em: 10 maio. 2021. Citado na página 23.

14 ICARROS. *Contran quer "caixa preta" nos carros até 2021*. 2017. Disponível em: <https://www.icarros.com.br/noticias/geral/contran-quer-caixa-preta-nos-carros-ate-2021/23834.html>. Acesso em: 10 maio. 2021. Citado na página 23.

15 DOUTOR-IE. *ELM 327 OBD2: tire suas dúvidas*. 2014. Disponível em: <https://www.doutorie.com.br/blog/elm-327-obd/>. Acesso em: 10 mar. 2021. Citado na página 26.

16 COBLI. *O que é um protocolo OBD2?* 2017. Disponível em: <https://www.cobli.co/blog/o-que-e-protocolo-obd2/>. Acesso em: 10 maio. 2021. Citado na página 26.

17 ZUMPANO, P. *Carro conectado: Entenda um pouco sobre a leitura de dados dos veículos (protocolo OBD)*. 2016. Disponível em: <https://www.dtidigital.com.br/blog/carro-conectado-obd/>. Acesso em: 10 maio. 2021. Citado 2 vezes nas páginas 26 e 27.

18 CARVALHO EDILSON ALVES DE;ARAúJO, P. C. d. *Noções básicas de sistema de posicionamento global GPS*. 2009. Disponível em: <encurtador.com.br/vwHMZ>. Acesso em: 09 maio. 2021. Citado 2 vezes nas páginas 28 e 29.

19 VIDAL, V. *GPS Neo-6M com Arduino – Aprenda a usar*. 2018. Disponível em: <https://blog.eletrogate.com/gps-neo-6m-com-arduino-aprenda-usar/>. Acesso em: 04 maio. 2021. Citado na página 28.

20 UNICAMP. *Como funciona o Sistema de posicionamento global (gps)*. Disponível em: <http://www.ime.unicamp.br/~apmat/o-sistema-gps/>. Acesso em: 10 maio. 2021. Citado na página 28.

21 TERRITORIAL, E. *Satélites de Monitoramento*. 2018. Disponível em: <https://www.embrapa.br/satelites-de-monitoramento/missoes/gps>. Acesso em: 10 Jun. 2021. Citado na página 28.

22 BLOX u. *NEO-6 u-blox 6 GPS Modules Data Sheet*. 2010. Disponível em: <https://components101.com/asset/sites/default/files/component_datasheet/NEO6MV2%20GPS%20Module%20Datasheet.pdf>. Acesso em: 10 maio. 2021. Citado 3 vezes nas páginas 29, 40 e 52.

23 LASTMINUTEENGINEERS. *Interface ublox NEO-6M Módulo GPS com Arduino*. 2020. Disponível em: <https://lastminuteengineers.com/neo6m-gps-arduino-tutorial/>. Acesso em: 10 maio. 2021. Citado na página 29.

24 CARDOSO., M. *O Que É Um Microcontrolador?* 2020. Disponível em: <https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/>. Acesso em: 10 março. 2021. Citado na página 29.

25 LEO, F. C. D. *SISTEMA DE CONTROLE DE ACESSO A RECURSOS COMPARTILHADOS*. 2019. Disponível em: <http://repositorio.poli.ufrj.br/monografias/monopoli10029754.pdf>. Acesso em: 04 maio. 2021. Citado na página 30.

26 BACK, M. *Detalhes sobre o Arduino Uno*. 2014. Disponível em: <http://arduinomais.blogspot.com/2014/06/detalhes-sobre-o-arduino-uno.html>. Acesso em: 04 maio. 2021. Citado na página 30.

27 ESPRESSIF. *ESP32*. 2021. Disponível em: <https://www.espressif.com/en/products/socs/esp32>. Acesso em: 04 maio. 2021. Citado 2 vezes nas páginas 30 e 31.

28 STRAUB, M. G. *ARDUINO IDE – O SOFTWARE PARA GRAVAÇÃO DE CÓDIGOS NO ARDUINO*. 2019. Disponível em: <https://www.usinainfo.com.br/blog/arduino-ide-o-software-para-gravacao-de-codigos-no-arduino/>. Acesso em: 04 maio. 2021. Citado na página 31.

29 BLANCHON, B. *What is ArduinoJson?* 2014. Disponível em: <https://arduinojson.org/about/>. Acesso em: 04 maio. 2021. Citado na página 32.

30 SPARKFUN. *serial communication*. 2016. Disponível em: <https://learn.sparkfun.com/tutorials/serial-communication>. Acesso em: 04 maio. 2021. Citado na página 32.

31 ROBOCORE. *Comparação Entre Protocolos de Comunicação Serial*. 2016. Disponível em: <https://www.robocore.net/tutoriais/comparacao-entre-protocolos-de-comunicacao-serial.html>. Acesso em: 04 maio. 2021. Citado na página 32.

32 SOUZA, I. de. *Banco de dados: saiba o que é, os tipos e a importância para o site da sua empresa*. 2020. Disponível em: <https://rockcontent.com/br/blog/banco-de-dados/>. Acesso em: 04 maio. 2021. Citado na página 33.

33 PANDIT, A. *Registrar dados do sensor de temperatura na planilha do Google usando NodeMCU ESP8266*. 2019. Disponível em: <https://circuitdigest.com/microcontroller-projects/log-temperature-sensor-data-to-google-sheet-using-nodemcu-esp8266>. Acesso em: 04 maio. 2021. Citado na página 33.

34 HART, M. *TinyGPS*. 2013. Disponível em: <https://github.com/mikalhart/TinyGPS>. Acesso em: 22 Out. 2020. Citado na página 38.

Apêndices

APÊNDICE A – Códigos

A.1 Gerar planilha utilizando PLX-DAQ

```
#include <TinyGPS.h>
SoftwareSerial serial1(6, 7); // RX, TX
TinyGPS gps1;
float latitude;
float longitude;
unsigned long idadeInfo;
unsigned short satelites;
unsigned long precisao;
float altitudeGPS;
unsigned long sentido;
float velocidade;
int linha = 0;
int LABEL = 1;
void setup()
serial1.begin(9600);
Serial.begin(9600);
Serial.println("CLEARDATA");
```

limpa o terminal

```
Serial.println("LABEL, TIME, DATE, satelites, Latitude, Longitude, Altitude, Velocidade, Sentido");
void loop()
bool recebido = false; static unsigned long delayPrint;
while (serial1.available())  char cIn = serial1.read(); recebido = (gps1.encode(cIn) || recebido); //Verifica até receber o primeiro sinal dos satelites
if ( (recebido)  ((millis() - delayPrint) > 5000) )  //Mostra apenas após receber o
```

primeiro sinal. Após o primeiro sinal, mostra a cada 5 segundos. delayPrint = millis();

gps1.f_get_position(&latitude, &longitude, &idadeInfo);

if (latitude != TinyGPS::GPS_INVALID_F_ANGLE)

if (longitude != TinyGPS::GPS_INVALID_F_ANGLE)

if (idadeInfo != TinyGPS::GPS_INVALID_AGE)

altitudeGPS = gps1.f_altitude();

if ((altitudeGPS != TinyGPS::GPS_INVALID_ALTITUDE) && (altitudeGPS != 1000000))

//velocidade

velocidade = gps1.f_speed_kmph(); //km/h

//sentito (em centesima de graus)

sentido = gps1.course();

//satelites e precisão

satelites = gps1.satellites(); precisao = gps1.hdop();

if (satelites != TinyGPS::GPS_INVALID_SATELLITES)

if (precisao ¯TinyGPS::GPS_INVALID_HDOP)

linha++;

Serial.print("DATA,TIME,DATE,");

Serial.print(satelites);

Serial.print(",");

Serial.print(latitude,6);

Serial.print(",");

Serial.print(longitude,6);

Serial.print(",");

Serial.print(altitudeGPS);

Serial.print(",");

Serial.print(velocidade,2);

Serial.print(",");

Serial.println(float(sentido) / 100, 2);

if(linha > 300)

linha = 0;

Serial.println("ROW,SET,2");

#include <SoftwareSerial.h>

#include <TinyGPS.h>

SoftwareSerial serial1(6, 7); // RX, TX

TinyGPS gps1;

float latitude;

float longitude;

unsigned long idadeInfo;

unsigned short satelites;

unsigned long precisao;

float altitudeGPS;

unsigned long sentido;

float velocidade;

int linha = 0;

int LABEL = 1;

void setup()

serial1.begin(9600);

Serial.begin(9600);

Serial.println("CLEARDATA");
limpa o terminal

Serial.println("LABEL, TIME, DATE, satelites, Latitude, Longitude, Altitude, Velocidade, Sentido");

void loop()

bool recebido = false; static unsigned long delayPrint;

while (serial1.available()) char cIn = serial1.read(); recebido = (gps1.encode(cIn) || recebido); //Verifica até receber o primeiro sinal dos satelites

if ((recebido) ((millis() - delayPrint) > 5000)) //Mostra apenas após receber o primeiro sinal. Após o primeiro sinal, mostra a cada 5 segundos. delayPrint = millis();

gps1.f_get_position(&latitude, &longitude, &idadeInfo);

if (latitude != TinyGPS::GPS_INVALID_F_ANGLE)

```
if (longitude != TinyGPS::GPS_INVALID_F_ANGLE)
if (idadeInfo != TinyGPS::GPS_INVALID_AGE)
altitudeGPS = gps1.f_altitude();
if ((altitudeGPS != TinyGPS::GPS_INVALID_ALTITUDE) && (altitudeGPS
!= 1000000))
//velocidade
velocidade = gps1.f_speed_kmph(); //km/h
//sentito (em centesima de graus)
sentido = gps1.course();
//satelites e precisão
satelites = gps1.satellites(); precisao = gps1.hdop();
if (satelites != TinyGPS::GPS_INVALID_SATELLITES)
if (precisao ~TinyGPS::GPS_INVALID_HDOP)
linha++;
Serial.print("DATA,TIME,DATE,");
Serial.print(satelites);
Serial.print(",");
Serial.print(latitude,6);
Serial.print(",");
Serial.print(longitude,6);
Serial.print(",");
Serial.print(altitudeGPS);
Serial.print(",");
Serial.print(velocidade,2);
Serial.print(",");
Serial.println(float(sentido) / 100, 2);
if(linha > 300)
linha = 0;
Serial.println("ROW,SET,2");
```

A.2 Código Arduíno Uno

```
#include <TinyGPS.h>
#include <ArduinoJson.h>
SoftwareSerial serial1(6, 7); // RX, TX
TinyGPS gps1;
SoftwareSerial linkSerial(10,11); // RX,TX - entradas UART arduino
float latitude, longitude;
unsigned long idadeInf; //idade da informação
int ano;
byte mes, dia, hora, minuto, segundo, centesimo;
float velocidade;
unsigned long sentido;
void setup()
linkSerial.begin(4800); // taxa da dados baixa para reduzir taxa de erros.
serial1.begin(9600);
Serial.begin(115200);
Serial.println("Aguardando sinal dos satelistes...");
void loop()
bool recebido = false;
static unsigned long delayPrint;
while (serial1.available())
char cIn = serial1.read();
recebido = (gps1.encode(cIn) || recebido); //Verifica até receber o primeiro sinal dos satelites
if ( (recebido) && ((millis() - delayPrint) > 5000) )  //Mostra apenas após receber o primeiro sinal. Após o primeiro sinal, mostra a cada 5 segundos.
delayPrint = millis();
Serial.println("--------------------");
gps1.f_get_position(&latitude, &longitude);
```

if (latitude != TinyGPS::GPS_INVALID_F_ANGLE)

Serial.print("Latitude: ");

Serial.println(latitude, 6);

if (longitude != TinyGPS::GPS_INVALID_F_ANGLE)

Serial.print("Longitude: ");

Serial.println(longitude, 6);

//Dia e Hora

gps1.crack_datetime(&ano, &mes, &dia, &hora, &minuto, segundo, &centesimo, &idadeInf); hora = hora - 3; //corrigindo manualmente o horário do meridiano de greenwich para o brasil.

Serial.print("Data (GMT): ");

Serial.print(dia);

Serial.print("/");

Serial.print(mes);

Serial.print("/");

Serial.println(ano);

Serial.print("Horario (UCT): ");

Serial.print(hora);

Serial.print(":");

Serial.print(minuto);

Serial.print(":");

Serial.println(segundo);

//velocidade velocidade = gps1.f_speed_kmph(); //km/h

Serial.print("Velocidade (km/h): ");

Serial.println(velocidade, 2); //Conversão para Km/h

//sentito (em centesima de graus)

sentido = gps1.course();

Serial.print("Sentido (grau): ");

Serial.println(float(sentido) / 100, 2);

// criando documento JSON StaticJsonDocument<200> doc;

```
doc["latitude"] = latitude;
doc["longitude"] = longitude;
doc["velocidade"] = velocidade;
doc["sentido"] = sentido;
doc["dia"] = dia;
doc["mes"] = mes;
doc["ano"] = ano;
doc["hora"] = hora;
doc["minuto"] = minuto;
doc["segundo"] = segundo;
//Enviando os dados do documento para esp32
serializeJson(doc, linkSerial);
#include <SoftwareSerial.h>
#include <TinyGPS.h>
#include <ArduinoJson.h>
SoftwareSerial serial1(6, 7); // RX, TX
TinyGPS gps1;
SoftwareSerial linkSerial(10,11); // RX,TX - entradas UART arduino
float latitude, longitude;
unsigned long idadeInf; //idade da informação
int ano;
byte mes, dia, hora, minuto, segundo, centesimo;
float velocidade;
unsigned long sentido;
void setup()
linkSerial.begin(4800); // taxa da dados baixa para reduzir taxa de erros.
serial1.begin(9600);
Serial.begin(115200);
Serial.println("Aguardando sinal dos satelistes...");
void loop()
```

bool recebido = false;

static unsigned long delayPrint;

while (serial1.available())

char cIn = serial1.read();

recebido = (gps1.encode(cIn) || recebido); //Verifica até receber o primeiro sinal dos satelites

if ((recebido) && ((millis() - delayPrint) > 5000)) //Mostra apenas após receber o primeiro sinal. Após o primeiro sinal, mostra a cada 5 segundos.

delayPrint = millis();

Serial.println("————————————————————-");

gps1.f_get_position(&latitude, &longitude);

if (latitude != TinyGPS::GPS_INVALID_F_ANGLE)

Serial.print("Latitude: ");

Serial.println(latitude, 6);

if (longitude != TinyGPS::GPS_INVALID_F_ANGLE)

Serial.print("Longitude: ");

Serial.println(longitude, 6);

//Dia e Hora

gps1.crack_datetime(&ano, &mes, &dia, &hora, &minuto, segundo, &centesimo, &idadeInf); hora = hora - 3; //corrigindo manualmente o horário do meridiano de greenwich para o brasil.

Serial.print("Data (GMT): ");

Serial.print(dia);

Serial.print("/");

Serial.print(mes);

Serial.print("/");

Serial.println(ano);

Serial.print("Horario (UCT): ");

Serial.print(hora);

Serial.print(":");

Serial.print(minuto);

```
Serial.print(":");
Serial.println(segundo);
//velocidade velocidade = gps1.f_speed_kmph(); //km/h
Serial.print("Velocidade (km/h): ");
Serial.println(velocidade, 2); //Conversão para Km/h
//sentito (em centesima de graus)
sentido = gps1.course();
Serial.print("Sentido (grau): ");
Serial.println(float(sentido) / 100, 2);
// criando documento JSON StaticJsonDocument<200> doc;
doc["latitude"] = latitude;
doc["longitude"] = longitude;
doc["velocidade"] = velocidade;
doc["sentido"] = sentido;
doc["dia"] = dia;
doc["mes"] = mes;
doc["ano"] = ano;
doc["hora"] = hora;
doc["minuto"] = minuto;
doc["segundo"] = segundo;
//Enviando os dados do documento para esp32
serializeJson(doc, linkSerial);
```

A.3 Código ESP32

#include "ArduinoJson.h"

#include <SoftwareSerial.h>

#include <WiFi.h>

#include <WiFiClientSecure.h>

SoftwareSerial linkSerial(13,15); //RX TX

WiFiClientSecure client;//Cria um cliente seguro (para ter acesso ao HTTPS)

String textFix = "GET //forms/d/e/1FAIpQLSe4VL0zodKgk4xHdNkY7lF0vPbrH8G9CYaAz
h6JmfQjnt-Q/formResponse?ifq&entry.=&entry.=&entry.=";

//Essa String sera uma auxiliar contendo o link utilizado pelo GET, para nao precisar ficar re-escrevendo toda hora

void wifi()

WiFi.mode(WIFI_STA);

WiFi.begin("Jennifer Cavalcante", "senha");//Conecta na rede

delay(2000); //Espera um tempo para se conectar no WiFi

void setup()

Serial.begin(115200);

linkSerial.begin(4800);

wifi();

//Tenta se conectar ao servidor do Google docs na porta 443 (HTTPS)

void loop()

if(linkSerial.available())

StaticJsonDocument<300> doc;

DeserializationError err = deserializeJson(doc, linkSerial);

if(err == DeserializationError::Ok)

//Se tudo certo, printa a leitura dos valores recebidos do arduino.

auto latitude = doc["latitude"].as<float>();

auto longitude = doc["longitude"].as<float>();

auto velocidade = doc["velocidade"].as<float>();

```
auto sentido = doc["sentido"].as<float>();
// Para visualização no monitor Serial
Serial.println("———————");
Serial.print("Latitude ");
Serial.println(latitude);
Serial.print("Longitude ");
Serial.println(longitude);
Serial.print("Velocidade ");
Serial.println(velocidade);
Serial.print("Sentido ");
Serial.println(sentido);
// Inicio do envio de dados para o google sheet
client.setInsecure();
if (client.connect("docs.google.com", 443) == 1)
//Atribuimos a String auxiliar na nova String que sera enviada
String toSend = textFix + "&entry.="+ latitude + "&entry.=" + longitude +
"&entry.=" + velocidade + "&entry.=" + sentido;
toSend += "&submit=Submit HTTP/1.1";
client.println(toSend);
//Envia o GET ao servidor- client.println("Host: docs.google.com");
client.println();
client.stop();//Encerra a conexao com o servidor
Serial.println("Dados enviados."); //Mostra no monitor que foi enviado
else
Serial.println("Erro ao se conectar");
//Se nao for possivel conectar no servidor, ira avisar no monitor.
else
// caso não consiga retornar os dados do JSON do arduino
Serial.print("deserializeJson() returned ");
Serial.println(err.c_str());
```

```
while(linkSerial.available()>0) linkSerial.read();

#include <Arduino.h>

#include "ArduinoJson.h"

#include <SoftwareSerial.h>

#include <WiFi.h>

#include <WiFiClientSecure.h>

SoftwareSerial linkSerial(13,15); //RX TX

WiFiClientSecure client;//Cria um cliente seguro (para ter acesso ao HTTPS)

String textFix = "GET //forms/d/e/1FAIpQLSe4VL0zodKgk4xHdNkY7lF0vPbrH8G9CYaAz
h6JmfQjnt-Q/formResponse?ifq&entry.=&entry.=&entry.=&entry.=";

//Essa String sera uma auxiliar contendo o link utilizado pelo GET, para nao
precisar ficar re-escrevendo toda hora

void wifi()

WiFi.mode(WIFI_STA);

WiFi.begin("Jennifer Cavalcante", "senha");//Conecta na rede

delay(2000); //Espera um tempo para se conectar no WiFi

void setup()

Serial.begin(115200);

linkSerial.begin(4800);

wifi();

//Tenta se conectar ao servidor do Google docs na porta 443 (HTTPS)

void loop()

if(linkSerial.available())

StaticJsonDocument<300> doc;

DeserializationError err = deserializeJson(doc, linkSerial);

if(err == DeserializationError::Ok)

//Se tudo certo, printa a leitura dos valores recebidos do arduino.

auto latitude = doc["latitude"].as<float>();

auto longitude = doc["longitude"].as<float>();

auto velocidade = doc["velocidade"].as<float>();
```

```
auto sentido = doc["sentido"].as<float>();
// Para visualização no monitor Serial
Serial.println("———————");
Serial.print("Latitude ");
Serial.println(latitude);
Serial.print("Longitude ");
Serial.println(longitude);
Serial.print("Velocidade ");
Serial.println(velocidade);
Serial.print("Sentido ");
Serial.println(sentido);
// Inicio do envio de dados para o google sheet
client.setInsecure();
if (client.connect("docs.google.com", 443) == 1)
//Atribuimos a String auxiliar na nova String que sera enviada
String toSend = textFix + "&entry.="+ latitude + "&entry.=" + longitude +
"&entry.=" + velocidade + "&entry.=" + sentido;
toSend += "&submit=Submit HTTP/1.1";
client.println(toSend);
//Envia o GET ao servidor- client.println("Host: docs.google.com");
client.println();
client.stop();//Encerra a conexao com o servidor
Serial.println("Dados enviados."); //Mostra no monitor que foi enviado
else
Serial.println("Erro ao se conectar");
//Se nao for possivel conectar no servidor, ira avisar no monitor.
else
// caso não consiga retornar os dados do JSON do arduino
Serial.print("deserializeJson() returned ");
Serial.println(err.c_str());
```

while(linkSerial.available()>0) linkSerial.read();

Obs: Os valores correspondentes ao &entry.foram ocultados, visto que eles funcionam como endereço específico do banco de dados, onde os valores do módulo devem ser gravados.