UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

RODRIGO CARAMASCHI VERNIZZI

Desenvolvimento de dispositivo para aquisição de dados da rede CAN de um veículo e transmissão através da Internet

São Carlos
2024

RODRIGO CARAMASCHI VERNIZZI

Desenvolvimento de dispositivo para aquisição de dados da rede CAN de um veículo e transmissão através da Internet

Monografia apresentada ao Curso de Engenharia Elétrica com ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Engenheiro Eletricista

Orientador: Prof. Dr. Marco Henrique Terra

Primeira Versão

São Carlos

2024

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Vernizzi, Rodrigo
V538d Desenvolvimento de dispositivo para aquisição de dados da rede CAN de um veículo e transmissão através da Internet / Rodrigo Vernizzi; orientador Marco Henrique Terra. São Carlos, 2024.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2024.

1. CAN Bus. 2. Telemetria. 3. Aquisição de Dados. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Rodrigo Caramaschi Vernizzi

Título: "Desenvolvimento de dispositivo para aquisição de dados da rede CAN de um veículo e transmissão através da Internet"

Trabalho de Conclusão de Curso defendido e aprovado em 30 / 10 / 2024**,**

com NOTA 9,0 (nove, zero), **pela Comissão Julgadora:**

Prof. Titular Marco Henrique Terra - Orientador - SEL/EESC/USP

Prof. Associado Valdir Grassi Junior - SEL/EESC/USP

Mestre Nícolas dos Santos Rosa - Doutorando EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior

Este trabalho é dedicado à minha família,
aos meus amigos e a minha namorada,
que sempre me apoiaram e incentivaram.
Em especial ao meu pai e minha mãe,
que com muito amor e carinho,
me ajudaram a chegar onde cheguei.

AGRADECIMENTOS

Gostaria de agradecer à Universidade de São Paulo (USP) e a equipe EESC USP Baja, pela oportunidade de realizar e publicar esta pesquisa e por fornecer os equipamentos necessários.

“Mesmo quando tudo parece desabar,
cabe a mim decidir entre rir ou chorar,
ir ou ficar, desistir ou lutar; porque descobri,
no caminho incerto da vida,
que o mais importante é o decidir.”
Coralina, Cora

RESUMO

VERNIZZI, R. C. Título: **Desenvolvimento de dispositivo para aquisição de dados da rede CAN de um veículo e transmissão através da Internet**. 2024. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

O presente trabalho descreve o desenvolvimento de um dispositivo de baixo custo para a aquisição de dados da rede CAN de veículos e sua transmissão pela internet, projetado com base no microcontrolador ESP32, juntamente com um sistema de armazenamento local, em um cartão SD, garantindo a integridade dos dados e preservando-os para análises posteriores. Além disso, foi feita uma interface para o usuário, permitindo assim, o monitoramento em tempo real das condições do automóvel, exibindo dados como velocidade e RPM. Testado em um carro da equipe EESC USP Baja, o sistema demonstrou robustez e grande eficiência em seu papel. Este dispositivo e todo o sistema relacionado, mostraram-se uma solução viável e acessível para a telemetria, com aplicações diversas, como diagnósticos preventivos e otimização do desempenho.

Palavras-chave: *CAN Bus*. Telemetria. ESP32. Aquisição de dados. Internet das Coisas (IoT).

ABSTRACT

VERNIZZI, R. C. Title:**Development of a device for acquiring data from a vehicle's CAN network and transmitting it over the Internet** 2024. Monograph (Course Conclusion Paper) - São Carlos School of Engineering, University of São Paulo, São Carlos, 2024.

This work describes the development of a low-cost device for acquiring data from the CAN network of vehicles and transmitting it over the Internet, designed using the ESP32 microcontroller. Together with a local storage system on an SD card, it guarantees the integrity of the data and preserves it for later analysis. In addition, a user interface was designed, allowing real-time monitoring of the car's condition, displaying data such as speed and RPM. Tested in a car from EESC USP Baja team, the system proved to be robust and highly efficient in its role. This device and the entire related system proved to be a viable and affordable solution for telemetry, with diverse applications such as preventive diagnostics and performance optimization.

Keywords: CAN Bus. Telemetry. ESP32. Data acquisition. Internet of Things (IoT).

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CAN	*Controller Area Network*
OBD	*On-Board Diagnostics*
IoT	*Internet of Things*
MQTT	*Message Queuing Telemetry Transport*
RPM	Rotação Por Minuto
STFT	*Short-Term Fuel Trim*
DTCs	*Diagnostic Trouble Codes*
PCB	*Printed Circuit Board*

SUMÁRIO

1 INTRODUÇÃO

O setor automotivo está em constante evolução, impulsionado pela crescente demanda por veículos conectados e autônomos. Essa mudança exige a coleta e análise de grandes volumes de dados em tempo real para diversos fins, como monitoramento do status do veículo, dirigibilidade segura e eficiente, desenvolvimento de veículos autônomos e serviços telemáticos. Nos veículos atuais, a rede CAN (*Controller Area Network*), desempenha um papel crucial como rede de comunicação serial padronizada para sistemas embarcados. Sua robustez, confiabilidade, baixo custo e capacidade de transmitir vários sinais em um único barramento a tornam essencial para diversas funções (Robert Bosch GmbH, 2012).

Contudo, a maioria dos carros, principalmente os populares, não permitem ao usuário um acesso facilitado aos dados de seu veículo, fazendo com que tenha menos informações a respeito da saúde e condições do veículo. Alterar o sistema embarcado do carro para que esse acesso seja facilitado é inviável devido a quebra de garantia, ao nível de complexidade e ao fato de que o usuário se torna dependente de um serviço. Uma das maneiras de contornar este problema e através do *CAN Sniffer*, um dispositivo conectado à porta OBD do veículo que consegue captar os dados da rede CAN (ELECTRONICS, 2024).

O CAN *sniffer* é uma solução prática e eficiente para a coleta de dados de veículos, permitindo assim a transmissão destes dados para diversos usos como, a manutenção preventiva e diagnóstico remoto (TIETOEVRY, 2023). Este dispositivo lê os dados brutos da rede CAN e os transmite para um servidor central, onde podem ser analisados para os fins desejados.

Ademais, a transmissão pela internet destes dados pode possibilitar outras funções para a telemetria automotiva (LI; LIU; LUO, 2008). Os dados podem ser utilizados por fabricantes de veículos para melhorar os projetos futuros, por seguradoras para ajustar políticas de seguros baseadas no comportamento de condução e por desenvolvedores de aplicativos para criar novas funcionalidades que aumentem a segurança e a conveniência dos motoristas.

Já no contexto de automóveis autônomos, este dispositivo pode ser utilizado para uma comunicação entre os veículos com uma infraestrutura inteligente, contribuindo para a construção de um ecossistema de transporte mais seguro e eficiente.

1.1 Objetivo Geral

Este trabalho visa explorar e desenvolver um dispositivo de baixo custo e de fácil utilização para aquisitar os dados da rede CAN de um veículo e transmiti-los pela internet.

1.2 Objetivos Específicos

Para atingir o objetivo geral é necessário, cumprir os seguintes objetivos específicos:

- Projetar e manufaturar um *hardware* necessário para coletar os dados;
- Desenvolver um *software* para salvar e transmitir os dados;
- Criar e estruturar um banco de dados na nuvem;
- Desenvolver um aplicativo web, iOS e Android que disponibilizará os dados em tempo real para o usuário.

2 REVISÃO BIBLIOGRÁFICA

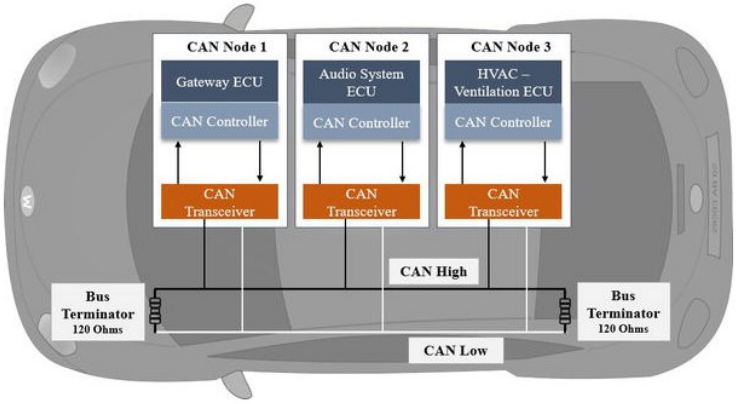
2.1 Considerações Iniciais

Nesta revisão bibliográfica, busca-se explorar estudos e soluções que envolvem a utilização de microcontroladores para aquisição de dados veiculares, com transmissão desses dados via internet, Internet das Coisas, do inglês Internet of Things (IoT). Assim, é de interesse identificar abordagens e métodos eficientes para captar informações da rede CAN de um veículo e enviá-las para uma plataforma de armazenamento em nuvem, proporcionando acesso remoto e em tempo real aos dados. Além disso, a pesquisa visa avaliar diferentes microcontroladores, considerando suas capacidades de comunicação sem fio, custos e compatibilidade com os requisitos do projeto.

2.2 *CAN Bus* veicular

CAN Bus é um protocolo de barramento serial adotado vastamente pela indústria automotiva. Atualmente esta tecnologia é adotada para redução dos chicotes assim como aprimorar o controle do veículo (SALUNKHE; KAMBLE; JADHAV, 2016). Atualmente em um veículo existem múltiplas redes CAN, conectadas no mesmo barramento, cada uma dessas redes é denominada de nós. A Figura 1 ilustra como é feita esta conexão.

Figura 1 – Conexão dos múltiplos nós de uma rede *CAN Bus*



Fonte: BOLAND MORGAN I. BURGETT (2021)

Nos carros atuais é possível localizar um conector que fornece acesso aos dados do veículo, este conector é comumente referido como OBD-II, (*On Board On Board Diagnostics*) e neste está presente os terminais da rede CAN.

2.2.1 Arquitetura física

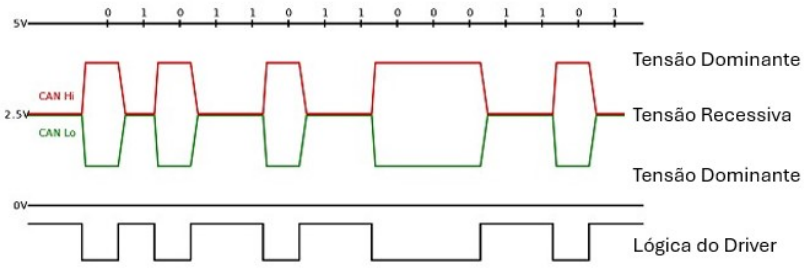
A arquitetura física de uma rede CAN é composta por 2 canais, um *CAN High* e outro para *CAN Low*, sendo assim é transmitido um sinal diferencial, com uma lógica de dois

estados:

- Dominante - Nível 0 - 3V5;
- Recessivo - Nível 1 - 2V5.

Como pode ser observado na Figura 2.

Figura 2 – Sinais Elétricos da Transmissão CAN



Fonte: (EMBARCADOS, 2020)

Com essa imagem é possível perceber que ao ser transmitido o bit 1, o estado é recessivo, ou seja, as tensões são iguais, já quando esta sendo transmitido o bit 0, as tensões são diferentes, sendo assim, o estado é dominante. Esta representação ajuda a compreender como o protocolo CAN utiliza a diferença de tensão entre CAN High e CAN Low para transmitir dados binários. Esse método torna a comunicação mais robusta e resistente a interferências, ideal para ambientes como os sistemas automotivos. Além da robustez contra interferências, o protocolo CAN apresenta detecção de erros e sistema de prioridade de mensagem, tornando-a ainda mais ideal para o sistema automotivo.

Porém uma das desvantagens que a CAN apresenta neste cenário é uma taxa de transmissão limitada em 1Mbps, para contornar esta barreira existe a tecnologia CAN FD (*Flexible Data Rate*), que com ela é possível ajustar a taxa de transmissão para até 5 Mbps.

Também existe o protocolo FlexRay, o qual possui uma taxa de transferência mais elevada, contudo também apresenta maior complexidade e custo.

2.3 Integração de Internet das Coisas (IoT) no monitoramento veicular

Com o objetivo de reduzir o número de acidentes de trânsito relacionados à saúde do veículo, Naurudin *et al*. (2023) realizaram um estudo para entender como a integração do IoT, do inglês, *Internet of Things*, pode tornar o monitoramento veicular mais útil e prático para os donos de automóveis. Para o projeto foi escolhido o protocolo MQTT (*Message Queuing Telemetry Transport*) devido à sua eficiência na transmissão de dados em redes com largura de banda limitada e alta latência, características comuns em sistemas de IoT móveis (NURUDIN; ZARLIS, 2023). O sistema desenvolvido coleta dados críticos do veículo,

como RPM (rotação por minuto), temperatura do motor, e os transmite para um servidor no qual podem ser acessados por uma interface web. Essa abordagem permite a criação de um sistema de monitoramento contínuo que fornece alertas imediatos caso alguma situação adversa seja detectada, promovendo uma manutenção preventiva e aumentando a segurança operacional dos carros (NURUDIN; ZARLIS, 2023). Em suma, este estudo enfatiza a importância da conectividade IoT para o monitoramento em tempo real das condições dos veículos, destacando o uso do protocolo MQTT como uma solução eficiente para a transmissão de dados em redes móveis.

2.4 Monitoramento de consumo de combustível e eficiência operacional com sensores OBD-II

Rimpas *et al.* (2020) tinha como objetivo principal investigar a utilização de sensores OBD-II para monitorar os parâmetros operacionais e de consumo de combustível em veículos, com o intuito de aprimorar a eficiência do consumo e reduzir as emissões de gases tóxicos, utilizando tecnologias acessíveis e disponíveis, como o OBD-II e o barramento CAN Bus (RIMPAS; PAPADAKIS; SAMARAKOU, 2020). O estudo se concentrou na seleção de parâmetros cruciais para o desempenho e a eficiência do veículo: temperatura do líquido de arrefecimento do motor, a razão de oxigênio no escape (sonda *lambda*), a taxa de fluxo de ar, a velocidade do veículo e o ajuste de combustível de curto prazo (STFT), do inglês *Short-Term Fuel Trim*. A coleta de dados foi realizada utilizando um *scanner* OBD-II (ELM 327 Mini Bluetooth) conectado ao veículo durante um trajeto de 5 km, incluindo vias com diferentes condições de tráfego (alta e baixa densidade). Os dados coletados foram transmitidos via Bluetooth para um notebook e registrados em tempo real utilizando o software *ScanMaster*.

2.5 Telemetria e aquisição de dados de veículos

A aquisição de dados de sensores presentes em um veículo assim como a análise deles, telemetria, são essenciais tanto para o público geral quanto para equipes de competição (CHANDIRAMANI; BHANDARI; HARIPRASAD, 2014). Monitorar as condições de direção tem como objetivo detectar parâmetros do veículo em si, condições do automóvel, assim como do ambiente ao redor deles, como por exemplo, clima, congestionamento, perfil da superfície (TAYLOR NATHAN GRIFFITHS; GELENCSER, 2016).

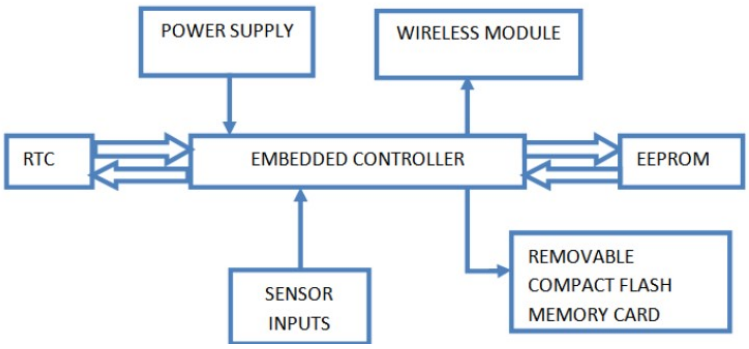
Métodos para aquisição dos dados

Para conseguir aquisitar os dados, é necessário definir um microprocessador que seja capaz de aquisitar os dados dos sensores e salvá-los ou transmiti-los para uma central.

Atmega 168

Uma pesquisa direcionada para aquisição de dados de um veículo de corrida, utilizando o microcontrolador da Atmega168 (CHANDIRAMANI; BHANDARI; HARIPRASAD, 2014), devido a sua facilidade de programação e pela quantidade de entradas ADC disponíveis no microcontrolador. O diagrama da Figura 3, representa o sistema de aquisição adotado

Figura 3 – Diagrama de Blocos



Fonte: CHANDIRAMANI; BHANDARI; HARIPRASAD (2014)

Como pode ser visto, no trabalho mencionado, os sensores enviam o sinal diretamente para a unidade de controle, que os salva e transmite utilizando o protocolo ZigBee, segundo os autores, este protocolo foi adotado por possuir um menor consumo de potência que o Wi-Fi.

PIC

(SAQFALHAIT; ABUSHAMMA, 2024) desenvolveram um projeto de *scanner* para diagnóstico veicular baseado em OBD-II visando fornecer uma solução independente e acessível para que os proprietários dos veículos possam ter acesso aos dados.

Com este projeto apresentado foi possível a aquisição dos dados dos automóveis, contudo, segundo os autores, por ser baseado em um microcontrolador mais antigo, seu desenvolvimento assim como integração com novas tecnologias são um desafio, além do fato de ser necessário adicionar um controlador CAN externo ao microprocessador.

Arduino UNO

Com o objetivo de criar um sistema de registro de dados (*Data logger*), para aquisição de parâmetros como RPM, velocidade do veículo, acionamento do freio e marcha, de um veículo de duas rodas, (YADAV; SAKLE, 2023), optaram por uma abordagem baixo custo. Para isso, optaram pelo microcontrolador Arduino UNO, visando economia de tempo e de recursos. Este microcontrolador recebe dados diretamente dos sensores, ou seja, não está conectado a nenhum barramento.

Inicialmente começaram o projeto em bancada e em seguida embarcaram o sistema em uma motocicleta Bajaj M-80, para testes em condições de condução reais, incluindo um circuito urbano com 2,61 km de extensão. Durante esses testes, os dados foram registrados a uma taxa de 3 Hz, o que segundo os autores fornece uma base robusta para a análise de padrões de troca de marchas e comportamento de direção (YADAV; SAKLE, 2023).

O estudo concluiu que o *data logger* desenvolvido não só é eficaz e preciso, mas também pode ser adaptado para diferentes tipos de veículos com modificações mínimas, fornecendo uma solução versátil e econômica para monitoramento e análise de dados veiculares. Ou seja, mesmo com um processador de fácil acesso e baixo custo é possível realizar um projeto neste tema com alta confiabilidade e eficácia.

Raspberry Pi

Um estudo liderado por Moniaga (2018) explorou a utilização de uma Raspberry Pi para aquisição de dados via OBD-II, visando analisar acidentes causados por mau funcionamento do carro, além disso, permitir que proprietários de veículos detectem e resolvam problema antes que se agravem, melhorando a segurança rodoviária geral (MONIAGA J. V.; SAHIDI, 2018).

Neste estudo foi utilizado um *scanner* OBD comercial, que envia os dados aquisitados para uma Raspberry, a qual processará e enviará os dados via Bluetooth para o usuário. A interface do usuário foi desenvolvida para exibir de forma clara as informações de diagnóstico, incluindo códigos de falha detectados DTCs, do inglês *Diagnostic Trouble Codes*, que indicam problemas específicos no veículo.

Os resultados do estudo demonstraram que a integração do OBD-II com o Raspberry Pi é eficaz para o diagnóstico em tempo real de veículos. A configuração permitiu a coleta e processamento de dados de diagnóstico de maneira eficiente. Além disso, o uso de conexão Bluetooth para comunicação entre a Raspberry Pi e o dispositivo de exibição de dados mostrou ser uma solução prática e de baixo consumo de energia.

2.6 Protocolo CAN com ESP32

A documentação oficial da Espressif Systems (SYSTEMS, 2023) oferece uma explicação detalhada sobre o funcionamento e implementação do protocolo CAN utilizando uma ESP32 como microcontrolador. Este material além de abordar requisitos e configurações de software também auxilia no desenvolvimento do hardware. No contexto de desenvolver um dispositivo de aquisição de dados veiculares, como o discutido no presente trabalho, utilizar um controlador CAN com o microcontrolador acima citado se mostra uma solução eficiente e de baixo custo. Apesar de possuir um controlador CAN integrado, ainda é necessário um *transceiver* externo a ESP32 para lidar com a interface física da rede CAN (SYSTEMS, 2023).

2.7 Considerações Finais

Analisando os trabalhos e soluções já estudadas, percebe-se uma lacuna na escolha de microprocessadores, uma vez que os escolhidos não possuem as novas tecnologias ou apresentam um custo muito elevado. Portanto busca-se outras opções mais modernas e com melhor custo benefício, como por exemplo ESP32 ou STM32.

Além disso, a maioria dos estudos não focou na facilidade de acesso aos dados adquiridos, deixando de lado a implementação de plataformas acessíveis, como um site ou aplicativo. Essa falta de enfoque na interface de visualização dos dados pode limitar o uso prático das informações coletadas.

3 METODOLOGIA

O objetivo principal deste projeto é desenvolver um sistema de aquisição de dados para um veículo de corrida. Para isso, será criada uma placa de aquisição eletrônica que, utilizando um microcontrolador com conectividade Wi-Fi, permitirá a leitura dos dados dos sensores e o envio dessas informações para um servidor web na nuvem. Com os dados disponíveis na plataforma de nuvem FireBase, espera-se possibilitar a visualização em um dispositivo smartphone, oferecendo praticidade no monitoramento e análise do desempenho do veículo.

O desenvolvimento deste projeto envolveu as seguintes etapas:

1. 1 - Projeto Eletrônico;
2. 2 - Criação e configuração do banco de dados;
3. 3 - Programação do microcontrolador;
4. 4 - Desenvolvimento do app/web.

Essa subdivisão também está em ordem de execução do projeto.

3.1 Projeto eletrônico

Nesta seção será apresentada toda a metodologia envolvida para o projeto eletrônico do dispositivo.

3.1.1 Microcontrolador

A primeira etapa do projeto foi tomar a decisão de qual microcontrolador utilizar, visto os trabalhos analisados na revisão bibliográfica, os autores optavam pela escolha mais prática e que atenda todos os requisitos do projeto. Sendo assim, as duas principais necessidades que o microcontrolador precisa ter é compatibilidade com protocolo CAN e com módulo Wi-Fi.

Para cumprir ambos requisitos, o ESP32 foi o escolhido para o projeto, ilustrado na Figura 4

Embora o ESP32 tenha um controlador compatível com barramento CAN integrado, ele não possui um CAN *transceiver* integrado, portanto, deve-se utilizar um externo para conectar-se a uma rede CAN. Em suma, um controlador CAN é a parte de *hardware* responsável por lidar com o protocolo CAN, já o *transceiver* é a parte física, ou seja, lidar com a comunicação diferencial.

Figura 4 – Microcontrolador ESP32

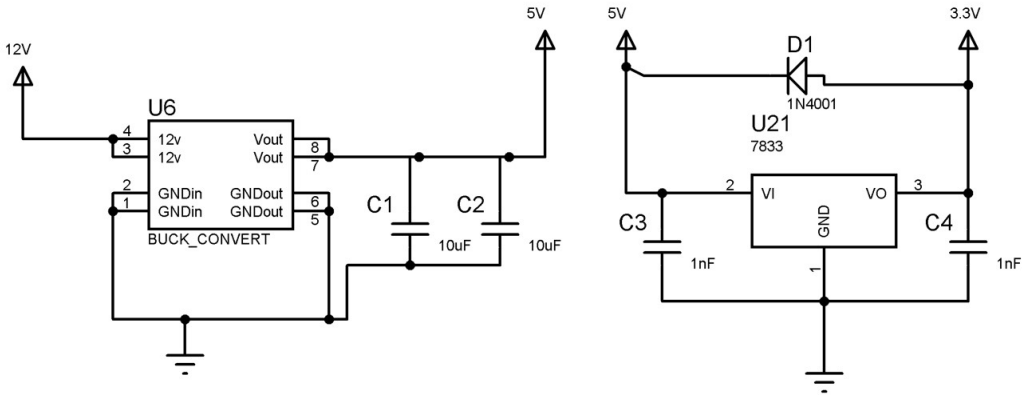
Fonte: Retirado de (SYSTEMS, 2023)

3.1.2 Alimentação

Levando em consideração que a tensão da bateria utilizada no carro do grupo extracurricular, Baja, e também dos carros convencionais é de 12 V, é necessário converter a tensão de 12 V para 3,3 V (tensão de funcionamento da rede CAN).

O esquemático pode ser visto na Figura 5.

Figura 5 – Alimentação do circuito



Fonte: próprio autor.

Fazer uma conversão direta de 12 V para 3,3 V, além de ser preciso um conversor muito bem regulado e estável, de acordo com Texas Instruments 2008, a dissipação de calor também pode ser um problema. Para isso, foi inicialmente utilizado um *Mini Buck* para converter de 12 V para 5 V, esta tensão é interessante também pois serve para alimentar outros circuitos, nessa etapa de regulação, os capacitores C1 e C2 (10 µF) estão presentes para filtrar e estabilizar a saída de 5V. Essa saída será a entrada de um regulador de tensão 7833 (componente U21), já nesta última etapa de regulação de tensão, o diodo D1 (1N4001) protege contra tensões reversas, enquanto os capacitores C3 e C4 (1 nF) filtram os ruídos na entrada e saída.

Para confirmar que o regulador de tensão está operando em condições corretas de temperatura, foi utilizada a seguinte equação:

$$T_J = T_A + \rho_{JA} \times V \times I = 30 + 65 \times (5 - 3,3) \times 100mA = 41,05℃ \quad (3.1)$$

Sendo:

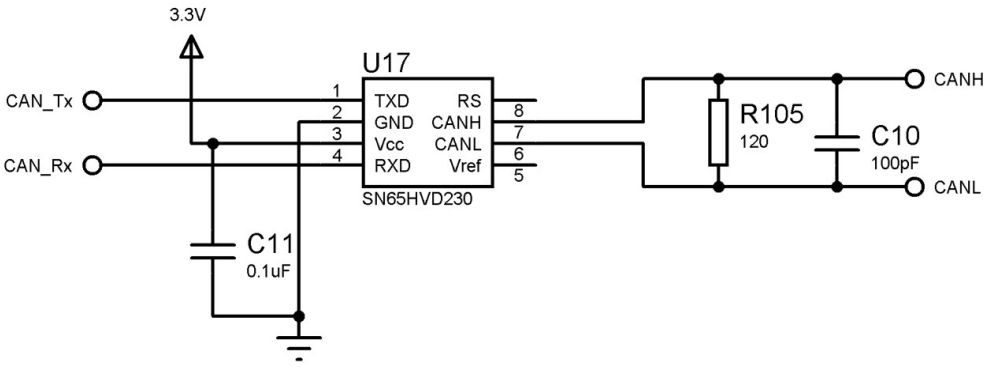
- T_A: temperatura ambiente;
- T_J: temperatura de junção;
- ρ_{JA}: resistividade térmica de junção com ambiente;
- V: tensão dissipada;
- I: Corrente consumida.

Como a temperatura de junção resultante, 41,05 ℃, foi menor que a máxima especificada pela fabricante Unisonic Technologies 2005 (150℃), conclui-se que o circuito é aplicável.

3.1.3 Comunicação CAN

Para a correta comunicação CAN, foi utilizado o *transceiver* SN65HVD233, e o circuito ilustrado na Figura 6.

Figura 6 – Comunicação CAN



Fonte: próprio autor.

Neste circuito o componente principal é o *transceiver* SN65HVD230, responsável por converter o sinal serial (`CAN_TX` e `CAN_RX`) do ESP32 em sinal diferencial (`CAN_H` e `CAN_L`), ou seja, faz a interface entre o microcontrolador e a rede CAN.

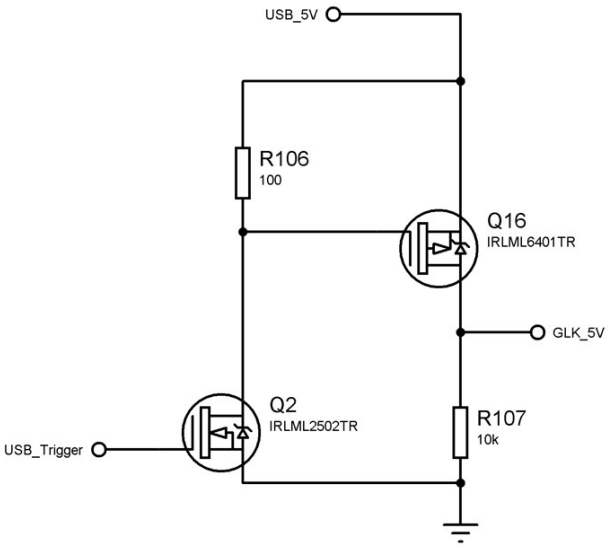
O capacitor C11, de 0,1 μF tem como objetivo desacoplar a alimentação do *transceiver*, ou seja, filtrar ruídos de alta frequência. Já o capacitor C10, de 100 pF, também é um filtro de alta frequência com o objetivo de minimizar ruídos na linha de transmissão.

Por fim, o resistor R105 de 120 Ω é o resistor de terminação da rede CAN, utilizado para evitar reflexões do sinal e garantir a integridade dos dados.

3.1.4 Cartão SD e USB

Com o intuito de salvar todos os dados em um *datalog* embarcado, também foi desenvolvido um circuito para salvamento e leitura de um cartão SD, assim como acessá-lo por meio de um cabo USB. Optou-se por desenvolver o circuito ao invés de comprar um módulo visando uma melhor integração na PCB. O circuito desenvolvido, está mostrado nas Figuras 7 e 8.

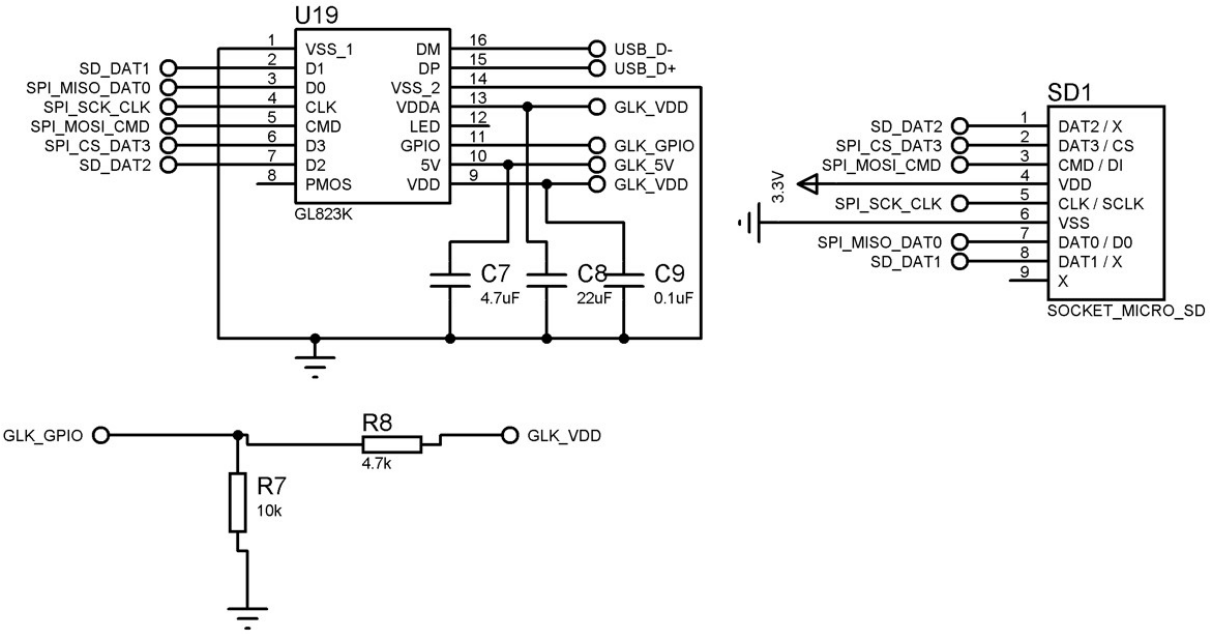
Figura 7 – Circuito para alimentação do cartão SD



Fonte: próprio autor.

O circuito da Figura 7 faz parte da comunicação USB, controlando a alimentação de 5V (`GLK_5V`) com base na entrada `USB_Trigger`. Neste circuito o MOSFET Q2 tem seu *gate* controlado pelo `USB_Trigger`, quando este sinal é ativado, o transistor Q16 é acionado, e por fim, este controla a alimentação 5V.

Figura 8 – Circuito para salvamento do cartão SD



Fonte: próprio autor.

O resistor F106, de 100 Ω limita a corrente de alimentação e o resistor R107, de 10 kΩ opera como um *pull-down*, conectado ao *gate* de Q16, garantindo assim que este permaneça desligado quando `USB_Trigger` está inativo.

Já o segundo circuito da Figura 8 é implementado uma interface entre um *host* USB e o cartão SD, utilizando o controlador GL823K. Em suma, este circuito tem o seguinte funcionamento, os pinos `USB_D-` e `USB_D+` no controlador GL823K se conectam ao *host* USB, enquanto os pinos `SPI_SCK`, `SPI_MOSI`, `SPI_MISO`, e `SPI_CS` permitem a comunicação serial com o cartão microSD.

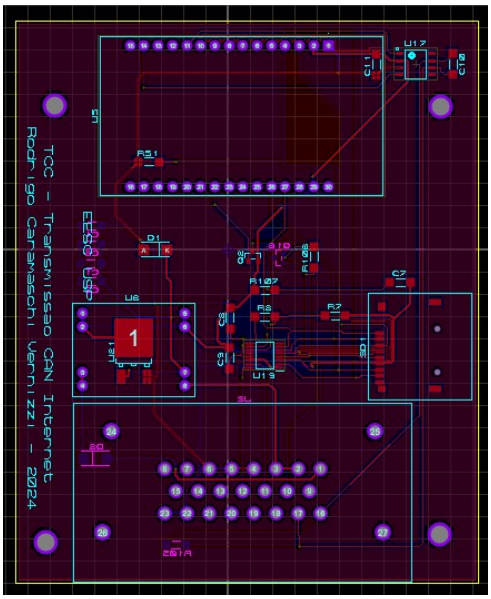
Os resistores R7 (10 kΩ) e R8 (4.7 kΩ), além de operarem como *pull-down* e *pull-up*, são também um divisor de tensão, garantindo que a tensão na linha `GLK_GPIO` seja uma fração da alimentação total `GLK_VDD`, estabilizando o sinal e prevenindo flutuações que poderiam causar mau funcionamento no circuito. E por fim, os capacitores C7 (4,7 µF), C8 (22 µF), e C9 (0,1 µF) são usados para filtrar e estabilizar a alimentação do circuito, prevenindo interferências e ruídos.

3.1.5 *Design* PCB

Também foi feita uma PCB, do inglês (Printed Circuit Board), para agrupar todos os circuitos em uma dimensão (101 x 83 mm) otimizada para ser embarcada em um carro do Baja, ou seja de forma compacta, porém ao mesmo tempo do tamanho que caiba na caixa já existente. Esta PCB pode ser vista na Figura 9.

Esta PCB foi feita com dimensionamento seguindo a norma IPC2221, (IPC, 1998),

Figura 9 – PCB projetada



Fonte: próprio autor.

e também tomando os devidos cuidados para minimizar a possibilidade de interferência eletromagnética nas trilhas. Foi pensando também em uma fixação ao chassi do carro com uma *case* impressa em 3D e fixada com coxins de borracha com o intuito de minimizar a vibração.

3.2 Banco de dados

O caminho que os dados seguirão até o usuário segue o esquemático da Figura 10.

Figura 10 – Esquemático de comunicação

Fonte: próprio autor.

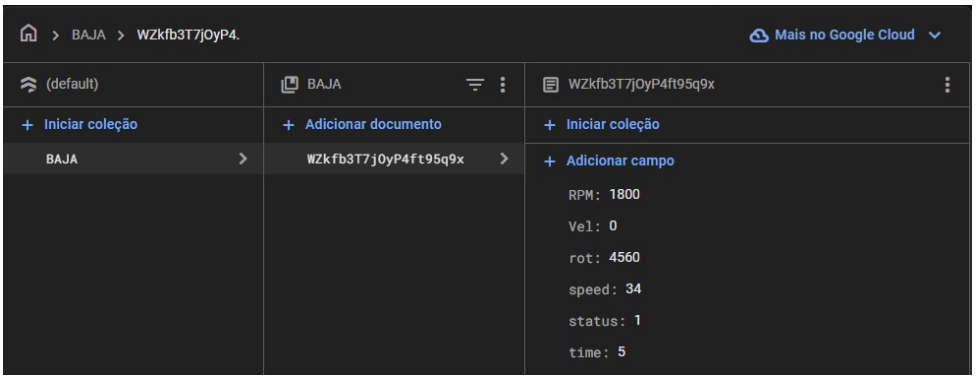
Na primeira parte da metodologia já foi abordado o módulo embarcado, e neste será apresentado o banco de dados. A plataforma *cloud* escolhida foi a FireBase da Google, visto sua facilidade de uso e por possuir uma versão gratuita com grande armazenamento.

Nesta plataforma foi escolhido o serviço FireStore, o qual funciona da seguinte maneira, simplificadamente, primeiro, define-se uma coleção e dentro dela terá um documento, e neste documento as variáveis que serão salvas. Foi escolhido este serviço, uma vez que é o mais atual da Google, e este permite cadastrar vários módulos em uma só plataforma, ou

seja, caso queira acompanhar os dados de mais de um carro, esta seria a maneira mais eficaz.

Na Figura 11 pode ser visto as divisões citadas acima, e os dados que serão salvos, mais a direita.

Figura 11 – Estrutura do Banco de Dados



Fonte: próprio autor.

Cada variável será utilizada para o seguinte proposito:

- `RPM`: aquisição em tempo real do valor do RPM do motor;
- `Vel`: aquisição em tempo real da velocidade [km/h];
- `rot`: registro do maior valor de RPM;
- `speed`: registro do maior valor de velocidade[km/h];
- `status`: status do Veículo (1 - ligado, 0 - desligado);
- `time`: temporizador[min].

3.3 Programação do microcontrolador

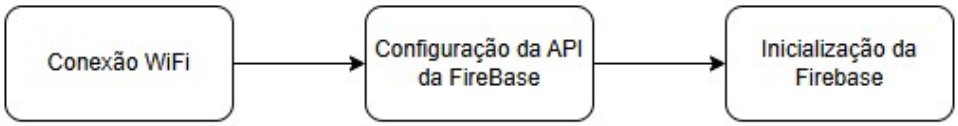
Para a programação do ESP32, foi utilizada a plataforma ESP-IDF no VSCode. O ESP-IDF oferece um ambiente de desenvolvimento robusto e completo que permite o acesso às bibliotecas e ferramentas para o desenvolvimento de sistemas IoT. Já o VSCode como IDE, oferece praticidade e eficiência, com uma interface leve, além de ferramentas de depuração avançadas. Assim utilizar ambas ferramentas, favorecem o projeto.

Para estabelecer a conexão com o Wi-Fi e a comunicação com o banco de dados foi utilizado o código presente no APÊNDICE A: DEFINIÇÕES DE PARÂMETROS. Com este código o dispositivo tenta conectar-se à uma rede Wi-Fi usando a biblioteca WiFi Manager, caso não consiga imprime "Falha na Conexão"e caso conecte, "Conectado.".

Depois, configura as credenciais da Firebase (API key, e-mail e senha do usuário) e inicia a conexão com ela.

O diagrama em alto nível deste código pode ser observado na Figura 12.

Figura 12 – Diagrama de alto nível para conexão WiFi

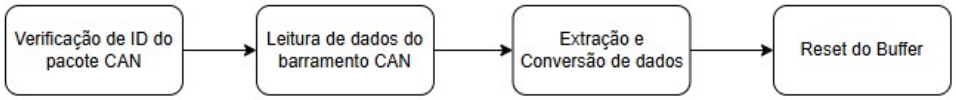


Fonte: próprio autor.

Para aquisitar os dados da CAN foi utilizada uma biblioteca já existente, a qual facilitou na programação e pode ser vista no APÊNDICE B: CAN. Neste exemplo, os dados de ID `0x010` são analisados e dentro do laço `while` são lidos até 4 bytes de dados da mensagem CAN e os armazena em um buffer. Após a leitura, o valor de `can.vel` (velocidade) é calculado combinando os dois primeiros bytes do *buffer* usando operações bit a bit, já o dado `can.rot` é combinado aos próximos 2 bytes. Por fim, o índice i é zerado ao final para que o *buffer* possa ser reutilizado na próxima leitura.

O diagrama em alto nível deste código pode ser observado na Figura 13.

Figura 13 – Diagrama de alto nível para aquisição de dados do barramento CAN



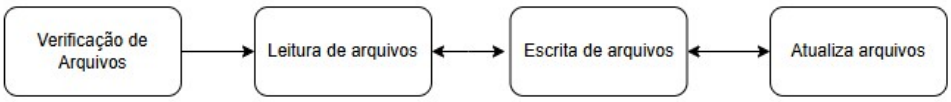
Fonte: próprio autor.

Para realizar as funções desejadas com o cartão SD, foi utilizado o código que pode ser visto no APÊNDICE C: FUNÇÕES DO CARTÃO SD, este código lida com a verificação, leitura e escrita de arquivos, seguindo a lógica:

- `Checkfiles`: Verifica se existem arquivos no formato `/fileNNN.txt` sendo "NNN"um número, caso já exista é acrescentado 1 ao valor do número criando um novo arquivo.
- `readFile`: Esta função lê e imprime o conteúdo de um arquivo, especificando seu *path*.
- `writeFile`: Abre um arquivo para salvar os dados nele.
- `appendFile`: Abre um arquivo para salvar dados no final.

O diagrama em alto nível deste código pode ser observado na Figura 14.

Figura 14 – Diagrama de alto nível para salvamento no cartão SD



Fonte: próprio autor.

E por fim para transmitir estes dados para o banco de dados, foi utilizado o código presente no APÊNDICE D: COMUNICAÇÃO COM O BANCO DE DADOS, realizando as devidas alterações, ou seja, alterando o nome de qual variável será enviada e o endereço para qual será enviada. O comando `Firebase.Firestore.createDocument`, faz uma requisição para criar um novo documento na FireStore, sendo:

- `fbdo`: Objeto gerenciador de respostas e erros.
- `FIREBASE_PROJECT_ID`: O ID do projeto no Firebase.
- `documentPath.c_str()`: O caminho do documento que será criado.
- `content.raw()`: O conteúdo (dados) que será salvo no documento.

3.4 Desenvolvimento do aplicativo

A última etapa do projeto foi o desenvolvimento do aplicativo, para isso foi utilizado o *framework* Flutter, o qual além de permitir uma fácil integração com a Firebase, permite desenvolver aplicativo web, iOs e também Android com o mesmo código.

Além disso, foi feita também uma pesquisa com o time EESC USP Baja para que o layout do aplicativo possua a identidade visual da equipe. Então foi utilizado o grito de guerra da equipe, assim como suas cores principais: preto, laranja e azul. Como pode ser visto na Figura 15, a página inicial do aplicativo.

Figura 15 – Página inicial do aplicativo

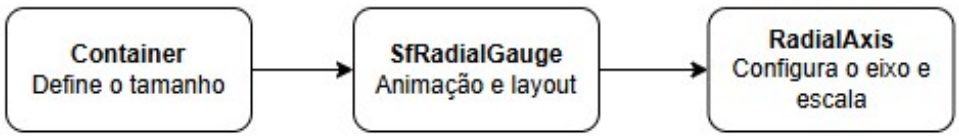
Fonte: próprio autor.

Nessa página é possível acompanhar em tempo real a velocidade e o RPM nos velocímetro e tacômetro, respectivamente.

Para fazer o gráfico dos velocímetro e tacômetro com movimentação, foi utilizado o código presente no APÊNDICE E: CÓDIGO PARA O VELOCÍMETRO E O TACÔMETRO, neste é definido um *container*, que será a área na qual estes itens serão disponibilizados, dentro dessa estrutura é definido o primeiro `child`, ou seja, uma estrutura, que será o velocímetro, o tacômetro não está no apêndice porém foi feito de forma análoga. Neste `child` é definido o ponteiro e também os valores intervalados, e para correlacionar o ângulo que o ponteiro está e o valor que apontará, foi utilizada uma biblioteca já pronta, *syncfusion*.

O diagrama em alto nível deste código pode ser observado na Figura 16.

Figura 16 – Diagrama de alto nível para *layout* do aplicativo

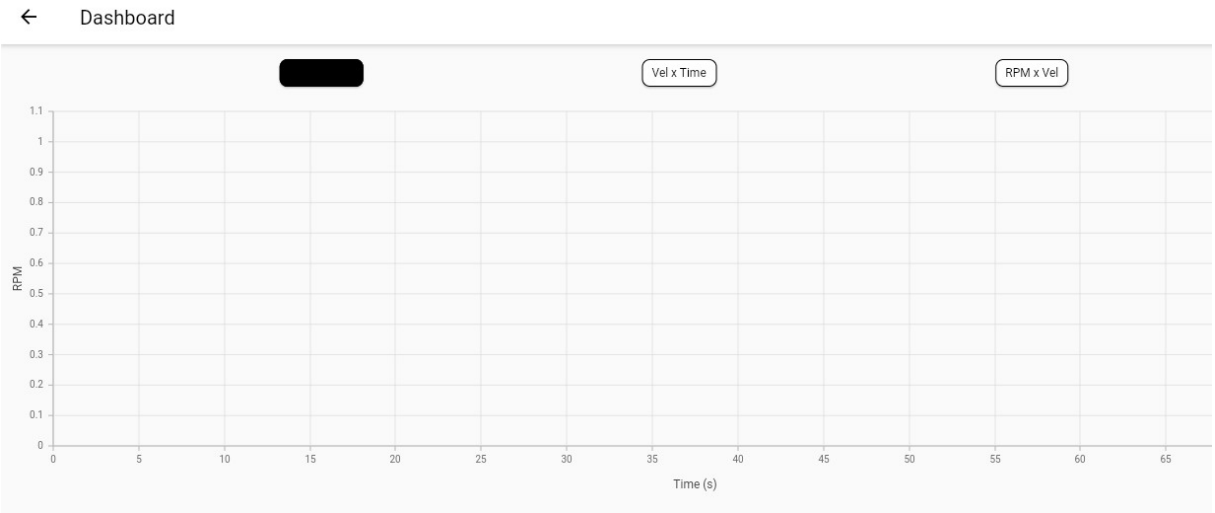


Fonte: próprio autor.

Na tabela abaixo do velocímetro e tacômetro, é registrado o maior valor de cada variável assim como quanto tempo de corrida já passou, em minutos. No canto inferior esquerdo

é possível ver um botão que direciona para a página de gráficos a qual pode ser vista na Figura 17.

Figura 17 – Página dedicada a gráfico



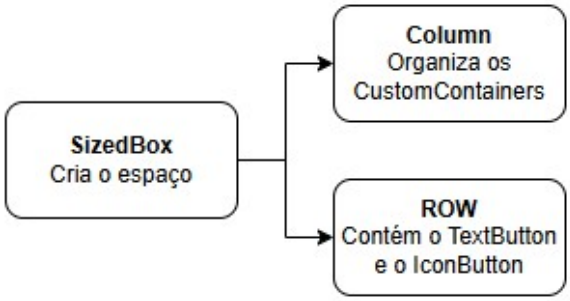
Fonte: próprio autor.

O primeiro quadrado está na cor preta, uma vez que este foi selecionado para mostrar na tela, e este seria o gráfico de RPM por tempo.

Para fazer essas estruturas foi utilizado o código presente no APÊNDICE F: TABELA E BOTÃO DO APLICATIVO, neste inicialmente é definido um `children`, uma estrutura dentro do `child`, que contém três `custom container`, que são as 3 linhas da tabela. Já para o botão é utilizado o `children, textButton` e que com a função `onPressed` é definido a função que será executada ao pressionar o botão.

Segue diagrama em alto nível deste código pode ser observado na Figura 18.

Figura 18 – Diagrama de alto nível para *layout* do aplicativo



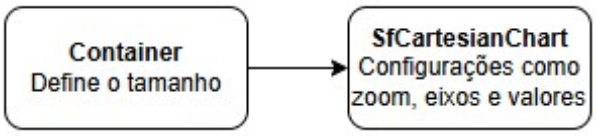
Fonte: próprio autor.

Nesta página, três gráficos podem ser selecionados, rotação por tempo, velocidade por tempo ou rotação por velocidade, todos são de interesse para testes como por exemplo o de

aceleração e velocidade. Estes gráficos são atualizados em tempo real e também possui cursor tanto para zoom quanto para analisar o valor de forma mais precisa. O código para gerar estes gráficos está no APÊNDICE G: GRÁFICOS, neste é criado um *widget*, o qual contém um *container* com o gráfico desejado. Nesta etapa foi utilizada a biblioteca, SfCartesianChart, na qual é possível selecionar opções como zoom (`enableSelectionZooming: true`), linhas de marcação (`crosshairBehavior: CrosshairBehavior(enable: true`) e movimentar o gráfico (`enablePanning: true`) de forma mais fácil. E para definir os eixos foi utilizada as funções `PrimaryXAxis` para o eixo X e `PrimaryYAxis` para o eixo Y.

O diagrama em alto nível dos gráficos pode ser observado na Figura 19.

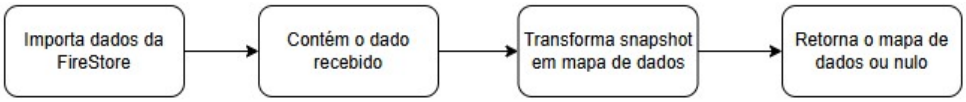
Figura 19 – Diagrama de alto nível para os gráficos



Fonte: próprio autor.

Por fim, o código no APÊNDICE H: AQUISIÇÃO DE DADOS DO APP foi utilizado para aquisitar os dados presentes no banco de dados, na função `FirebaseFirestore.instance` é definido o caminho para os dados, e a função `snapshot` é executada para captar os dados e salvá-los nas variáveis do aplicativo. O qual esta explicado no diagrama de alto nivel da Figura 20.

Figura 20 – Diagrama de alto nível para aquisição dos dados da núvem



Fonte: próprio autor.

4 RESULTADOS E DISCUSSÃO

Após a aplicação da metodologia proposta, todo o projeto foi colocado em teste, e os resultados evidenciaram que o dispositivo foi capaz de adquirir os dados da rede CAN e transmiti-los pela internet. Sendo assim, pode-se concluir que todas as etapas foram bem sucedidas, a parte eletrônica cumpriu todos os requisitos, a configuração do banco de dados que garantiu que os dados fossem armazenados nos locais corretos, assim como o código do ESP32, que possibilitou uma aquisição e transmissão confiável e funcional. E o aplicativo que cumpriu o propósito de mostrar ao usuário os dados do veículo em tempo real.

4.1 Desempenho da rede CAN

A rede CAN foi configurada para operar com uma taxa de 250 kbps, um valor capaz de atender as demandas do projeto. Um pacote era enviado a cada 775 μs, contendo todas os dados dos sensores de RPM e de velocidade. Tanto a estabilidade quanto a integridade dos dados da rede CAN foram comprovadas pelos testes realizados, ou seja, durante os testes não foi visto nenhuma perda de pacote na comunicação diferencial, mostrando assim ser robusta e eficaz.

4.2 Frequência de aquisição e salvamento

O sistema de salvamento de dados no cartão SD é de 50 Hz, ou seja, existe uma alta taxa de aquisição e salvamento local, o que permite uma análise mais completa e fiel após a realização do teste ou da corrida. Ou seja, este sistema local garante a aquisição de dados completa e integra mesmo que ocorra erros de comunicação via internet. Esta configuração balanceou de forma eficiente a necessidade de alta resolução temporal com a capacidade de armazenamento e processamento do sistema.

4.3 Transmissão de dados via Internet

Os dados coletados pela ESP32 são enviados para a Firebase utilizando uma conexão via internet, com uma taxa média de transmissão de 30 kbps, embora esse valor seja relativamente baixo em comparação com sistemas de maior complexidade, se mostrou adequado no contexto deste trabalho.

Durante os testes também foi registrado que o atraso, `delay` médio entre a aquisição dos dados e sua exibição no aplicativo foi de 180 ms. Além disso, foi medido que uma mensagem é enviada a cada 100 ms sem que ocorra nenhuma perda de pacotes entre o ESP32 e o aplicativo.

No entanto em situações extremas (utilização de dados móveis) foi notado um *delay* de até 1 segundo. Este atraso, porém, não é prejudicial ao projeto uma vez que existe em paralelo os dados do cartão SD.

4.4 Aplicativo

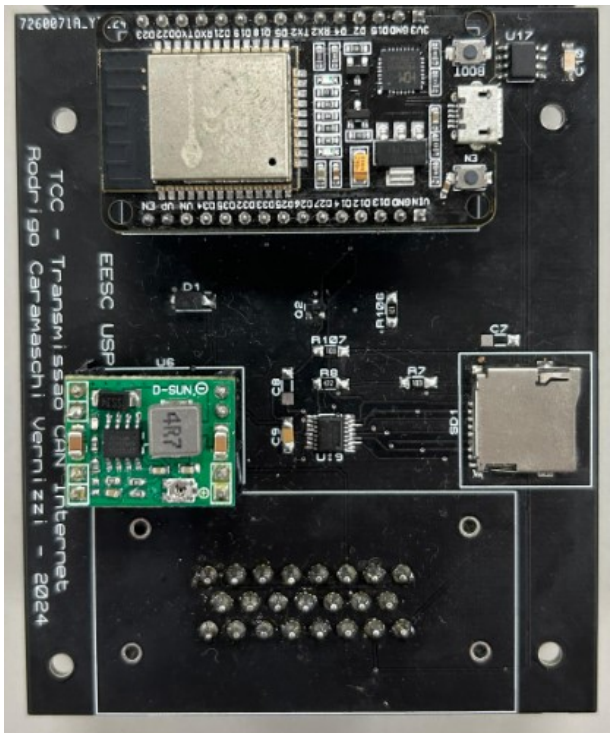
O aplicativo teve o funcionamento estável durante todos os testes e pelo fato de ser multiplataforma, é de ampla utilização pelos usuários. Ademais o aplicativo por possuir um layout mais simples e de fácil usabilidade, permitiu uma integração melhor e mais eficiente com a equipe.

Os gráficos gerados em tempo real cumpriram o propósito de informar ao usuário as condições em tempo real do veículo, contudo as análises destes dados, pelo fato do *delay*, não foram tão precisas quanto o esperado, especialmente em situações onde o tempo de resposta é crítico, como em testes de aceleração ou frenagem. Porém esse problema foi facilmente contornado pela utilização dos dados salvos na memória local.

4.5 Placa de circuito impresso

A PCB não apresentou falhas com a vibração do carro nem sinais de interferências eletromagnéticas, sendo assim, obteve um ótimo resultado. A versão final da PCB pode ser vista na Figura 21.

Figura 21 – PCB finalizada



Fonte: próprio autor.

Figura 22 – PCB finalizada, parte de trás

Fonte: próprio autor.

5 CONCLUSÃO

Este trabalho teve como objetivo desenvolver um dispositivo para aquisição de dados da rede CAN de um veículo e transmiti-los pela internet, abordando todas as etapas necessárias para o funcionamento eficiente do sistema. O *hardware* e *software* desenvolvido resultou em um dispositivo de baixo custo, capaz de atingir os objetivos com êxito.

Os testes demonstraram que os requisitos necessários, para o monitoramento veicular em aplicações cotidianas e também em competições, foram atingidos. Mesmo com o atraso observado, a robustez do dispositivo consegue garantir sua confiabilidade.

Em suma, o trabalho foi concluído de forma satisfatória, atingindo os objetivos propostos, apresentando ainda, grande potencial para futuras aplicações e melhorias.

REFERÊNCIAS

BOLAND MORGAN I. BURGETT, A. J. E. R. M. S. I. H. M. An overview of can-bus development, utilization, and future potential in serial network messaging for off-road mobile equipment. In: AHMAD, F.; SULTAN, M. (Ed.). *Technology in Agriculture*. Rijeka: IntechOpen, 2021. cap. 25. Disponível em: https://doi.org/10.5772/intechopen.98444.

CHANDIRAMANI, J. R.; BHANDARI, S.; HARIPRASAD, S. Vehicle data acquisition and telemetry. In: *2014 Fifth International Conference on Signal and Image Processing*. [S.l.: s.n.], 2014. p. 187–191.

ELECTRONICS, C. *CAN Bus Sniffer - Reverse Engineer Vehicle Data [Savvy-CAN/Wireshark]*. 2024. Disponível em: https://www.csselectronics.com/pages/can-bus-sniffer-reverse-engineering.

EMBARCADOS. *Barramento CAN entre Arduinos UNO*. 2020. Accessed: 2024-08-27. Disponível em: https://embarcados.com.br/barramento-can-entre-arduinos-uno/.

IPC. *IPC-2221: Generic Standard on Printed Board Design*. [S.l.], 1998. Acesso em: 3 set. 2024. Disponível em: https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf.

LI, R.; LIU, C.; LUO, F. A design for automotive can bus monitoring system. In: *2008 IEEE Vehicle Power and Propulsion Conference*. [S.l.: s.n.], 2008. p. 1–5.

MONIAGA J. V., M. S. R. H. D. A.; SAHIDI, F. Diagnostics vehicle's condition using obd-ii and raspberry pi technology: study literature. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2018. v. 978, p. 012011.

NURUDIN, A. A. S.; ZARLIS, M. Monitoring applications for vehicle based on internet of things (iot) using the mqtt protocol. *Procedia Computer Science*, Elsevier, v. 227, p. 73–82, 2023.

RIMPAS, D.; PAPADAKIS, A.; SAMARAKOU, M. Obd-ii sensor diagnostics for monitoring vehicle operation and consumption. *Energy Reports*, Elsevier, v. 6, p. 55–63, 2020.

Robert Bosch GmbH. *CAN Specification*. [S.l.], 2012. Accessed: 2024-06-19. Disponível em: http://esd.cs.ucr.edu/webres/can20.pdf.

SALUNKHE, A. A.; KAMBLE, P. P.; JADHAV, R. Design and implementation of can bus protocol for monitoring vehicle parameters. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. [S.l.: s.n.], 2016. p. 301–304.

SAQFALHAIT, A.; ABUSHAMMA, M. *On-Board Diagnostics Project*. 2024. Hardware graduation project, Supervisor: Dr. Samer Arandi.

SYSTEMS, E. *ESP32 TWAI (CAN) Controller — ESP-IDF Programming Guide*. [S.l.], 2023. Accessed: 2024-09-16. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/twai.html.

TAYLOR NATHAN GRIFFITHS, A. B. S. A. T. P. Z. X. P.; GELENCSER, A. Data mining for vehicle telemetry. *Applied Artificial Intelligence*, Taylor & Francis, v. 30, n. 3, p. 233–256, 2016. Disponível em: https://doi.org/10.1080/08839514.2016.1156954.

Texas Instruments. *SN65HVD230 3.3-V CAN Transceiver*. [S.l.], 2008. Acesso em: 3 set. 2024. Disponível em: https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf?ts=1725330017442&ref_url=https%253A%252F%252Fwww.google.com%252F.

TIETOEVRY. *Connected Vehicle and Telematics Development Services*. [S.l.], 2023. Accessed: 2024-06-19. Disponível em: https://www.tietoevry.com/en/industries/automotive/connected-vehicle-and-telematics/.

Unisonic Technologies Co., Ltd. *LM78XX: 3-Terminal 1A Positive Voltage Regulator*. [S.l.], 2005. Acesso em: 3 set. 2024. Disponível em: https://www.unisonic.com.tw.

YADAV, A.; SAKLE, N. Development of low-cost data logger system for capturing transmission parameters of two-wheeler using arduino. *Materials Today: Proceedings*, Elsevier, v. 72, p. 1697–1703, 2023.

6 APÊNDICE A: DEFINIÇÕES DE PARÂMETROS

```
1     // definindo a biblioteca Wifi
2   WiFiManager wm;
3   bool res;
4   res = wm.autoConnect("BAJA-IoT");
5   if(!res) {
6         Serial.println("Falha na conex o.");
7     }
8     else {
9         Serial.println("Conectado.");
10    }
11    //definindo as chaves e os par metros da Firebase
12  Serial.printf("Firebase CLient v%s\n\n",
       FIREBASE_CLIENT_VERSION);
13  config.api_key = API_KEY;
14
15  auth.user.email = USER_EMAIL;
16  auth.user.password = USER_PASSWORD;
17
18
19  config.token_status_callback = tokenStatusCallback;
20
21  Firebase.begin(&config, &auth);
22  Firebase.reconnectWiFi(true);
23  }
```

7 APÊNDICE B: CAN

```
1      if(CAN.packetId() == 0X010){
2      while (CAN.available() && i<4) {
3        can.buffer[i] = uint8_t(CAN.read());
4          i += 1;
5        }
6
7        can.vel = ((uint16_t)can.buffer[1] << 8) | can.buffer[0];
8        // filter = ((uint16_t)can.buffer[1] << 8) | can.buffer
             [0];
9        // if (filter<9999)
10       //   can.vel = filter;
11
12       can.rot = ((uint16_t)can.buffer[3] << 8) | can.buffer[2];
13       // filter = ((uint16_t)can.buffer[3] << 8) | can.buffer
             [2];
14       // if (filter<9999)
15       //   can.rot = filter;
16
17       // memcpy(&can.temp_celsius, can.buffer+4, 4);
18     i = 0;
19   }
```

8 APÊNDICE C: FUNÇÕES DO CARTÃO SD

```
1      void Checkfiles(fs::FS &fs, char *path) {
2    int i = 0;
3       while(fs.exists(path)){
4       i+=1;
5      sprintf(path,"/file%03d.txt",i);
6      Serial.println(path);
7      }
8  }
9
10 // Fun  o para leitura SD
11 void readFile(fs::FS &fs, const char * path){
12     Serial.printf("Reading␣file:␣%s\n", path);
13
14     File file = fs.open(path);
15     if(!file){
16         Serial.println("Failed␣to␣open␣file␣for␣reading");
17         return;
18     }
19
20     Serial.print("Read␣from␣file:␣");
21     while(file.available()){
22         Serial.write(file.read());
23     }
24 }
25
26 // Fun  o Escrita do SD
27 void writeFile(fs::FS &fs, const char * path, const char *
      message){
28     Serial.printf("Writing␣file:␣%s\n", path);
29
30     File file = fs.open(path, FILE_WRITE);
31     if(!file){
32         Serial.println("Failed␣to␣open␣file␣for␣writing");
33         return;
34     }
35     if(file.print(message)){
36         Serial.println("File␣written");
```

```
37      } else {
38          Serial.println("Write failed");
39      }
40  }
41
42  // Criar ou adiocionar dados (essa vai ser utilizada para
      salvar no sd)
43  void appendFile(fs::FS &fs, const char * path, const char *
      message){
44      File file = fs.open(path, "a");
45      file.println(message);
46      file.close();
47  }
```

9 APÊNDICE D: COMUNICAÇÃO COM O BANCO DE DADOS

```
1     if(Firebase.Firestore.createDocument(&fbdo,
          FIREBASE_PROJECT_ID, "", documentPath.c_str(), content.
          raw() )){
2     Serial.printf("ok\n%s\n\n", fbdo.payload().c_str());
3     return;
4   }else{
5     Serial.println(fbdo.errorReason());
6   }
```

10 APÊNDICE E: CÓDIGO PARA O VELOCÍMETRO E O TACÔMETRO

```
Container(
                    height: 150,
                    width: 150,
                    child: SfRadialGauge(
                      enableLoadingAnimation: true,
                      animationDuration: 4500,
                      axes: <RadialAxis>[
                        RadialAxis(
                            minimum: 0,
                            maximum: 60.5,
                            axisLineStyle: AxisLineStyle(
                                thickness: 20, color: Colors.
                                   grey),
                            showTicks: true,
                            axisLabelStyle: GaugeTextStyle(
                               color: Colors.white),
                            minorTickStyle: MinorTickStyle(
                               color: Colors.white),
                            majorTickStyle: MajorTickStyle(
                               color: Colors.white),
                            pointers: <GaugePointer>[
                              NeedlePointer(
                                  value: vel,
                                  enableAnimation: true,
                                  needleStartWidth: 0,
                                  needleEndWidth: 5,
                                  needleColor: Color(0xFFDADADA
                                     ),
                                  knobStyle: KnobStyle(
                                      color: Color.fromARGB
                                         (255, 235, 3, 3),
                                      borderColor: Color(0
                                         xFFDADADA),
                                      knobRadius: 0.06,
                                      borderWidth: 0.04),
                                  tailStyle: TailStyle(
                                      color: Color.fromARGB
```

```
                                    (255, 165, 4, 4),
                                width: 5,
                                length: 0.15)),
                          RangePointer(
                              value: vel,
                              width: 20,
                              enableAnimation: true,
                              color: Colors.orange)
                        ],
                        annotations: <GaugeAnnotation>[
                          GaugeAnnotation(
                            widget: Container(
                                child:
                                    Text(vel.
                                       toStringAsFixed(0) +
                                        '␣km/h',
                                        style: TextStyle(
                                          fontSize: 15,
                                          fontWeight:
                                             FontWeight.
                                             bold,
                                          color: Colors.
                                             white,
                                        ))),
                              angle: 90,
                              positionFactor: 0.87,
                          )
                        ]),
                    ],
                  ),
                ),
```

11 APÊNDICE F: TABELA E BOTÃO DO APLICATIVO

```
1     SizedBox(height: 25),
2               // Custom containers
3               Column(
4                 children: [
5                   CustomContainer(
6                     'Velocidade M x.',
7                     snapshot.data!['speed'].toStringAsFixed(0),
8                     Icons.speed_sharp,
9                   ),
10                  CustomContainer(
11                    'RPM M x.',
12                    snapshot.data!['rot'].toStringAsFixed(0),
13                    Icons.speed_outlined,
14                  ),
15                  CustomContainer(
16                    'Tempo.',
17                    snapshot.data!['time'].toStringAsFixed(0),
18                    Icons.speed_outlined,
19                  ),
20                ],
21              ),
22              SizedBox(height: 5),
23              // "Gr ficos" button
24              Row(
25                children: [
26                  TextButton(
27                    onPressed: () {
28                      Navigator.push(
29                        context,
30                        MaterialPageRoute(
31                            builder: (context) =>
                                GraficosScreen()),
32                      );
33                    },
34                    child: Text(
35                      'Gr ficos',
36                      style: TextStyle(color: Colors.white),
```

```
                    ),
                  ),
                  IconButton(
                    onPressed: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(
                            builder: (context) =>
                                GraficosScreen()),
                      );
                    },
                    icon: Icon(Icons.arrow_forward, color:
                       Colors.white),
                  ),
                ],
              ),
            ],
          ),
        );
      },
    );
  }
}
```

12 APÊNDICE G: GRÁFICOS

```
1   Widget _buildRpmTimeChart(List<int> timeValues, List<double
       > rpmValues) {
2   return Container(
3     height: 400,
4     child: SfCartesianChart(
5       zoomPanBehavior: ZoomPanBehavior(
6           // Enables pinch zooming
7           enablePinching: true,
8           enablePanning: true,
9           enableSelectionZooming: true,
10          selectionRectBorderColor: Colors.red,
11          selectionRectBorderWidth: 1,
12          selectionRectColor: Colors.grey),
13      primaryXAxis: NumericAxis(
14        interactiveTooltip: InteractiveTooltip(
15            // Enables the crosshair tooltip
16            enable: true),
17        title:
18            AxisTitle(text: 'Time (s)', textStyle: TextStyle(
                 fontSize: 12)),
19        labelFormat: '{value}',
20        labelStyle: TextStyle(fontSize: 10), // Adjust label
             font size
21      ),
22      primaryYAxis: NumericAxis(
23        title: AxisTitle(text: 'RPM', textStyle: TextStyle(
             fontSize: 12)),
24        labelStyle: TextStyle(fontSize: 10), // Adjust label
             font size
25      ),
26      series: <ChartSeries>[
27        LineSeries<dynamic, dynamic>(
28          dataSource: _getDataPoints(timeValues, rpmValues),
29          xValueMapper: (dynamic data, _) => data['x'],
30          yValueMapper: (dynamic data, _) => data['y'],
31        ),
32      ],
```

```
33          crosshairBehavior: CrosshairBehavior(
34            enable: true,
35            activationMode: ActivationMode.singleTap,
36            lineType: CrosshairLineType.both,
37            shouldAlwaysShow: true,
38            lineColor: Colors.blue.withOpacity(0.5),
39            lineWidth: 1,
40            lineDashArray: [5, 5],
41          ),
42        ),
43      );
44    }
```

13 APÊNDICE H: AQUISIÇÃO DE DADOS DO APP

```
1     import 'package:cloud_firestore/cloud_firestore.dart';
2
3  class DataRetrievalService {
4    static Stream<Map<String, dynamic>?> retrieveData() {
5      return FirebaseFirestore.instance
6          .collection('BAJA')
7          .doc('')
8          .snapshots()
9          .map((snapshot) {
10       if (snapshot.exists) {
11         Map<String, dynamic>? data =
12             snapshot.data() as Map<String, dynamic>?; //
                  Explicit cast
13         if (data != null) {
14           return {
15             'RPM': data['RPM'],
16             'Vel': data['Vel'],
17             'Status': data['status'],
18             'speed': data['speed'],
19             'rot': data['rot'],
20             'time': data['time'],
21           };
22         }
23       }
24       return null; // Return null if the document does not
            exist or if the fields are not found
25     });
26   }
27 }
```