UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

VÍTOR DA COSTA MATIAS

Ferramenta de Monitoramento de Barramento CAN para o Protótipo Automobilístico da EESC-USP Tupã

São Carlos
2024

VÍTOR DA COSTA MATIAS

Ferramenta de Monitoramento de Barramento CAN para o Protótipo Automobilístico da EESC-USP Tupã

Monografia apresentada ao Curso de Engenharia Elétrica - 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. Pedro Oliveira

VERSÃO CORRIGIDA

São Carlos

2024

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

M
433f

Matias, Vítor da Costa
Ferramenta de monitoramento de barramento CAN para o protótipo automobilístico da EESC-USP Tupã / Vítor da Costa Matias; orientador Pedro Oliveira. 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. 2. LoRa. 3. Logger. 4. Dashboard. 5. Automobilismo. 6. Tupã. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Vitor da Costa Matias

Título: “Ferramenta de Monitoramento de Barramento CAN para o Protótipo Automobilístico da EESC-USP Tupã”

Trabalho de Conclusão de Curso defendido e aprovado em 29/12/2024,

com NOTA (7,0), pela Comissão Julgadora:

Prof. Dr. Pedro de Oliveira Conceição Júnior - Orientador - SEL/EESC/USP

Prof. Dr. Maximiliam Luppe - SEL/EESC/USP

Prof. Associado José Roberto Boffino de Almeida Monteiro - SEL/EESC/USP

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

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

AGRADECIMENTOS

Agradeço principalmente aos meus pais, Osmar Joaquim Matias e Cecília Maria Ribeiro da Costa, por todo o apoio que me deram ao longo de minha vida, pelas portas que abriram para mim e pela motivação constante, desde a entrada em uma faculdade renomada até o momento em que me encontro hoje. Sou igualmente grato à minha irmã, Júlia da Costa Matias, que me acompanha desde que tinha dois anos de idade, e com quem pude compartilhar meus anos na universidade, ainda que de forma distante, eu aqui e ela na UNIFAL, em Alfenas.

Também sou profundamente agradecido a todas as pessoas que contribuíram para o meu crescimento ao longo da graduação, especialmente as que conheci no Tupã, na LESC, no Consulting Club e no ambiente empreendedor da cidade de São Carlos. Esse convívio me proporcionou contato com diferentes visões de mundo e enriqueceu minha bagagem cultural e profissional. Tenho imensa gratidão também às pessoas com quem trabalhei durante meu estágio na Embraer, no programa do KC-390, onde vivenciei a rotina intensa de uma equipe de engenharia. Por fim, agradeço ao meu orientador, Prof. Dr. Pedro de Oliveira Conceição Junior, pela orientação e suporte ao longo deste projeto.

“ The world is complicated, [...].
That’s why it’s interesting.”
(OBAMA, 2020, p. 7)

RESUMO

MATIAS, V. C. **Ferramenta de monitoramento de barramento CAN para o protótipo automobilístico da EESC-USP Tupã**. 2024. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

O processo de aquisição de dados desempenha um papel crucial na engenharia, permitindo a análise do comportamento dos sistemas sob condições adversas e a detecção precoce de falhas de projeto. No setor automobilístico, essa prática é indispensável tanto na validação de conceitos quanto na identificação de falhas antes de incidentes. Este trabalho tem como objetivo desenvolver um equipamento de registro de dados de baixo custo, independente da arquitetura do projeto, destinado à validação do protótipo da equipe EESC-USP Tupã. A necessidade deste dispositivo emergiu devido ao fato de que a telemetria é frequentemente relegada a uma prioridade inferior nos projetos, em virtude da elevada demanda de atividades enfrentada pela equipe de desenvolvimento de *software*. Para resolver esse problema, foi desenvolvido um sistema que monitora o barramento CAN do protótipo e transmite os dados via protocolo LoRa para um receptor, que os processa e exibe em um computador de forma clara e intuitiva. O projeto visa garantir que a leitura dos dados atenda aos requisitos da equipe, como alcance das antenas superior à extensão da pista e exibição clara dos parâmetros, sem interferir na arquitetura ou comunicação dos componentes. Este estudo tem potencial para otimizar a manufatura e a validação de futuros projetos da equipe, refinando os testes e a análise de desempenho. Por fim, o equipamento desenvolvido atendeu aos requisitos do projeto, incluindo o alcance das antenas e a exibição dos dados e custos. Além disso, foi gerada uma documentação utilizando a ferramenta *GitHub Pages* para garantir a retenção de conhecimento.

Palavras-chave: CAN. LoRa. Logger. Dashboard. Automobilismo. Tupã.

ABSTRACT

MATIAS, V. C. **CAN bus monitoring tool for the EESC-USP Tupã automotive prototype**. 2024. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

The data acquisition process plays a crucial role in engineering, enabling the analysis of system behavior under adverse conditions and early detection of design issues. In the automotive sector, this practice is essential for both concept validation and fault detection before incidents occur. This work aims to develop a low-cost data logger device, independent of the project's architecture, for the validation of the EESC-USP Tupã team's prototype. The necessity for this device arose from the fact that telemetry is often deprioritized in projects due to the high demand for activities faced by the software development team. To address this issue, a system was developed to monitor the prototype's CAN bus and transmit the data via LoRa protocol to a receiver, which processes and displays it on a computer in a clear and intuitive manner. The project seeks to ensure that data acquisition meets the team's requirements, such as antenna range exceeding the track length and clear presentation of parameters, without interfering with the architecture or communication of the components. This study has the potential to optimize the manufacturing and validation of future team projects by refining testing and performance analysis. Finally, the developed equipment met the project requirements, including the range of the antennas and the display of data and costs. In addition, documentation was generated using the *GitHub Pages* tool to ensure knowledge retention.

Keywords: CAN. LoRa. Logger. Dashboard. Motorsport. Tupã

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CAN	*Controller Area Network*
LoRa	*Long Range*
RSSI	*Received Signal Strength Indication*
SNR	*Signal-to-noise ratio*
RTOS	*Real-Time Operating System*
ISR	*Interrupt Service Routines*
ESP32	*Espressif Systems* de 32 bits.
TX	Transmissor
RX	Receptor
SQL	*Structured Query Language*
SAE	*Society of Automotive Engineers*
ECPA	Esporte Clube Piracicabano de Automobilismo
ECU	*Eletronic Control Unit*
TWAI	*Two-Wire Automotive Interface*
UART	*Universal Asynchronous Receiver / Transmitter*
SPI	*Serial Peripheral Interface*
API	*Application Programming Interface*

SUMÁRIO

1 INTRODUÇÃO

A competição Formula SAE BRASIL tem como objetivo proporcionar aos estudantes de Engenharia a chance de colocar em prática e em equipe os conhecimentos adquiridos em sala de aula, por meio do desenvolvimento de um veículo do tipo Fórmula que atenda as regras pré-estabelecidas pela comissão organizadora. Meses antes da competição, os estudantes enviam para o comitê organizador relatórios de custos, estrutura e projeto. Os relatórios são avaliados por engenheiros especialistas e são a primeira parte da avaliação dos protótipos (SAE-BRASIL, 2024).

A competição ocorre durante o período de três dias no Esporte Clube Piracicabano de Automobilismo (ECPA) em Piracicaba, onde são realizados testes estáticos e dinâmicos com os carros, com o objetivo de avaliar o desempenho de cada projeto na pista. Além disso, as equipes apresentam suas propostas técnicas, incluindo detalhes sobre o projeto, custos e uma apresentação de *marketing*. Durante a competição, nas provas estáticas, as equipes devem demonstrar mais detalhadamente se o carro apresentado no projeto equivale ao que foi apresentado. As provas dinâmicas são realizadas no segundo dia do evento. Todas as provas são pontuadas, de maneira a garantir que o melhor conjunto de projeto e carro vença a competição (SAE-BRASIL, 2024).

A pista onde a prova de carros elétricos ocorre está presente na Figura 1, a qual possui um diâmetro de aproximadamente 200 m (ECPA, 2024). Trata-se de uma pista de campo aberto, ou seja, não há obstáculos entre o veículo e a equipe. Além disso, os carros correm individualmente durante as provas dinâmicas. Para que o protótipo possa participar destas provas, ele deve ser aprovado em todos os testes de inspeção, que consistem em blindagem contra água, verificação estrutural e de segurança elétrica.

Figura 1 – Vista aérea da pista onde ocorre a competição (ECPA, 2024).

A Equipe EESC USP Tupã é uma das equipes de fórmula que adere a esta competição estudantil. O Tupã projeta, manufatura e valida um protótipo de um carro elétrico de alto desempenho. A equipe foi oficialmente fundada em 2012, tendo o projeto idealizado pelos estudantes da Escola de Engenharia de São Carlos. O primeiro lançamento de um protótipo ocorreu no ano de 2015, com o nome de T01, mas só veio a ser campeã nacional em 2021, durante a pandemia. Na última edição da competição que ocorreu em 2024, foi conquistado o 3º lugar de 23 equipes (EESC, 2024) com o protótipo T08, presente na Figura 2.

Figura 2 – Protótipo desenvolvido em 2024 durante a competição.

O processo de aquisição de dados tem sido um desafio na equipe durante sua história, devido à perda de conhecimento que ocorre periodicamente, quando membros mais antigos deixam a equipe por motivos de trabalho ou formatura. Também é comum o projeto de telemetria ser deixado em segundo plano, pois os esforços do time estão voltados em atividades de maior prioridade como os projetos de controle e de sensoriamento. Desta forma, a telemetria acaba por não ser terminada a tempo da competição.

No entanto, o processo de coleta de dados é essencial para o desenvolvimento de um projeto de engenharia, pois ele permite compreender o comportamento do veículo em situações adversas, a detecção precoce de falhas e possibilita um melhor entendimento do desempenho dos componentes e otimizar processos de manufatura. Os dados podem ser coletados tanto em testes que ocorrem no próprio Campus da USP quanto durante as provas da competição. Ademais, com esse processo cria-se um sistema em malha fechada onde os resultados das medições em testes geram aperfeiçoamento no projeto, os quais exigem novos testes e criam uma demanda maior por dados (LI; WANG; KANG, 2021).

1.1 Objetivos

Por este motivo, o projeto tem como objetivo desenvolver um equipamento de registro de dados de baixo custo, que seja independente da arquitetura do projeto, para a validação do protótipo da equipe EESC-USP Tupã. Notou-se que alguns times nacionais e outros internacionais fazem o uso de equipamentos de registro em seu projeto, no entanto, esse tipo de equipamento apresenta alto custo.

A Figura 3 apresenta um esquema do projeto proposto. Para desenvolvê-lo, foram utilizados microcontroladores ESP32 e periféricos destinados à leitura e ao envio de dados, como transceptores de CAN e LoRa. Esses dispositivos operam por meio de protocolos de comunicação, como UART e SPI.

No projeto, o protocolo UART foi empregado em dois contextos distintos. Primeiramente, ele foi utilizado para estabelecer a comunicação entre o transceptor CAN (SN65HVD230) e a ESP32. Em um segundo momento, o mesmo protocolo foi empregado na transmissão de dados entre a ESP32 (conFigurada como transmissor) e o computador, utilizando uma taxa de transmissão de 115200 bps. Este protocolo tem potencial de alcançar até 10 m de distância entre o transmissor e o receptor.

Já o protocolo SPI foi utilizado para a comunicação entre o transceptor LoRa (SX1278) e as ESP32 presentes tanto no receptor quanto no transmissor a comunicação SPI operou a uma frequência de 8 MHz. Já a comunicação entre os dispositivos LoRa operou em uma frequência de 433 MHz com um fator de espalhamento de 10. Este protocolo tem alcance de até um metro de distância entre trasnmissor e receptor.

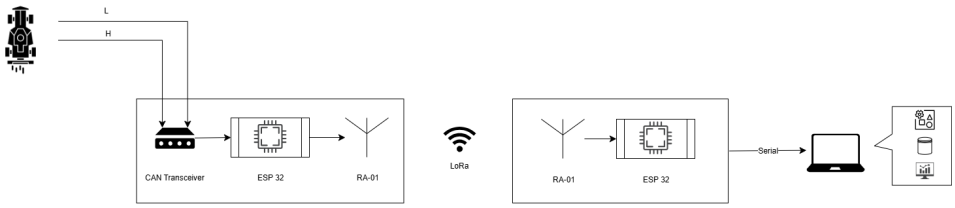

Figura 3 – Esquema do projeto proposto.

1.1.1 Objetivos Específicos

Desta forma, foram listados os requisitos que o projeto deve atender. Este método foi utilizado pois o levantamento de requisitos em um projeto de engenharia é essencial para garantir a definição clara e precisa das expectativas e necessidades das partes envolvidas. Assegurando que o produto final atenda às especificações funcionais e de desempenho estipuladas (IEEE, 1998). Levando em consideração o escopo deste projeto e as características do "cliente", uma equipe universitária composta por estudantes de graduação da Escola de Engenharia de São Carlos, cujo objetivo é desenvolver um protótipo

sem fins lucrativos para participar da competição organizada pela SAE-Brasil, os requisitos são:

- O equipamento de leitura de dados e registro do barramento CAN deve ter custo inferior ao de outras alternativas presentes no mercado.
- O equipamento de leitura e registro de dados do barramento CAN deve garantir que o alcance de sua antena seja maior que 200 metros, considerando o diâmetro da pista da competição.
- O equipamento de leitura e registro de dados do barramento CAN deve incluir documentação clara e completa (manual de instalação, uso e desenvolvimento), facilitando a colaboração de terceiros no projeto.
- O equipamento de leitura e registro de dados do barramento CAN deve ser capaz de realizar leituras com uma frequência mínima de 100 Hz, garantindo precisão e consistência nos dados coletados.
- O equipamento de leitura e registro de dados do barramento CAN deve ser capaz de registrar os estados dos seguintes componentes do veículo:
  - APPS (Accelerator Pedal Position Sensor): posição atual do pedal do acelerador;
  - Buzzer: estado (ativo ou inativo);
  - Brake Light: estado (ativo ou inativo) da luz de freio;
  - Botão de Partida: estado (pressionado ou não pressionado);
  - Shutdown: status de ativação do sistema de desligamento de emergência.

2 REVISÃO BIBLIOGRÁFICA

Na primeira parte, será apresentado o conceito de sistemas embarcados e telemetria, pois são neles que este projeto se sustenta. Além disso, é feita uma contextualização para o setor automobilístico. Também é exposta a ideia central do dispositivo desenvolvido, bem como onde ele é aplicado no contexto deste trabalho. Em seguida, são introduzidos os principais protocolos empregados, CAN, LoRa, UART, SPI também é feito uma justificativa do porquê eles foram adotados no projeto. Bem como, é discutido como os dados obtidos serão armazenados e exibidos para o usuário final, em um contexto de engenharia. Por fim, é exposta as principais soluções utilizadas para fazer o gerenciamento de acesso a esses dados. Concluiremos retomando a proposta inicial à luz das discussões desenvolvidas ao longo da revisão.

2.1 Sistemas Embarcados

Um sistema embarcado é um conjunto de *hardware* e *software* projetado para executar uma tarefa específica, muitas vezes integrado a um escopo maior. Esses sistemas são utilizados para controlar ou monitorar funções ou processos, sendo, em geral, desenvolvidos com foco em simplicidade e baixo custo. Seus componentes principais incluem microcontroladores, sensores e atuadores, que, juntos, garantem a eficiência da operação (SOUZA, 2022).

Atualmente, os sistemas embarcados estão amplamente presentes em nosso cotidiano, com aplicações que vão desde produtos domésticos cotidianos, como eletrodomésticos, até em áreas de tecnologia de ponta, como equipamentos médicos. Eles também são encontrados nos setores financeiros, distribuição de energia, telecomunicações, transportes, construção civil, aeroespacial, entre outros (UEL, 2024).

No âmbito do automobilismo, esses sistemas estão em quase todos os lugares, desde simples sensores para detecção do encaixe de cintos de segurança quando o veículo está em movimento, quanto para regular partes mais funcionais como a direção ou frenagem. Eles também são empregados em funções mais sofisticadas como em assistentes de direção (CHEN et al., 2009), medindo uma distância ou enviando alertas para notificar o motorista sobre uma possível colisão (KATARE; EL-SHARKAWY, 2019), ou até mesmo direção autônoma (MUSTAFA; KHABOUR; MUSTAFEH, 2024).

2.2 Telemetria

O avanço do poder computacional ocorrido nas últimas décadas nos dispositivos microcontroladores, bem como a constante diminuição do tamanho desses equipamentos, catalisou um grande avanço no sistema de sensoriamento remoto. Isso proporcionou fenômenos como o da indústria 4.0 e internet das coisas. No âmbito do automobilismo, a

telemetria apresenta papel crucial pois fornece dados em tempo real sobre o desempenho do veículo, bem como o comportamento do piloto. Dentre os dados mais comuns podemos encontrar a velocidade do veículo, consumo de gasolina ou bateria, e temperatura dos motores e pneus. Com esses dados, os engenheiros podem tomar decisões estratégicas imediatas, ajustar configurações do carro, além de ajudar na identificação de problemas mecânicos antes que se tornem críticos, permitindo a realização de manutenções preventivas e aumentando a segurança dos pilotos (TOTVS, 2024).

2.3 Protocolo *Controller Area Network* (CAN)

Para obtenção dos dados do veículo, foi utilizado o protocolo CAN (Controller Area Network), o qual foi desenvolvido pela BOSCH como um sistema de transmissão de mensagens multimestre. O protocolo é definido pela ISO-11898: 2003 desenvolvido originalmente para a indústria automotiva para substituir o complexo chicote de fiação por um barramento de dois fios. No entanto, o protocolo foi difundido e se popularizou em uma variedade de indústrias, incluindo automação predial, médica e manufatura. A especificação do protocolo CAN exige alta imunidade à interferência elétrica e a capacidade de autodiagnosticar e reparar erros de dados. Além disso, o protocolo especifica uma taxa máxima de sinalização de 1 megabit por segundo (bps). Ao contrário de uma rede tradicional como USB ou Ethernet, o CAN não envia grandes blocos de dados entre seus nós (CORRIGAN, 2002).

No protocolo CAN há uma rede composta por um conjunto de nós (dispositivos) que são interconectados por um par de fios de transmissão de dados, conhecido como barramento, esses fios são chamados de CAN high e CAN low. Os dados transmitidos são recebidos por todas as outras ECUs na rede, então cada uma verifica os dados e decida se deseja aceitá-los ou ignorá-los, como ilustra a Figura 4. A comunicação no protocolo é feita via CAN frames, ilustrado na Figura 5, que consiste na mensagem enviada pelo barramento, composta por oito sessões definidas na Tabela 1.

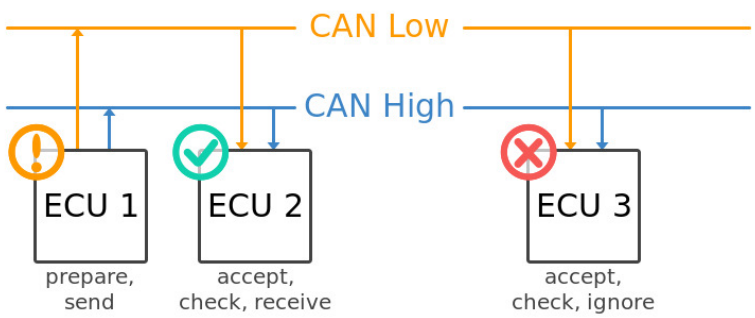

Figura 4 – Diagrama da Rede CAN, mensagem enviada, recebida e ignorada (CSS-ELECTRONICS, 2023).

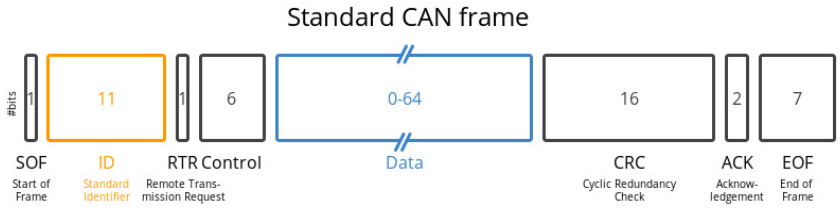

Figura 5 – Estrutura de um Frame do protocólo CAN (CSS-ELECTRONICS, 2023).

Tabela 1 – Estrutura do Frame CAN

Sigla	Descrição	Definição
SOF	Start of Frame	Bit dominante de valor ‘0’, indica o início de uma nova mensagem CAN, alertando os nós sobre a transmissão.
ID	Identificador do Frame	Define a prioridade do frame no barramento, com prioridade descendente.
RTR	Remote Transmission Request	Informa se a mensagem está enviando ou solicitando dados.
Control	Controle	Inclui o bit de extensão do identificador (IDE), que é dominante para identificadores de 11 bits, e o código de comprimento de dados (DLC) de 4 bits, que especifica o número de bytes de dados (0 a 8) a serem transmitidos.
Data	Dados de Payload	Contêm a carga útil da mensagem, constituída por bytes de dados que incluem sinais CAN específicos, os quais podem ser extraídos e decodificados para obter informações relevantes do sistema.
CRC	Verificação de Redundância Cíclica	Método para verificar a integridade dos dados transmitidos, garantindo que não houve erros durante a transmissão do frame.
ACK	Acknowledgment Slot	Confirma que o frame foi recebido com sucesso por ao menos um nó, garantindo a comunicação efetiva na rede.
EOF	End of Frame	Delimita o final da mensagem CAN, assegurando que todos os nós reconheçam o término do frame de dados.

2.4 Registrador de Dados de um Barramento CAN

Um *CAN Bus Data Logger* é um dispositivo utilizado para capturar e registrar mensagens transmitidas em uma rede CAN, amplamente utilizada em automóveis e sistemas industriais. Esse tipo de equipamento de registro monitora o barramento de comunicação, capturando dados como a velocidade do veículo, status dos sensores, e diagnósticos do motor, para análise posterior ou em tempo real (ELECTROMATE, 2024). Ele é particularmente útil em testes, diagnósticos e desenvolvimento de sistemas embarcados, permitindo a análise de comportamento do veículo ou equipamento em diversas condições operacionais

(PHYTOOLS, 2024).

2.5 Protocolos de Comunicação com Fio

Neste projeto, dois protocolos de comunicação amplamente utilizados foram empregados para garantir a transferência eficiente de dados entre os dispositivos. A primeira parte envolveu o uso da comunicação *Universal Asynchronous Receiver-Transmitter* (UART), enquanto a segunda utilizou a *Serial Peripheral Interface* (SPI). Ambos os protocolos desempenharam papéis essenciais na troca de informações entre a ESP32, o transceptor de CAN e o computador. A seguir, apresentamos uma explicação sobre o funcionamento de cada um desses protocolos. Mesmo a CAN sendo um protocolo de comunicação com fio, ela recebeu uma seção própria por causa de sua relevância para o presente projeto.

2.5.1 *Universal Asynchronous Receiver-Transmitter* (UART)

Neste projeto, a comunicação UART foi utilizada em dois momentos: o primeiro deles para fazer a comunicação entre o transceptor de CAN e a ESP32. Já em um segundo momento, esse protocolo foi utilizado para transmitir dados entre o transmissor, também uma ESP32, e o computador, a um *baud rate* de 115200 bps.

A UART opera transmitindo dados como uma série de bits, incluindo um bit de início, bits de dados, um bit de paridade opcional e bits de parada. Diferentemente da comunicação paralela, em que vários bits são transmitidos simultaneamente, a UART envia dados em série, um bit de cada vez. Como o nome revela, o protocolo opera de forma assíncrona, o que significa que ele não depende de um sinal de relógio compartilhado. Em vez disso, ele usa *baud rates* predefinidas para determinar o tempo dos bits de dados (SIEBENEICHER, 2024).

Os principais componentes da UART incluem o transmissor, o receptor e a taxa de transmissão. O transmissor coleta dados de uma fonte, formata-os em bits seriais e os envia por meio de um pino TX (*Transmit*). O receptor os recebe por meio de um pino RX (*Receive*), processa os dados seriais recebidos e os converte em dados paralelos para o sistema hospedeiro. A taxa de transmissão determina a velocidade da transmissão de dados (SIEBENEICHER, 2024).

2.5.2 *Serial Peripheral Interface* (SPI)

O SPI é uma interface síncrona e *full duplex* (os dados podem ser transmitidos e recebidos simultaneamente em ambas as direções), baseada em uma arquitetura de principal e subnós. Os dados entre o principal e o subnó são sincronizados na borda ascendente ou descendente do sinal de relógio. Tanto o principal quanto o subnó podem transmitir dados simultaneamente. A interface SPI pode ser configurada com 3 ou 4 fios, e o

dispositivo que gera o sinal de relógio é denominado principal. Os dados transmitidos entre o principal e o subnó são sincronizados com o *clock* gerado pelo principal (DHAKER, 2018).

O sinal de seleção de *chip* do principal é utilizado para selecionar o subnó. Tipicamente, este é um sinal ativo em nível baixo, que é puxado para alto para desconectar o subnó do barramento SPI. Quando vários subnós são utilizados, um sinal de seleção de chip individual para cada subnó é necessário a partir do principal. MOSI e MISO são as linhas de dados: MOSI transmite dados do principal para o subnó, enquanto MISO transmite dados do subnó para o principal (DHAKER, 2018).

2.6 Protocolos de Comunicação sem Fio

No projeto, foi utilizado o protocolo de comunicação de rádio frequência LoRa. Nesta seção, apresentamos os motivos que justificam a escolha deste protocolo, bem como uma comparação entre o LoRa e outros protocolos disponíveis no mercado, como Wi-Fi, *Bluetooth* e ZigBee. Exceto o LoRa, os protocolos mencionados estão integrados no microcontrolador utilizado no projeto, eliminando a necessidade de periféricos adicionais.

2.6.1 Protocolo LoRa

LoRa, abreviação de *long range*, consiste em uma técnica de modulação derivada da tecnologia *chirp spread spectrum* (CSS). A plataforma LoRa da Semtech, possui longo alcance e baixo consumo de energia, por isso tornou-se padrão para projetos de Internet das Coisas (IoT). Dispositivos e redes LoRa, como o LoRaWAN possibilitam soluções para vários desafios de engenharia como o gerenciamento de energia, diminuição do uso de recursos naturais, controle de poluição, aumento da eficiência de infraestrutura e prevenção de desastres (Semtech Corporation, 2015).

O módulo LoRa escolhido foi o RA-01 da fabricante chinesa Ai-Thinker, o componente foi baseado no *chip* SX1278, o qual opera na faixa de frequência de 410 a $525MHz$, sendo amplamente utilizado para comunicação de espectro de propagação de distância ultra longa. O módulo utiliza interface de comunicação SPI com taxa de bits programável de até $300kbps$, com potência máxima de saída de $20dBm$ e sensibilidade de $-141dBm$. Por fim, é importante mencionar que o RA-01 possui tensão de alimentação de $3,3V$ e deve ter uma corrente superior a $200mA$, caso contrário a transmissão ou o alcance são comprometidos (Ai-Thinker Technology Co., Ltd., 2024).

Os principais parâmetros para avaliar a qualidade de recepção de um sinal LoRa são o RSSI e a SNR. Ambos esses parâmetros são fornecidos pelo componente RA-01 e são essenciais para garantir a integridade e a eficiência da comunicação. O RSSI e a SNR são utilizados para monitorar e otimizar o desempenho do sistema, permitindo ajustes finos que asseguram uma transmissão de dados robusta e confiável. Através da análise desses

parâmetros, foi possível identificar possíveis interferências ou problemas de sinal, tomando medidas corretivas para manter a qualidade da comunicação em níveis ideais.

O RSSI, *Received Signal Strength Indicator*, é uma métrica utilizada em sistemas de comunicação sem fio para medir a potência do sinal recebido por um dispositivo, expressa em decibéis-miliwatts (dBm). Esse valor indica a intensidade do sinal que chega ao receptor, sendo utilizado para avaliar a qualidade da comunicação entre o transmissor e o receptor. O RSSI geralmente varia de valores mais próximos de $0dBm$, que indicam um sinal forte, a valores negativos mais baixos, como $-100dBm$, que correspondem a sinais mais fracos e de menor confiabilidade. No contexto de redes de longa distância e baixa potência, como LoRa, o RSSI é crucial para determinar a viabilidade de enlaces de comunicação em cenários desafiadores, onde o sinal pode ser degradado por interferências, distâncias elevadas ou obstáculos físicos (The Things Network, 2024).

Já a Razão Sinal-Ruído (SNR) é a relação entre a potência do sinal recebido e o nível de potência do ruído. Normalmente, o piso de ruído é o limite físico de sensibilidade, no entanto, o LoRa opera abaixo desse nível. Os valores típicos de SNR estão entre $-20dB$ e $+10dB$. Um valor mais próximo de $+10dB$ indica que o sinal recebido está menos corrompido. Além disso, esse protocolo tem a capacidade de demodular sinais entre $-7,5dB$ a $-20dB$ abaixo do piso de ruído. Esta característica permite operações em condições nas quais outros sistemas falhariam, dessa forma garante robustez e adaptabilidade em ambientes de comunicação desafiadores(The Things Network, 2024).

Nesse projeto foi utilizado a forma de comunicação chamada LoRa P2P (*Peer to Peer*) na qual ao invés da criação de uma rede de comunicação complexa, como é feito no caso do LoRaWan, foi implementado um sistema simples no qual duas antenas trocam mensagens. Uma segunda simplificação foi feita, na qual uma das antenas é sempre um receptor e outra é sempre um transmissor. A antena Receptora foi conectada a um computador para realizar a comunicação com o sistema de banco de dados. Já a antena transmissora foi instalada no dispositivo de medição.

2.6.2 Wi-Fi

O Wi-Fi teve sua origem no Havaí em 1971, com a criação da rede de pacotes UHF sem fio chamada ALOHAnet, utilizada para conectar as ilhas. Protocolos subsequentes, desenvolvidos em 1991 pela NCR e AT&T e denominados WaveLAN, tornaram-se os precursores dos padrões IEEE 802.11 (WaTech, 2024). Atualmente, o Wi-Fi é amplamente utilizado em ambientes domésticos e vias públicas, operando nas frequências de 2,4 GHz, 5 GHz e 6 GHz. Seu alcance varia entre 30 e 100 metros, dependendo da frequência e do ambiente. A taxa de transmissão pode variar de 600 Mbps até 7 Gbps.

2.6.3 *Bluetooth*

O *Bluetooth* é uma tecnologia de comunicação sem fio padronizada pelo IEEE 802.15.1, projetada para conexões de curto alcance em aplicações de baixo consumo de energia. Opera em frequências de 2,4 GHz utilizando a técnica de espectro espalhado por salto de frequência (FHSS), o que reduz interferências. Seu alcance varia conforme a classe do dispositivo variando de 10 a 100m, dependendo das condições ambientais e tipo de dispositivo. As taxas de transmissão de dados também diferem entre versões com taxa de dados entre 1 Mbps e 3 Mbps (INTEL, 2024).

2.6.4 ZigBee

Em 2002, foi criada a *ZigBee Alliance*, uma associação de empresas, universidades e agências governamentais com o objetivo de desenvolver o protocolo ZigBee. Este protocolo sem fio, baseado no padrão IEEE 802.15.4, foi projetado para redes de baixo consumo de energia e alta densidade de dispositivos, com foco em aplicações de IoT. Por isso, é comum sua utilização em monitoramento e controle industrial, automação residencial e sistemas de energia (UFRJ, 2017). O ZigBee opera principalmente na frequência de 2,4 GHz, mas também pode usar 868 MHz e 915 MHz em algumas regiões, com taxas de transmissão de dados variando entre 20 kbps e 250 kbps, dependendo da banda de frequência. Com alcance de 10 a 75 metros, este protocolo é altamente eficiente no consumo de energia, permitindo que dispositivos operem por anos com baterias de baixa capacidade.

2.6.5 Escolha do Protocolo

Entre os protocolos citados, foi escolhido o LoRa devido ao seu maior alcance em comparação com os outros protocolos avaliados. Embora apresente a menor taxa de dados entre eles, essa taxa é suficiente para atender à demanda do projeto, como indica a Tabela 2. Além disso, o LoRa possui elevada imunidade a obstáculos, uma característica crucial para um equipamento destinado ao uso em campo.

Tabela 2 – Comparação entre os protocolos Wi-Fi, Bluetooth, ZigBee e LoRa (ABDERRAHMANE; NOURREDINE; MOHAMMED, 2024).

Critério	**Wi-Fi**	**Bluetooth**	**ZigBee**	**LoRa**
Frequência de banda	2.4 e 5 GHz	2.4 GHz	2.4 GHz (Global)	868 MHz (Europa), 915 MHz (America do Norte), 433 MHz (Asia)
Taxa de dados	600 Mbps- 7 Gbps	1 - 3 Mbps	20-250 kbps	0.3 - 50 kbps
Consumo de potência	Relativamente alto	Baixo a moderado	Muito baixo	Muito baixo
Imunidade a obstáculos (Shadowing)	Moderado a baixo	Moderado a alto	Alto	Alto

2.7 Sistemas Operacionais de Tempo Real

Neste projeto, foi utilizado um sistema operacional de tempo real, o *FreeRTOS*. Assim, esta seção tem como objetivo, primeiramente, apresentar uma definição desse tipo de sistema e discutir os principais conceitos relacionados. Além disso, são destacadas as vantagens que ele pode oferecer em projetos de engenharia. Posteriormente, esta seção fundamenta a escolha do sistema operacional de tempo real para o projeto. Para isso, foram listados os *RTOS* disponíveis no mercado e, ao final, realizado um comparativo entre eles, justificando a escolha do *FreeRTOS*.

Para compreender os Sistemas Operacionais de Tempo Real (RTOS), é importante começar pela definição de um sistema operacional. De acordo com (ANH; TAN, 2009), um sistema operacional é uma coleção especializada de programas que gerenciam os recursos físicos de um computador. No entanto, o conceito de "tempo real"pode ser amplamente mal interpretado. Na linguagem cotidiana, muitas pessoas associam "em tempo real"a algo "imediato"ou "instantâneo"(TANENBAUM; BOS, 2022). Embora esses sistemas sejam caracterizados pela relevância do tempo como parâmetro essencial, um sistema em tempo real é, na realidade, um sistema computacional que deve atender a restrições temporais rígidas cujo descumprimento pode levar a consequências graves, incluindo falhas críticas (LAPLANTE, 2011). Os sistemas em tempo real podem ser classificados em três categorias principais: *hard real-time*, *soft real-time* e *firm real-time systems*. A Tabela 3 apresenta um resumo dessas categorias, incluindo exemplos práticos.

2.7.1 Sistemas *Hard Real-Time*

Devem fornecer garantias absolutas de que uma ação ocorrerá dentro de um prazo específico. A falha em cumprir esse prazo pode resultar em catástrofe. Por exemplo, em uma linha de montagem automotiva, ações como a soldagem de componentes precisam ocorrer

em momentos precisos; um erro de tempo pode comprometer todo o veículo (LAPLANTE, 2011).

2.7.2 Sistemas *Soft Real-Time*

Permitem algum nível de flexibilidade em relação aos prazos. Embora a perda ocasional de um prazo seja indesejável, ela é aceitável e não causa danos permanentes. Exemplos incluem sistemas de áudio ou multimídia digital (LAPLANTE, 2011).

2.7.3 Sistemas *Firm Real-Time*

Representam uma categoria intermediária entre os dois anteriores. Nesse caso, perder prazos esporádicos não implica falha total, mas exceder um certo limite pode causar danos graves ou falhas catastróficas ao sistema (LAPLANTE, 2011).

Tabela 3 – Exemplos de sistemas em tempo real (LAPLANTE, 2011).

Classificação	**Sistema**	**Explicação**
Hard	Sistema de lançamento de armas aviônicas em que pressionar um botão lança um míssil ar-ar.	Perder o prazo para lançar o míssil dentro de um tempo especificado após pressionar o botão pode fazer com que o alvo seja perdido, o que resultará em catástrofe.
Firm	Controlador de navegação para um robô herbicida autônomo	Perder alguns prazos de navegação faz com que o robô se desvie de um caminho planejado e danifique algumas colheitas.
Soft	Jogo de hóquei para console	Perder até mesmo vários prazos só irá degradar o desempenho.

2.7.4 Componentes

Um sistema operacional em tempo real é constituído por diversos componentes. As próximas seções têm como objetivo definir aqueles abordados neste relatório, a saber: *tasks*, *scheduler* e *kernel*.

2.7.4.1 *Task*

Uma *task* (ou "tarefa", em português) é o objeto ativo de um sistema de tempo real e representa a unidade básica de processamento gerenciada pelo agendador (LAPLANTE, 2011). Ela é definida como uma *thread* com estado, incluindo elementos como pilha, registradores, contador de programa (PC), manipuladores de sinal, variáveis de tarefa, ID e nome da tarefa, prioridade, ponto de entrada e dados relacionados ao estado e à comunicação entre tarefas (SIEWERT, 2016).

2.7.4.2 *Scheduler*

Um *scheduler* (ou "agendador", em tradução livre) é o algoritmo utilizado por um *RTOS* para organizar a execução de tarefas. O agendador decide qual delas será executada em cada CPU do sistema em um dado momento como ilustra a Figura 6. Para isso, ele utiliza a fila de execução, que é composta por tarefas que podem estar em execução ou aguardando a oportunidade de serem executadas (SIEWERT, 2016).

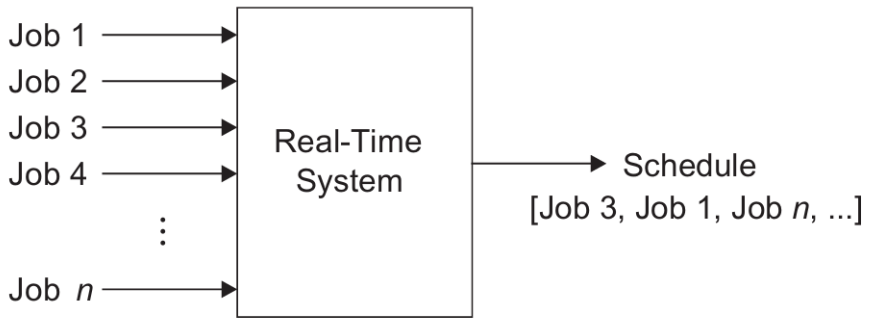

Figura 6 – Sistema em tempo real como uma sequência de tarefas programáveis (LAPLANTE, 2011).

O agendamento é uma função primária de sistemas operacionais, especialmente em ambientes de tempo real. Para atender aos requisitos temporais de um programa, é fundamental adotar uma estratégia sólida que ordene o uso dos recursos do sistema, além de prever o pior desempenho (ou tempo de resposta) associado a uma política de agendamento específica. De modo geral, as políticas de agendamento podem ser classificadas em duas categorias principais: pré-tempo de execução e tempo de execução. O objetivo central de ambas é garantir que as especificações de tempo de resposta sejam rigorosamente atendidas (LAPLANTE, 2011).

No agendamento de pré-tempo de execução, todas as decisões sobre a ordem de execução das tarefas são tomadas antes da execução do programa. Essa abordagem é útil em sistemas onde as tarefas são bem definidas e previsíveis. Por outro lado, o agendamento de tempo de execução toma decisões durante a execução do programa, permitindo maior flexibilidade e adaptação a eventos dinâmicos, como a chegada de novas tarefas ou mudanças nos requisitos do sistema (SIEWERT, 2016)..

2.7.4.3 *Kernel*

O *kernel* é o componente central do sistema operacional responsável pelo controle direto de todos os recursos críticos, como CPU, memória e dispositivos de entrada e saída. Ele atua como uma ponte entre o *hardware* e o *software*, sendo acessado por aplicativos por meio de *APIs*, por exemplo (SIEWERT, 2016).

2.7.5 Vantagens

Esta seção tem como objetivo explicar as principais vantagens de se utilizar um sistema em tempo real neste projeto. Conforme já descrito, a priorização de tarefas pode ajudar a garantir que um aplicativo cumpra seus prazos de processamento. Além disso, o *kernel* pode oferecer outros benefícios menos óbvios, como a modularidade, pois as tarefas são módulos independentes, cada um com um propósito bem definido (BARRY, 2016).

A abstração de informações temporais é outra vantagem significativa. O *kernel* é responsável pelo gerenciamento do tempo de execução e fornece uma *API* relacionada ao tempo para o aplicativo. Isso simplifica a estrutura do código, reduzindo o tamanho geral e as interdependências entre módulos. Como resultado, o software pode evoluir de forma mais controlada e previsível. Além disso, o desempenho do aplicativo torna-se menos suscetível a mudanças no *hardware* subjacente, uma vez que a abstração temporal é gerenciada pelo *kernel* (BARRY, 2016).

Outra vantagem relevante desses sistemas é a maior eficiência energética. Utilizar um *kernel* permite que o *software* seja totalmente orientado a eventos, evitando desperdício de tempo de processamento ao verificar eventos inexistentes. O código é executado apenas quando há algo a ser feito, permitindo que o processador permaneça por mais tempo em modos de baixo consumo de energia (BARRY, 2016).

2.7.6 *RTOSs* Presentes no Mercado

Nesta seção, são apresentados três dos sistemas operacionais em tempo real (RTOS) amplamente utilizados em aplicações embarcadas: EmbOS, FreeRTOS e Zephyr. Cada um deles possui características específicas que os tornam adequados para diferentes tipos de projetos, variando desde aplicações industriais robustas até dispositivos com recursos limitados. A seguir, são descritas suas principais características, funcionalidades e casos de uso, oferecendo uma visão geral de suas vantagens e aplicações no desenvolvimento de sistemas embarcados.

2.7.7 EmbOS

O EmbOS é uma família de sistemas operacionais em tempo real projetada para servir como base para o desenvolvimento de aplicativos embarcados, com quatro décadas de desenvolvimento pela Segger. O EmbOS está disponível para vários processadores, compiladores e ferramentas de desenvolvimento, o que demonstra alta compatibilidade com diferentes dispositivos. A família EmbOS inclui o *embOS-Safe*, *embOS-MPU*, *embOS-Base* e *embOS-Ultra*.

O EmbOS é adequado para uma ampla gama de aplicações, especialmente no setor industrial, onde frequentemente opera com microcontroladores ou processadores. Além

disso, possui certificações relevantes, como IEC 61508 SIL 3, IEC 62304 Class C e ISO 26262 ASIL D (Segger, 2023).

2.7.8 FreeRTOS

Distribuído gratuitamente sob a licença de código aberto MIT, o FreeRTOS inclui um kernel e um conjunto crescente de bibliotecas, sendo amplamente utilizado em diversos setores industriais (FreeRTOS, 2024). Desenvolvido com ênfase na confiabilidade e na facilidade de uso, o FreeRTOS é mantido pela *Real Time Engineers Ltd* em parceria com as principais empresas de chips do mundo há mais de uma década. Esse sistema operacional oferece um software premiado e de alta qualidade, ideal para aplicações em tempo real profundamente embarcadas que utilizam microcontroladores ou microprocessadores pequenos. Esses aplicativos geralmente combinam requisitos de tempo real *hard* e *soft* (BARRY, 2016).

2.7.9 Zephyr

O Zephyr Project é um esforço colaborativo de código aberto hospedado pela Linux Foundation. Ele reúne desenvolvedores e usuários para construir um sistema operacional pequeno, escalável e de tempo real, otimizado para dispositivos com recursos limitados e projetado para suportar diversas arquiteturas (Zephyr, 2023). O projeto Zephyr é neutro e permite que fornecedores de silício, OEMs, ODMs, ISVs e OSVs contribuam com tecnologias que reduzem custos e aceleram o tempo de lançamento de bilhões de dispositivos embarcados conectados ao mercado. Suas aplicações incluem sensores simples, dispositivos *wearables* de LED, modems e pequenos *gateways* sem fio. O Zephyr é modular, oferecendo flexibilidade e suporte a múltiplas arquiteturas (Zephyr, 2023).

2.7.10 Comparações

O sistema operacional de tempo real escolhido neste projeto foi o *FreeRTOS*. Na Tabela 4, apresenta-se uma comparação entre as principais características dos sistemas descritos. O principal motivo da escolha foi a vasta documentação disponível, tanto em sites da internet quanto em livros escritos pelo fabricante. Outro motivo foi o suporte a uma ampla variedade de microcontroladores e arquiteturas de processadores, facilitando a portabilidade de *hardware* futuras. O fato de esse sistema ser gratuito também contribuiu para a escolha final. Por fim, destaca-se o fato de que esse sistema já vem instalado de fábrica no ambiente de desenvolvimento da ESP32, ou seja, não foi necessário nenhum tipo de configuração prévia.

Tabela 4 – Comparação entre os sistemas operacionais de tempo real avaliados.

	EmbOS	***FreeRTOS***	***Zephyr***
Licença	Comercial	MIT license	Apache-2.0 license
Pegada de memória	1,7 kB	5 – 10 kB	7 – 8 kB
Comunidade	Menor, mas experiente	Grande, com suporte comercial (AWS)	Crescente, com suporte da Linux Foundation
Nativo na ESP32	Não	Sim	Não
Especialidade	Microcontroladores de baixo custo	IoT, dispositivos conectados	Aplicações industriais de alta performance

2.8 *Dashboard*

A utilização de dashboards, painel de exibição de dados em português, em projetos de Internet das Coisas é fundamental para a visualização e análise em tempo real dos dados coletados, permitindo que engenheiros e operadores acompanhem o desempenho do sistema e identifiquem anomalias rapidamente. Um exemplo de painel disponível no mercado, que ilustra essa funcionalidade, pode ser visualizado na Figura 7.

O painel desenvolvido para este projeto fornece uma visão detalhada e organizada de parâmetros críticos do veículo, incluindo dados de localização GPS, velocidade e aceleração. Além disso, são exibidas os estados de componentes essenciais, como a presença de panes elétricas ou possíveis sobrecargas da bateria, promovendo uma análise abrangente do estado operacional do veículo.

Para desenvolver a interface gráfica do painel, utilizou-se o pacote Streamlit, uma ferramenta de código aberto que permite criar aplicações web interativas e personalizadas com facilidade. O Streamlit, lançado em 2018 e cofundado por Adrien Treuille, Amanda Kelly e Thiago Teixeira, foi criado com o objetivo de fornecer uma interface gráfica de usuário (GUI) simplificada para aplicações em Python, facilitando a visualização de dados sem a necessidade de conhecimentos avançados em desenvolvimento web (MHADHBI, 2021). Essa escolha se justificou pela capacidade da ferramenta de atualizar os dados em tempo real e sua facilidade de integração com bibliotecas de visualização em Python, permitindo que o dashboard interativo atendesse aos requisitos de responsividade e praticidade exigidos pelo projeto.

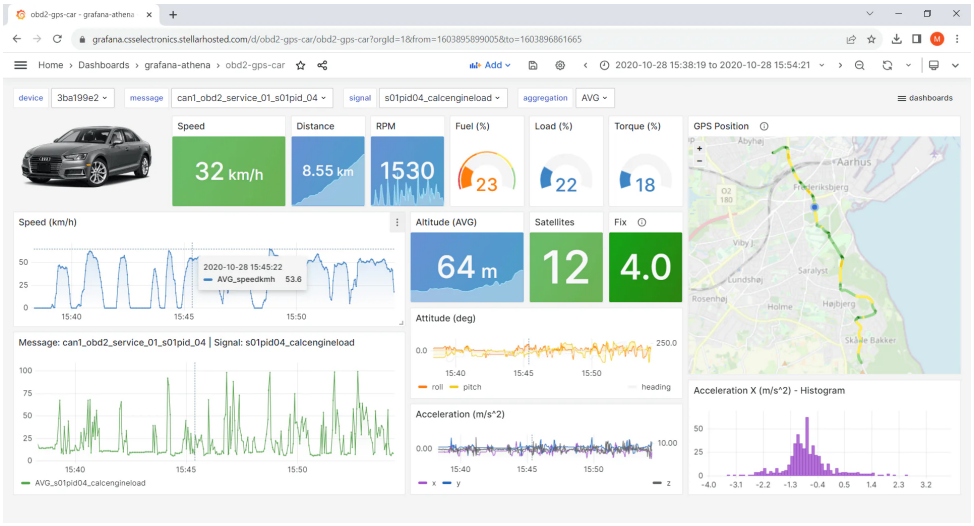

Figura 7 – Exemplo de *dashboard* automobilístico encontrado no mercado (CSS-ELECTRONICS, 2024).

Para fazer o armazenamento de dados foi utilizado um banco de dados SQLite. Devido a sua simplicidade de implementação e manutenção. Para realizar a programação do banco de dados foi utilizado o pacote de Python SQLAlchemy, o qual permite aos desenvolvedores acessar e gerenciar bancos de dados SQL usando a linguagem de domínio Pythonic (AWAN, 2024). Devido aos acessos simultâneos ao banco e períodos longos de espera de entradas, foi utilizado o modelo assíncrono de gerenciamento de entrada e saida. Para implementação foi utilizado a biblioteca *asyncio* do Python a qual facilitando a execução de múltiplas tarefas de forma concorrente sem bloquear o fluxo do programa. Com ela é possível lidar com várias requisições simultaneamente, permitindo que o programa continue executando outras tarefas enquanto aguarda uma resposta de uma operação (PYTHON-SOFTWARE-FOUNDATION, 2023).

No contexto do módulo de banco de dados, o asyncio foi utilizado para gerenciar consultas e inserções de dados de maneira eficiente, otimizando o desempenho e o tempo de resposta do sistema. No âmbito do projeto documentado nesta monografia, o CAN-BUS Data Logger foi desenvolvido como uma solução de baixo custo para atender as necessidades de validação de projeto da equipe EESC-USP TUPÃ. Para realizar a leitura dos dados foi utilizado a leitura dos frames presentes no protocolo CAN, para transmissão foi utilizado uma antena LoRa com alcance que atendesse as necessidades da equipe no contexto de prova, levando em consideração o tamanho das pistas. Por fim foi desenvolvida uma dashboard visando melhor atender as necessidades do usuário final.

3 MATERIAL E MÉTODOS

No desenvolvimento tanto dos receptores quanto dos transmissores foram utilizadas placas de desenvolvimento ESP32. Foi escolhido esta família de dispositivos devido ao baixo custo e alta quantidade de componentes já embarcadas na placa de desenvolvimento, como módulos de Wi-Fi e Bluetooth e um controlador de rede CAN, além de possuírem o sistema operacional FreeRTOS instalado no firmware por padrão. Os transceptores utilizados foram um SN65HVD230 e outro MCP2515, para leitura dos dados provenientes do veículo, o primeiro foi utilizado no no módulo que simulava o comportamento do veículo já que esse estava sendo em fases iniciais de projeto durante a execução do presente trabalho. Já o SN65HVD230 foi utilizado no transmissor, ou seja é parte integrante desse trabalho. Também foram utilizados dois módulos de antenas RA-01 cujo chip de rádio é o SX1278.

Para o desenvolvimento do código foi utilizado o framework de Arduino, pois ele oferecia um vasto conjunto bibliotecas para realizar a interface com os periféricos obtidos. Ademais foi utilizado o sistema operacional *FreeRTOS* para fazer o gerenciamento de tasks e interrupções, bem como da comunicação entre essas funções por meio de filas. Por fim foi utilizado o ambiente de desenvolvimento PlatformIO, uma extensão do Visual Studio Code, foi feita esta escolha porque é possível utilizar todo o ambiente de desenvolvimento do editor de texto e reutilizar o código entre placas da mesma familia.

Como o protótipo do Tupã não estava finalizado durante boa parte do desenvolvimento deste projeto, foi utilizado um conjunto de dados obtido em Ridha (2019) com dados de GPS para fazer parte da validação do projeto, através de um módulo que simula do comportamento do veículo. Estes dados forneciam informações de latitude, longitude e velocidade. Os dados de latitude e longitude, em especial, deveriam ter uma base de realidade pois há um mapa exibido a posição do veículo em um mapa e por isso deveria ter uma sequência lógica de valores. Os estados foram obtidas ou por operações matemáticas sobre os dados referenciados acima, ou por meio de valores simulados, a citar APPS, Buzzer, Brake, Botão de Partida e Shutdown.

Foram desenvolvidos dois dispositivos e uma *dashboard*, tela de exibição de dados. O dispositivo de transmissão é responsável por ler os dados do barramento CAN do veículo e transmitir esses dados por meio do protocolo LoRa para o receptor, ele está esquematizado na Figura 11. O transceiver de CAN utilizado no TX, conectado ao veículo, foi utilizado o modo de operação *Listen Only Mode*, pois este modo impede que o dispositivo participe da atividade do barramento. Neste modo de operação as transmissões de mensagens, confirmação, quadros de erro serão desativadas. Entretanto, o controlador CAN recebe mensagens sem confirmação. O que o torna ideal para a aplicação do projeto (ESPRESSIF, 2024).

3.1 Periféricos de CAN

Neste projeto, dois dispositivos foram utilizados para interfacear com o protocolo CAN: o MCP2515 e o SN65HVD230. O MCP2515 foi empregado no microcontrolador que simulava o comportamento do veículo, desempenhando um papel auxiliar para testes e validações, mas não fazendo parte do projeto final. Já o SN65HVD230 é o dispositivo responsável por fazer a interface com a rede CAN no projeto, na Tabela 5 há um comparativo entre esses dois dispositivos.

O MCP2515, presente na Figura 8, é responsável pela camada de enlace da comunicação, implementando o protocolo CAN, incluindo funções como controle de acesso ao barramento e manipulação de mensagens padrão e estendidas. Ele se comunica com o microcontrolador por meio do protocolo SPI e precisa de um transceptor CAN para interfacear com o barramento físico. Esse controlador é amplamente usado para adicionar suporte a CAN em microcontroladores que não possuem essa funcionalidade embutida. Entre suas características estão a compatibilidade com taxas de dados de até 1 Mbps, *buffers* internos para mensagens e suporte a mensagens padrão e estendidas (MICROCHIP, 2019).

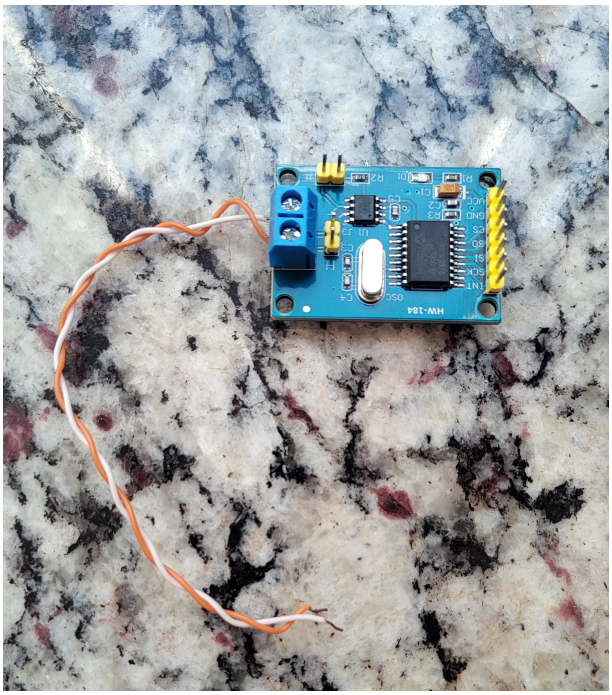

Figura 8 – Periférico MCP2515.

Por outro lado, o SN65HVD230, presente na Figura 9, atua na camada física, convertendo os sinais digitais do controlador em sinais diferenciais para o barramento CAN e vice-versa. Ele se conecta ao microcontrolador integrado e ao barramento físico por meio dos terminais CANH e CANL. Este transceptor possui alta imunidade a ruídos, suporte a velocidades de até 1 Mbps e é compatível com tensões de 3,3V, sendo ideal para sistemas modernos (TI, 2018).

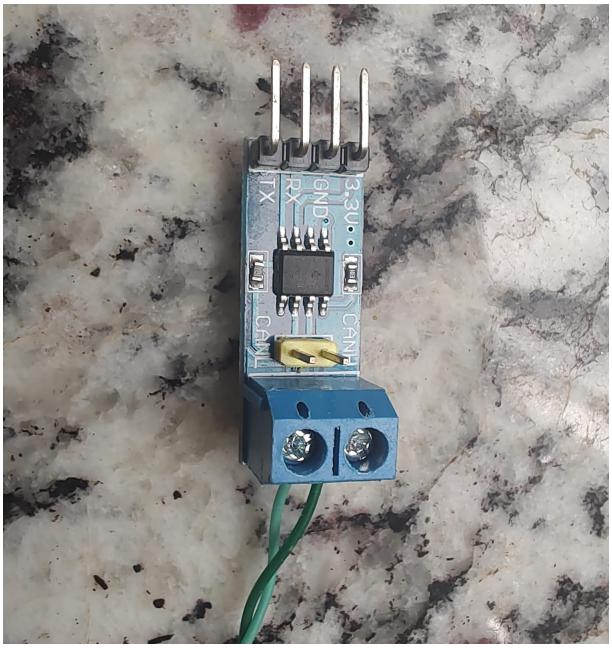

Figura 9 – Periférico SN65HVD230.

Tabela 5 – Comparativo entre os transceptores CAN MCP2515 e SN65HVD230.

Característica	**MCP2515**	**SN65HVD230**
Função	Controlador CAN	Transceptor CAN
Tarefas	Gerencia a comunicação, gera frames, filtra mensagens.	Converte sinais elétricos, isola a rede.
Interface	SPI	UART

3.2 Módulo LoRa

Os módulos Ra-01, retratado na Figura 10, possuem o *chip* de rádio SX1268, que utiliza o modem remoto LoRa para comunicação de espectro espalhado de longa distância. Ele apresenta forte capacidade anti-interferência e baixo consumo de corrente. Com a tecnologia de modulação LoRa patenteada pela SEMTECH, o SX1268 oferece alta sensibilidade (superior a -148dBm), potência de saída de +22dBm, longa distância de transmissão e alta confiabilidade. Suas áreas de aplicação incluem leitura automática de medidores, automação residencial, sistemas de segurança e sistemas de irrigação remota(Ai-Thinker Technology Co., Ltd., 2024).

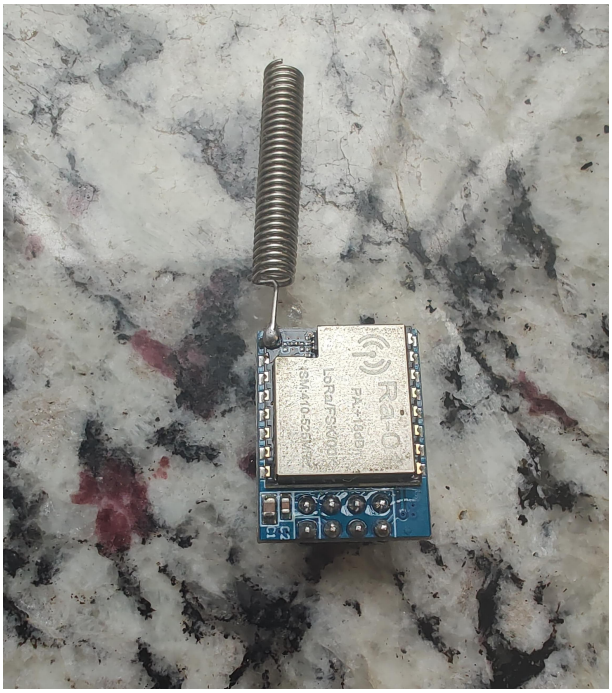

Figura 10 – Periférico Ra-01.

3.3 ESP32

A *ESP32* é uma plataforma de microcontrolador de baixo custo, desenvolvida pela *Espressif Systems*. Este dispositivo é equipado com conectividade sem fio, como Wi-Fi e *Bluetooth*, tornando-o uma opção popular para aplicações em sistemas embarcados. A ela integra diversas funcionalidades em um único *chip*, incluindo múltiplos núcleos de processamento, memória RAM e flash, além de diversos periféricos para interfaces de comunicação. Entre eles, destaca-se o suporte a *Controller Area Network* (*CAN*) através de um controlador embarcado. A ISO11898 que define o protocolo *CAN* é nomeado de *Two-Wire Automotive Interface* (TWAI), pois o nome *CAN* é propriedade intelectual da alemã Bosch (ESPRESSIF-SYSTEMS, 2024).

3.4 Transmissor e Receptor

O transmissor (TX), esquematizado na Figura 11, funciona da seguinte maneira: quando uma mensagem CAN chega no barramento, ocorre uma interrupção de hardware que chama uma função responsável por fazer a leitura da mensagem do barramento CAN. Essa função, chamada pela ISR (*Interruption Service Routine*), acessa o barramento CAN e realiza a leitura do frame de dados.

Uma vez que o frame foi lido, a informação é colocada em uma fila do *FreeRTOS*, que irá armazenar os dados de forma segura. Para garantir que as tarefas que precisam processar essa mensagem CAN não tentem acessar a fila ao mesmo tempo, foram utilizados

semáforos. Quando a interrupção é terminada esse dado é lido da fila por uma *task* responsável por efetuar a transmissão LoRa, a Figura 12 retrata esses processo.

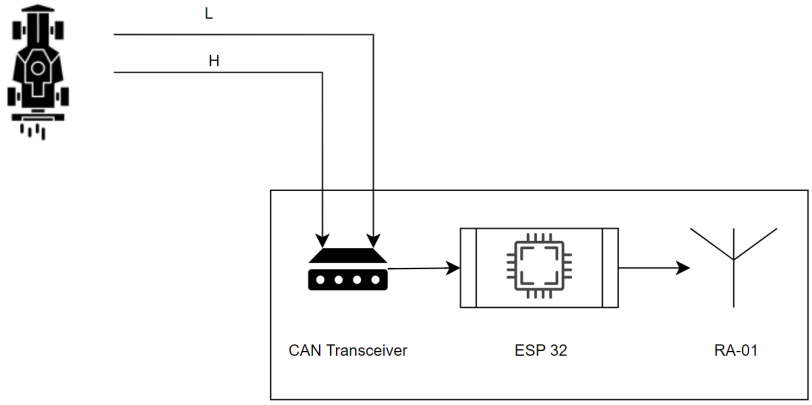

Figura 11 – Diagrama do componente Transmissor (TX).

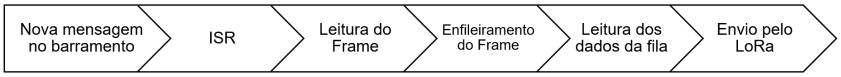

Figura 12 – Diagrama da lógica de programação implementada no transmissor.

Já o dispositivo receptor, tem como responsabilidade receber os dados vindos do transmissor e enviar via serial para o computador, representado pela Figura 13. O dado chega ao receptor por meio do protocolo LoRa, então ele é transmitido via serial para o computador. Esse protocolo foi escolhido pois além de sua facil implementação, ele já se encarrega de fornecer energia a ESP32, essa lógica é ilustrada na Figura 14 . Quando o dado é recebido no computador é verificado em qual tabela a informação será armazenada, por meio do ID do frame da CAN, depois possíveis processamentos podem ser feitos, como mudança de tipos ou operações matemáticas. Essa informação então é salva em um banco de dados e então exibida para o usuário.

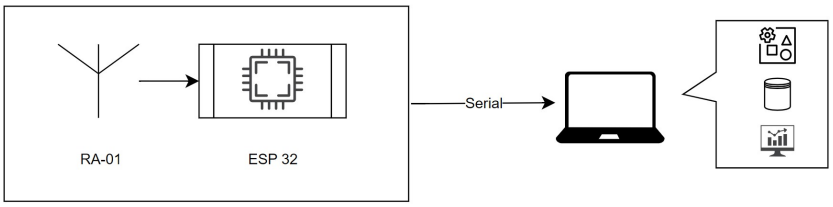

Figura 13 – Diagrama do componente Receptor (RX)

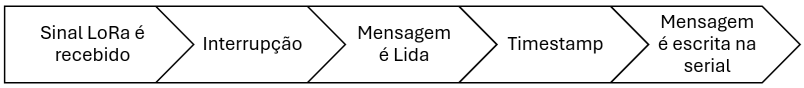

Figura 14 – Diagrama da lógica de programação implementada no receptor.

O protocolo LoRa foi escolhido para este projeto devido à sua capacidade de comunicação de longo alcance, essencial para superar a distância entre o veículo (transmissor) e o notebook (receptor), que pode ultrapassar 200 metros em linha reta. Esse requisito torna inviável o uso de protocolos de comunicação de curto alcance, como Bluetooth e Wi-Fi, embora estejam disponíveis no chip ESP32.

Para validar o desempenho do módulo LoRa e sua adequação ao projeto, foram realizados testes de alcance em campo aberto com obstáculos no Campus I da USP, em São Carlos. Posicionou-se o transmissor (TX) na oficina da equipe EESC-USP Tupã, enquanto o receptor (RX) foi deslocado por várias regiões do campus para medir o alcance e a qualidade do sinal. Esses testes avaliaram o índice de SNR (Signal-to-Noise Ratio) e RSSI (Received Signal Strength Indication) em diferentes distâncias e condições, assegurando que o módulo LoRa atendesse aos requisitos de comunicação do sistema.

Um *script* em Python gerencia a recepção de dados do dispositivo receptor, além de estabelecer a comunicação serial com a placa de desenvolvimento ESP32. Para isso, foram utilizados os pacotes pyserial-asyncio e serial. O pacote pyserial-asyncio permite uma recepção de dados assíncrona, essencial para processar as informações em tempo real sem bloquear outras operações do sistema. Já o pacote serial é utilizado para listar os dispositivos conectados e estabelecer a conexão inicial com o ESP32, garantindo a comunicação serial necessária para o projeto.

3.5 Dashboard

O painel de exibição de dados, também denominado *dashboard*, integra e visualiza as informações armazenadas no banco de dados. Este painel compreende diversos elementos para o monitoramento em tempo real do veículo, incluindo um mapa que exibe a trajetória e a velocidade em forma de um mapa de calor, onde áreas mais quentes indicam velocidades mais altas. Além disso, o painel apresenta gráficos de barras que demonstram a evolução da velocidade e aceleração do veículo ao longo do tempo, assim como um gráfico específico para a temperatura dos pneus e do motor. Em uma seção adicional da interface, são exibidas os estados dos sistemas de segurança do automóvel, permitindo a rápida identificação de eventuais alertas. Por fim, um relatório exibe todas as entradas registradas no sistema, facilitando a rastreabilidade das atividades.

O mapa foi desenvolvido utilizando o pacote Folium, que possibilita a vinculação de dados diretamente ao mapa, suportando visualizações ricas em HTML, como marcadores

personalizados (FOLIUM, 2024). Já os gráficos interativos foram implementados com Plotly Express, uma biblioteca gratuita e de código aberto para Python que permite gerar gráficos dinâmicos e visualmente detalhados (PLOTLY, 2024). Adicionalmente, o painel possui uma seção de controle lateral, na qual é possível ajustar o intervalo de atualização dos dados, alternar entre o modo de recepção ativa ou passiva, além de configurar a atualização automática da página para garantir que as informações exibidas estejam sempre sincronizadas com os dados mais recentes coletados pelo sistema.

3.6 Comparação de Custo

Para realizar a comparação de custo entre o equipamento desenvolvido e seus equivalentes disponíveis no mercado, algumas premissas foram adotadas. A primeira delas é que a mão de obra para desenvolver um equipamento deste tipo em uma equipe como a EESC-USP Tupã possui custo zero, pois os membros são voluntários. A segunda premissa é que o equipamento em questão não será produzido em grandes quantidades ou em atacado, por isso, foi considerado o custo de uma única unidade. Os valores de frete e tributações não foram considerados, tanto para o equipamento descrito nesta monografia quanto para os equipamentos disponíveis no mercado, devido à volatilidade dessas taxas, assim como a variação do valor do dólar e do frete. Dessa forma, a comparação feita aqui está mais focada em avaliar se é financeiramente vantajoso comprar o equipamento ou realizar sua fabricação própria, considerando o contexto da atividade extracurricular.

Os componentes utilizados para a criação do *logger* CAN e suas respectivas quantidades e preços foram organizados na Tabela 6. Todos os valores estão em reais, sem considerar frete, visto que este valor pode variar dependendo da região do fornecedor e do destino da entrega, o valor foi cotado no site AliExpress (ALIEXPRESS, 2023). Também foram considerados os preços de varejo dado que o objetivo do projeto é a construção de um dispositivo único, sem previsão de produção em escala.

Tabela 6 – Custo dos componentes presentes no projeto

Componente	**Quantidade**	**Preço (R$)**
ESP32	2	34,99
SN65HVD230	1	7,66
RA-01	2	22,58

Para avaliar a viabilidade do projeto, também foi feito um estudo dos preços de produtos análogos presentes no mercado esses valores estão presentes na Tabela 7. Como o mercado nacional carece de alternativas foi feito a pesquisa em lojas internacionais (CONNECT, 2023).

Tabela 7 – preços de *Loggers* comerciais

Produto	**Preço (USD$)**
CANedge1	349,95
CL1000	208,95
CL2000	279,95

Ao converter os valores dos produtos comerciais para reais, com uma taxa de câmbio de aproximadamente 5,00, o custo desses dispositivos varia de R$ 1399,75 a R$ 1.749,75 ou seja um valor médio de R$1399,75.

3.7 Documentação de Código

A documentação de código é uma prática essencial em projetos de engenharia, pois permite a preservação e disseminação de conhecimento sobre a estrutura e funcionalidades de um sistema. Em um ambiente de desenvolvimento de software, especialmente quando se trata de projetos colaborativos ou de longo prazo, a documentação detalhada facilita a compreensão, manutenção e expansão do código por membros da equipe, bem como por futuros desenvolvedores que possam trabalhar no projeto. Além disso, ela contribui para a gestão de conhecimento, permitindo que os conceitos e decisões de projeto estejam acessíveis de forma clara e concisa, reduzindo a dependência do conhecimento tácito e assegurando que mudanças no time ou nas demandas do projeto não comprometam o desenvolvimento e evolução do sistema.

Para alcançar uma documentação de qualidade e padronizada, utilizamos o Sphinx(SPHINX, 2023), uma ferramenta popular para documentação de *software*. O Sphinx é amplamente utilizada por sua capacidade de gerar documentação bem estruturada e visualmente amigável, oferecendo suporte para múltiplos formatos de saída, incluindo HTML e PDF, o que permite uma distribuição acessível. No contexto do projeto, a escolha pelo Sphinx se deu pela sua flexibilidade e integração com repositórios de código, o que permite manter a documentação sempre atualizada em sincronia com o código base. Para garantir que a documentação esteja constantemente atualizada, implementamos uma pipeline de integração contínua utilizando GitHub *Actions*, de modo que o Sphinx é executado automaticamente a cada novo *commit* na *branch* principal do projeto gerando a documentação mais recente. Esse processo automatizado minimiza erros manuais e assegura que as atualizações de código estejam refletidas na documentação, aumentando a eficiência e confiabilidade da documentação.

4 RESULTADOS E DISCUSSÃO

Nesta seção será analisado a eficácia do leitor de barramento CAN desenvolvido, destacando-se a eficiência da comunicação via protocolo LoRa e a qualidade dos dados exibidos e do funcionamento no *dashboard*. Também será discutido a viabilidade econômica do projeto, comparando com soluções comerciais disponíveis. Discute-se ainda as limitações enfrentadas durante o desenvolvimento. Por fim, são abordadas sugestões para futuras melhorias, incluindo a integração de novas funcionalidades e o aprimoramento da interface do usuário.

4.1 Transmissor e Receptor

Por meio do uso da ISR e uma fila foi possível fazer a leitura e a transmissão dos dados da CAN de modo apropriado, o que demonstra que o dispositivo cumpre com a sua principal utilidade. Nos testes realizados não houve estouro do *buffer* da fila. O que indica que a taxa de dados que entram e que saem estão em equilíbrio. A escolha do periférico SN65HVD230 como transceptor de CAN também se mostrou uma boa escolha pois utiliza menos pinos que outros modelos como o MCP2515 o qual foi utilizado para simular um nodo do veículo, que se comunica via SPI.

Uma limitação que o projeto pode enfrentar seria o aumento considerável da frequência do fluxo de dados no barramento CAN, isso ocasiona sobrecarga da fila, cenário onde a quantidade de entradas é maior que o de saída, isso implicaria em perda de dados, ou erros de execução. Outro fator limitante é a taxa de envio de dados do periférico RA01 que também pode ocasionar sobrecarga da pilha, mas agora devido a baixa taxa de consumo dos dados. Para contornar ambos os problemas poderia ocorrer um aumento da memória alocada para esta estrutura de dados.

4.2 LoRa

Os resultados dos testes feitos para medir o alcance da antena do protocolo LoRa podem ser vistos na Figura 16 e 17. Já a dimensão da pista de corrida está presente na Figura 15 a distância calculada no mapa é entre a posição ende ficaria o receptor e a maior distância possível que o veículo pode ter deste ponto enquanto ele estiver percorrendo a pista de corrida.

Por meio dessas análises é possível perceber que a antena utilizada seria capaz de cobrir a pista de provas pois ela atingiu o alcance necessário mesmo em um cenário com obstáculos de prédios e arvores, algo que não ocorre no contexto real. Na Tabela 8 temos o valor médio em decibéis do SNR e RSSI, além da distância em metros do emissor e receptor o qual está indicado nas Figuras 16 e 17 por um marcador azul. Por meio da Tabela 8 temos os valores tanto de SNR e RSSI obtidos no Campus I da USP de São Carlos, durante a

coleta dos dados houve perdas de pacotes próximos a II e a VI, provavelmente por causa da distância e grande número de obstáculos presentes como vegetação e construções, por esses motivos esses locais são considerados os limites de alcance da antena. Quando houve um caminho mais limpo entre a antena emissora e a receptora o alcance foi de 354m como indica o ponto I, ana Figura 16.

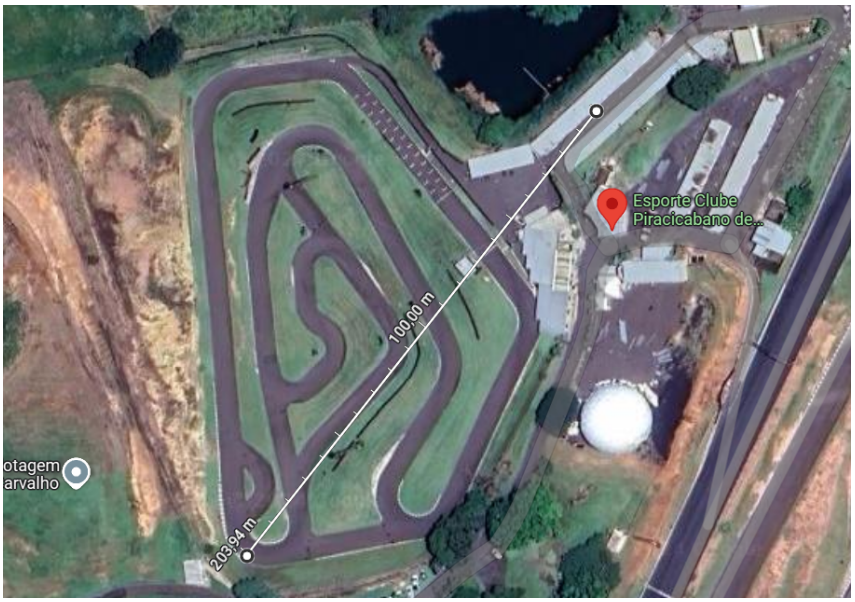

Figura 15 – Vista de satélite da pista de corrida.

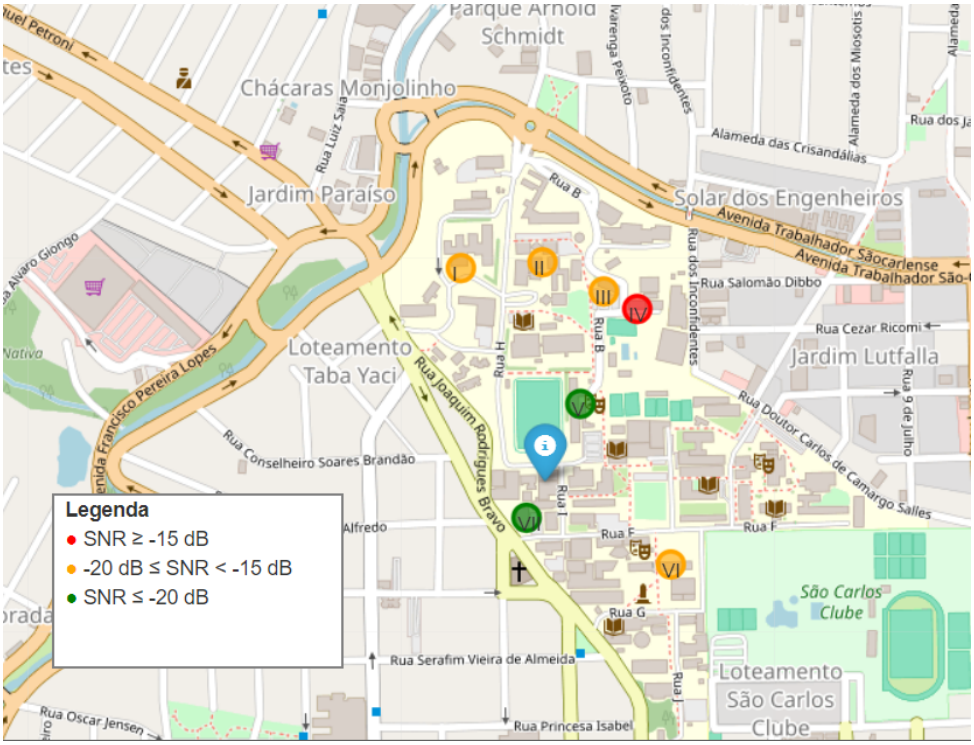

Figura 16 – Mapa com os valores médios de SNR obtidos por meio de um teste realizado no Campus I da USP.

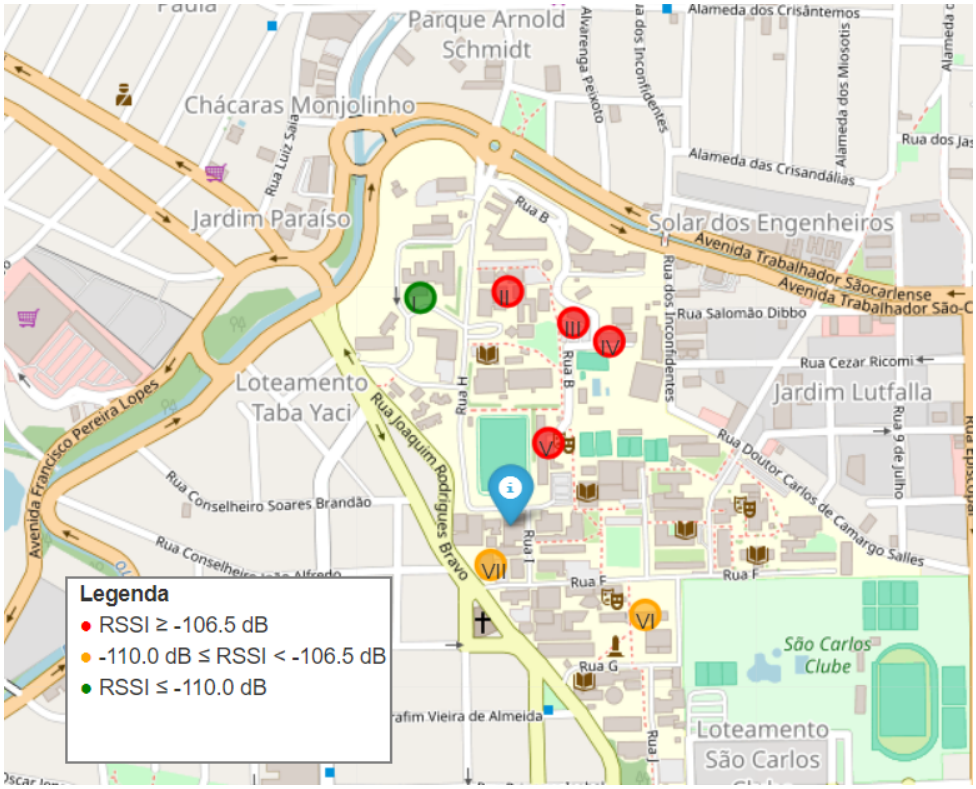

Figura 17 – Mapa com os valores de RSSI obtidos por meio de um teste realizado no Campus I da USP.

Tabela 8 – Média de SNR, RSSI e Distância do Emissor nos Locais Monitorados

Local	**Mapa**	**SNR (dB)**	**RSSI (dB)**	**Distância (m)**
Departamento de Engenharia de Estruturas (SET)	I	-16,1	-106,5	354
Prefeitura do Campus	III	-16,9	-114,8	330
Bloco D	II	-19,1	-115,0	320
Salão de Eventos - Campus da USP São Carlos	IV	-14,3	-115,9	263
E1	VI	-19,3	-108,2	235
Departamento de Engenharia Elétrica e de Computação (SEL)	V	-27,4	-113,7	117
Laboratórios do SEL	VII	-28,6	-109,9	83

4.3 *Dashboard*

A interface de exibição, ilustrada na Figura 18 e 19, consegue ler os dados e apresentá-los sem interrupções ou travamentos graças ao uso de técnicas de entrada e saída assíncronas. A adoção dos *session states* do Streamlit possibilitou a atualização dinâmica da página sem a perda de informações fornecidas pelo usuário ou dados críticos do sistema, como a conexão com o banco de dados. Além disso, a combinação da comunicação assíncrona com

o leitor serial e a integração eficiente com o banco de dados proporciona uma experiência fluida ao usuário, permitindo uma taxa de atualização adequada. Essa abordagem evita desconfortos visuais, como a renderização completa e perceptível da interface durante as atualizações, garantindo uma interação contínua e intuitiva.

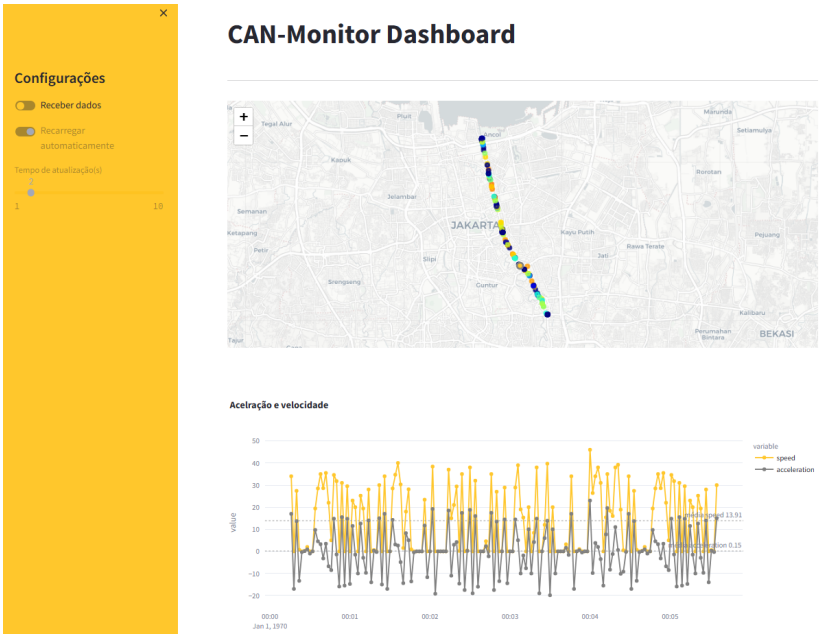

Figura 18 – Mapa e gráfico do painel desenvolvido.

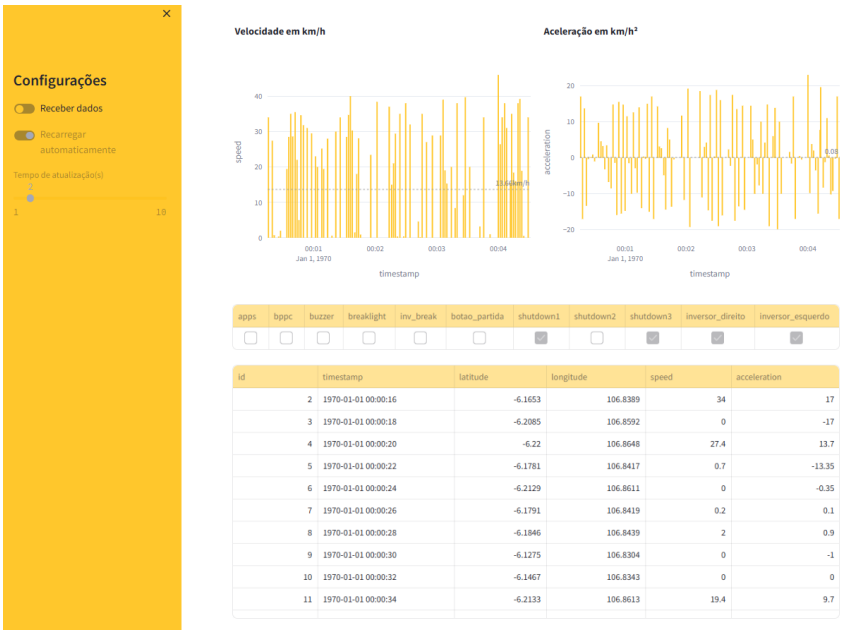

Figura 19 – Estados do veículo e registro do painel.

4.4 Avaliação de Custo

A comparação de custos presentes nas Tabelas 6 e 7 evidencia que o projeto oferece uma solução economicamente vantajosa. Mesmo sem considerar produção em escala, o valor final é significativamente menor que o dos produtos comerciais, atendendo ao objetivo de viabilidade e baixo custo , presente na Secção 1.1.1. Essa economia permite que o dispositivo seja uma alternativa para o projeto da EESC-USP Tupã, sem a necessidade de um grande investimento inicial. Validando o objetivo inicial de baixo custo e viabilidade para as aplicações da equipe.

4.5 Documentação

A geração de documentação automática alcançou plenamente seu principal objetivo. Ela permitiu documentar de maneira clara e detalhada as APIs do projeto, garantindo que informações técnicas relevantes estejam acessíveis a qualquer pessoa interessada por meio do GitHub Pages. A Figura 20 ilustra a página principal da documentação da dashboard. Um dos principais benefícios dessa abordagem é a sincronização contínua entre o código e a documentação, eliminando defasagens que poderiam comprometer a confiabilidade e a utilidade do material técnico. Dessa forma, assegura-se que as informações disponíveis sejam sempre precisas e atualizadas, refletindo a realidade do sistema.

Além disso, a documentação desempenha um papel crucial na preservação e disseminação de conhecimento sobre a estrutura e as funcionalidades do sistema. Ela facilita a entrada de novos colaboradores no projeto, reduzindo o tempo necessário para compreender o funcionamento do código. Também é um recurso valioso para manutenção futura, permitindo que alterações ou expansões sejam realizadas de maneira segura e eficiente, com uma visão clara do impacto potencial no sistema como um todo.

O uso de ferramentas como Git e GitHub potencializa esses benefícios, promovendo um ambiente colaborativo e organizado para o desenvolvimento de software. O controle de versão fornecido pelo Git permite rastrear mudanças no código e na documentação, garantindo um histórico completo das evoluções do projeto. Isso é essencial para identificar regressões, reverter alterações problemáticas e manter a integridade do sistema.

Por sua vez, o GitHub amplia esses recursos ao oferecer uma plataforma centralizada para colaboração e compartilhamento. Funcionalidades como *pull requests*, revisão de código e *issues* promovem um fluxo de trabalho estruturado, enquanto a integração com GitHub Pages viabiliza a publicação de documentação diretamente a partir do repositório. Dessa forma, a combinação de documentação automática com ferramentas de controle de versão e hospedagem pública não só melhora a eficiência do desenvolvimento, mas também garante a transparência e acessibilidade do projeto para a comunidade.

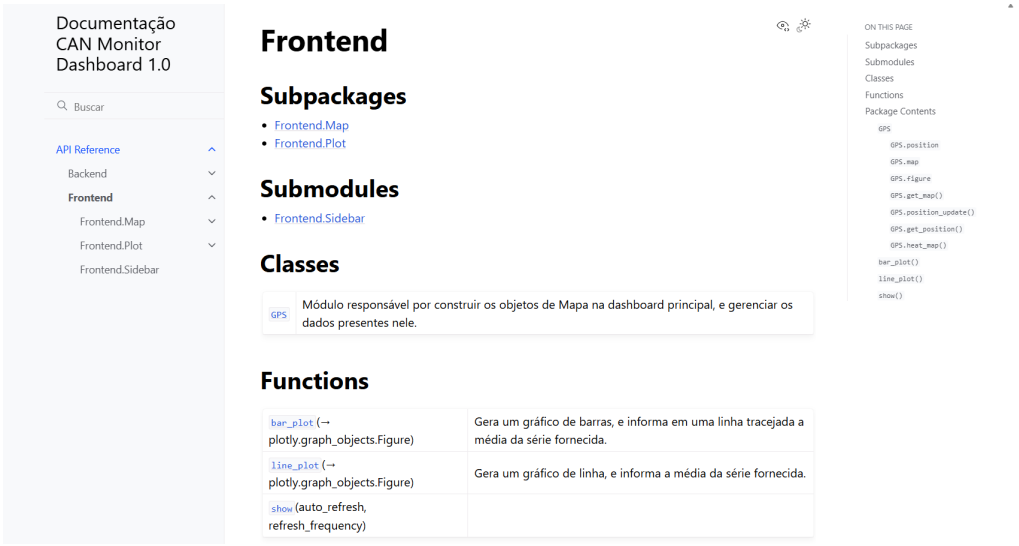

Figura 20 – Captura de tela da documentação do projeto.

5 CONCLUSÃO

O equipamento atendeu aos requisitos apresentados na Secção 1.1.1, como baixo custo e a capacidade de se comunicar com o veículo enquanto este realiza provas na pista do ECPA ou testes prévios a competição, exigindo que o alcance da antena superasse o diâmetro da pista de corrida que é de aproximadamente 200 m.

Em relação à aquisição de dados, o projeto foi bem-sucedido em atender às demandas especificadas. Esse resultado foi alcançado devido à utilização do *FreeRTOS* e das suas estruturas, como as rotinas de serviço de interrupção (*ISR*) para lidar com a chegada de novos *frames*, as filas para armazenamento temporário das mensagens CAN até que sejam processadas e o agendador para gerenciar as tarefas de forma eficiente. Caso a frequência de leitura dos dados aumente significativamente, a ponto de causar perdas, é possível mitigar o problema ampliando a capacidade da fila, garantindo maior robustez ao sistema.

Quanto à transmissão de dados, foi possível realizá-la a uma distância considerável, próxima ao tamanho da pista de testes, que representa o limite superior para o uso do protocolo LoRa. Em testes locais, como os realizados no campus universitário, o alcance das antenas foi muito superior ao necessário.

A exibição dos dados no *dashboard* também atendeu às expectativas, uma vez que a entrada assíncrona garante maior fluidez na visualização das informações. Além disso, a API de exibição permite uma interface interativa, possibilitando navegação, ampliação de detalhes e exibição de valores médios. Em relação aos custos, o valor despendido para o desenvolvimento da aplicação foi consideravelmente inferior ao de produtos comerciais. Além de oferecer capacidade de personalizações para os objetivos da equipe.

Portanto, este trabalho cumpriu com sucesso as demandas estabelecidas, atingindo seus objetivos tanto em termos de pesquisa quanto de engenharia, e demonstra potencial para servir como base para projetos futuros.

REFERÊNCIAS

ABDERRAHMANE, T.; NOURREDINE, A.; MOHAMMED, T. Experimental analysis for comparison of wireless transmission technologies: Wi-fi, bluetooth, zigbee and lora for mobile multi-robot in hostile sites. *International Journal of Electrical & Computer Engineering (2088-8708)*, v. 14, n. 3, 2024.

Ai-Thinker Technology Co., Ltd. *Ra-01/Ra-02 LoRa Module User Manual*. [S.l.], 2024. Disponível em: https://docs.ai-thinker.com/en/lora/man.

ALIEXPRESS. *AliExpress: Affordable Prices on Top Brands with Free Shipping*. 2023. Accessed: 2024-04-06. Disponível em: https://www.aliexpress.com/.

ANH, T. N. B.; TAN, S.-L. Real-time operating systems for small microcontrollers. *IEEE Micro*, v. 29, n. 5, p. 30–45, 2009.

AWAN, A. A. *Discover SQLAlchemy: A Beginner Tutorial With Examples*. 2024. https://www.datacamp.com/tutorial/sqlalchemy-tutorial-examples.

BARRY, R. *Mastering the FreeRTOS™ Real Time Kernel*. Real Time Engineers Ltd, 2016. Disponível em: https://www.freertos.org/media/2018/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf.

CHEN, Y.-Y. et al. An embedded system for vehicle surrounding monitoring. In: *2009 2nd International Conference on Power Electronics and Intelligent Transportation System (PEITS)*. [S.l.: s.n.], 2009. v. 2, p. 92–95.

CONNECT, G. *Grid Connect: Networking and Embedded Products*. 2023. Accessed: 2024-04-06. Disponível em: https://www.gridconnect.com/.

CORRIGAN, S. *Introduction to the Controller Area Network (CAN)*. [S.l.], 2002. Revised May 2016. Disponível em: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

CSS-ELECTRONICS. *CAN bus - the ultimate guide*. [S.l.], 2023. Disponível em: https://www.csselectronics.com/pages/can-bus-ultimate-guide.

CSS-ELECTRONICS. *Telematics Dashboard - Open Source*. [S.l.], 2024.

DHAKER, P. *Introduction to SPI Interface | Analog Devices*. 2018. Publisher: ANALOG DEVICES. Disponível em: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html.

ECPA. *ECPA - Serviços*. 2024. http://www.ecpa.com.br/capa.asp?s=servico&idservico=1066&confirma=1.

EESC. *EESC-USP Tupã conquista 3º lugar na competição Fórmula SAE Brasil*. 2024. https://saocarlos.usp.br/81734-2/.

ELECTROMATE. *A Guide to Controller Area Network (CAN) for Industrial Applications*. 2024. https://www.electromate.com/news/post/a-guide-to-controller-area-network-can-for-industrial-applications.

ESPRESSIF. *CAN Driver - API Reference*. 2024. https://docs.espressif.com/projects/esp-idf/en/release-v3.3/api-reference/peripherals/can.html.

ESPRESSIF-SYSTEMS. *ESP-IDF Programming Guide*. 2024. Disponível em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/esp-idf-en-v5.3.2-esp32.pdf.

FOLIUM. *Folium Documentation*. 2024. https://python-visualization.github.io/folium/latest/.

FreeRTOS. *FreeRTOS™ - FreeRTOS™*. 2024. Disponível em: https://freertos.org.

IEEE. Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, p. 1–40, 1998.

INTEL. *How Does Bluetooth® Technology Work?* 2024. https://www.intel.com/content/www/us/en/products/docs/wireless/how-does-bluetooth-work.html#:~:text=Bluetooth%C2%AE%20short%2Drange%20wireless,cables%20or%20supporting%20network%20infrastructure.

KATARE, D.; EL-SHARKAWY, M. Embedded system enabled vehicle collision detection: An ann classifier. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.: s.n.], 2019. p. 0284–0289.

LAPLANTE, P. A. *Real-Time Systems Design and Analysis: Tools for the Practitioner*. 4 th. ed. [S.l.]: Wiley-IEEE Press, 2011.

LI, Z.; WANG, D.; KANG, Q. [retracted] the development of data acquisition system of formula sae race car based on can bus communication interface and closed-loop design of racing car. *Wireless Communications and Mobile Computing*, v. 2021, n. 1, p. 4211010, 2021. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/4211010.

MHADHBI, N. *Python tutorial: Streamlit*. DataCamp, 2021. Disponível em: https://www.datacamp.com/tutorial/streamlit.

MICROCHIP. *MCP2515*. 2019. Accessed: 2024-04-06. Disponível em: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf.

MUSTAFA, D.; KHABOUR, S. M.; MUSTAFEH, I. G. Enhancing real-time embedded system education with self-driving car models. *Human Behavior and Emerging Technologies*, v. 2024, n. 1, p. 8578058, 2024. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1155/2024/8578058.

OBAMA, B. *A Promised Land*. Crown, 2020. ISBN 9781524763183. Disponível em: https://books.google.com.br/books?id=hvr4DwAAQBAJ.

PHYTOOLS. *CAN Data Logger*. 2024. https://phytools.com/collections/can-data-logger.

PLOTLY. *Plotly Express*. 2024. https://plotly.com/python/plotly-express/.

PYTHON-SOFTWARE-FOUNDATION. *asyncio: Asynchronous I/O framework*. 2023. https://docs.python.org/3/library/asyncio.html. Accessed: 2024-09-28.

RIDHA, R. *Transjakarta Bus GPS Data*. 2019. Disponível em: https://www.kaggle.com/datasets/rasyidstat/transjakarta-bus-gps-data.

SAE-BRASIL. *Formula SAE Brasil: Programa Estudantil*. 2024. https://saebrasil.org.br/programas-estudantis/formula-sae-brasil/.

Segger. *embOS - RTOS, Real-Time Operating System | SEGGER*. 2023. Disponível em: https://www.segger.com/products/rtos/embos/.

Semtech Corporation. *LoRa Modulation Basics*. [S.l.], 2015. Disponível em: https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf.

SIEBENEICHER, H. Documentation, *Universal Asynchronous Receiver-Transmitter (UART)*. 2024. Publisher: Arduino C.C. Disponível em: https://docs.arduino.cc/learn/communication/uart/.

SIEWERT, S. *Terminology Guide*. 2016. Disponível em: https://experts.colorado.edu/display/coursename_ECEN-5623.

SOUZA, F. *O que são sistemas embarcados?* 2022. Disponível em: https://embarcados.com.br/o-que-sao-sistemas-embarcados/.

SPHINX. *Sphinx Documentation*. [S.l.], 2023. Accessed: 2024-04-06. Disponível em: https://www.sphinx-doc.org.

TANENBAUM, A. S.; BOS, H. *Modern operating systems*. Fifth edition. rental edition. Hoboken, NJ: Pearson, 2022. OCLC: 1442793663. ISBN 9780137618873.

The Things Network. *RSSI and SNR*. [S.l.], 2024. Disponível em: https://www.thethingsnetwork.org/docs/lorawan/rssi-and-snr/.

TI. *SN65HVD23x 3.3-V CAN Bus Transceivers*. 2018. Accessed: 2024-04-06. Disponível em: https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf?ts=1733504293169&ref_url=https%253A%252F%252Fwww.google.com%252F.

TOTVS. *Telemetria Veicular: O que é, como funciona e por que utilizar?* 2024. https://www.totvs.com/blog/gestao-de-servicos/telemetria-veicular/.

UEL. *Sistemas Embarcados*. 2024. https://www.uel.br/pos/ese/?page_id=27.

UFRJ. *Protocolo Zigbee*. 2017. Publisher: UFRJ. Disponível em: https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2017_2/802154/zigbee.html.

WaTech. *WiFi definition and meaning | WaTech*. 2024. Disponível em: https://watech.wa.gov/wifi-definition-and-meaning.

Zephyr. *About the Zephyr Project*. 2023. Disponível em: https://zephyrproject.org/learn-about/.