UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA
DE CONTROLE E AUTOMAÇÃO

Antonio Vitor da Silva

Desenvolvimento de Sistema Embarcado para Telemetria Veicular

Recife
2025

Antonio Vitor da Silva

Desenvolvimento de Sistema Embarcado para Telemetria Veicular

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel.

Orientador(a): Prof. Dr. Marcio Evaristo da Cruz Brito

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Antonio Vitor da.
Desenvolvimento de Sistema Embarcado para Telemetria Veicular / Antonio Vitor da Silva. - Recife, 2025.
60p : il., tab.

Orientador(a): Marcio Evaristo da Cruz Brito
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2025.
Inclui referências.

1. Sistemas Embarcados. 2. Telemetria Veicular. 3. Internet das Coisas. 4. Protocolo CAN. 5. OBD-II. I. Cruz Brito, Marcio Evaristo da. (Orientação). II. Título.

620 CDD (22.ed.)

Antonio Vitor da Silva

Desenvolvimento de Sistema Embarcado para Telemetria Veicular

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel.

Aprovado em: 18/12/2025.

BANCA EXAMINADORA

_____

Prof. Dr. Marcio Evaristo da Cruz Brito (Orientador)
Universidade Federal de Pernambuco

_____

Prof. Dr. Davidson da Costa Marques (Examinador Interno)
Universidade Federal de Pernambuco

_____

Eng. M.Sc. Renato Andrade Freitas (Examinador Interno)
Universidade Federal de Pernambuco

AGRADECIMENTOS

Dedico este trabalho a todos colegas, amigos, professores e familiares que contribuíram para a minha formação pessoal e acadêmica ao longo desta longa trajetória. Agradeço à Equipe Mandacaru AeroDesign por ter revelado a um jovem inocente a magia da engenharia e a satisfação de transformar projetos em realidade, além de me ensinar os verdadeiros significados de perseverança e companheirismo. Igualmente, direciono meus agradecimentos à Equipe Mangue Baja, que me incentivou a buscar a excelência e me mostrou que somos sempre capazes de ultrapassar nossos próprios limites. Agradeço, ainda, ao LIVE por me abrir as portas a um mundo repleto de novas possibilidades. Por fim, mas não menos importante, manifesto toda a minha gratidão aos meus pais, pelo amor e apoio incondicional, sem os quais essa jornada não poderia ser trilhada.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema embarcado modular e de baixo custo para aquisição, processamento e transmissão de dados veiculares. A solução integra os protocolos CAN (via OBD-II), Bluetooth Low Energy e MQTT, permitindo coletar parâmetros do veículo, normalizá-los e publicá-los em redes locais ou remotas. A arquitetura é organizada em módulos de interface (OBD-II/CAN), processamento e comunicação, adotando formatos de dados estruturados que favorecem a interoperabilidade com diferentes plataformas. Os resultados demonstram a viabilidade da solução para aplicações em telemetria veicular, com capacidades de expansão para novos sensores e protocolos. O sistema prioriza acessibilidade, utilizando componentes de baixo custo e formatos de dados estruturados, garantindo interoperabilidade com diferentes plataformas. A proposta oferece uma base sólida para monitoramento veicular em tempo real, com potencial para uso em gestão de frotas, diagnóstico remoto e pesquisas em mobilidade inteligente.

Palavras-chave: Sistemas Embarcados; Telemetria Veicular; Internet das Coisas (IoT); Protocolo CAN; OBD-II; Bluetooth Low Energy; MQTT.

ABSTRACT

This work presents the development of a modular, low-cost embedded system for vehicle data acquisition, processing, and transmitting. The solution integrates CAN (via OBD-II), Bluetooth Low Energy, and MQTT protocols, enabling the collection, normalization, and published over local or remote networks. The architecture is organized into interface (OBD-II/CAN), processing, and communication modules, adopting structured data formats that favor interoperability with different platforms. The results demonstrate the system's viability for vehicle telemetry applications, offering extensibility for new sensors and protocols. The system prioritizes accessibility, using low-cost components and structured data formats, ensuring interoperability with different platforms. The proposal offers a solid foundation for real-time vehicle monitoring, with potential for use in fleet management, remote diagnostics, and smart mobility research.

Keywords: Embedded Systems, Vehicle telemetry; Internet of Things (IoT); CAN protocol; OBD-II; Bluetooth Low Energy; MQTT.

LISTA DE ILUSTRAÇÕES

Figura 1 - Barramento CAN 18
Figura 2 - Níveis de Sinal diferenciais CAN 18
Figura 3 - Conector OBD-II 20
Figura 4 - Formato de mensagem OBD 21
Figura 5 - Arquitetura BLE 25
Figura 6 - Arquitetura MQTT 26
Figura 7 - Ciclo de Projeto em V 28
Figura 8 - Arquitetura de Hardware 29
Figura 9 - Arquitetura de Comunicação 30
Figura 10 - Arquitetura de Firmware 38
Figura 11 - Arquitetura Física de Hardware 43
Figura 12 - Parte Superior da Placa Base 44
Figura 13 - Parte Inferior da Placa Base 44
Figura 14 - Placa Modular 44
Figura 15 - Montagem das Placas 45
Figura 16 - ESP32 DEV KIT V1 46
Figura 17 - Módulo MCP2515 47
Figura 18 - Módulo Sim800L 47
Figura 19 - Conversor Buck LM2596 48
Figura 20 - Arquitetura de Montagem para Teste de Hardware 48
Figura 21 - Montagem Inicial do Sistema 49
Figura 22 - Solicitação e Resposta de PIDs disponíveis no barramento 50
Figura 23 - Envio e Recebimento de Mensagens no Modo 01 50
Figura 24 - Decodificação do conteúdo das mensagens 51
Figura 25 - Veículos Testados 52
Figura 26 - Recebimento de Mensagem BLE em aparelho celular 53
Figura 27 - Formatação de mensagem JSON com os dados veiculares 54
Figura 28 - Recebimento de Mensagens no Tópico Assinado 54
Figura 29 - Área de testes delimitada 55
Figura 30 - Sistema Instalado na Porta OBD-II 56

LISTA DE TABELAS

Tabela 1 - Tabela de Conversão de PIDs 21
Tabela 2 - Comparação de Tecnologias de Comunicação 31
Tabela 3 - Comparação de Microcontroladores 35

LISTA DE ABREVIATURAS E SIGLAS

2G	Second Generation (Segunda Geração)
BLE	Bluetooth Low Energy (Bluetooth de Baixa Energia)
CAN	Controller Area Network (Rede de Área do Controlador)
CAN ID	Controller Area Network Identifier (Identificador da Rede CAN)
CPU	Central Processing Unit (Unidade Central de Processamento)
CRC	Cyclic Redundancy Check (Verificação de Redundância Cíclica)
ECU	Electronic Control Unit (Unidade de Controle Eletrônico)
EPA	Environmental Protection Agency (Agência de Proteção Ambiental)
GATT	Generic Attribute Profile (Perfil de Atributos Genéricos)
GPRS	General Packet Radio Service (Serviço Geral de Pacotes por Rádio)
GSM	Global System for Mobile Communications (Sistema Global de Comunicações Móveis)
I2C	Inter-Integrated Circuit (Circuito Inter-integrado)
ID	Identification / Identifier (Identificação / Identificador)
IoT	Internet of Things (Internet das Coisas)
JSON	JavaScript Object Notation (Notação de Objetos JavaScript)
LED	Light Emitting Diode (Diodo Emissor de Luz)
LoRaWAN	Long Range Wide Area Network (Rede de Longo Alcance e Baixo Consumo)
MQTT	Message Queuing Telemetry Transport (Transporte de Telemetria de Enfileiramento de Mensagens)
OBD II	On-Board Diagnostics II (Diagnóstico a Bordo II)
PCB/PCI	Printed Circuit Board (Placa de Circuito Impresso)
PID	Parameter ID (Identificador de Parâmetro)
PTH	Plated Through-Hole (Placa com Furos Metalizados)
RTOS	Real-Time Operating System (Sistema Operacional em Tempo Real)
RX	Receive (Recepção / Receptor)
SMD	Surface-Mount Device (Dispositivo de Montagem em Superfície)
SPI	Serial Peripheral Interface (Interface Periférica Serial)
SSL/TLS	Secure Sockets Layer / Transport Layer Security (Camada de Soquetes Seguros / Camada de Segurança de Transporte)

TX	Transmit (Transmissão / Transmissor)
UART	Universal Asynchronous Receiver-Transmitter (Receptor-Transmissor Assíncrono Universal)
VM	Virtual Machine (Máquina Virtual)
Wi-Fi	Wireless Fidelity

LISTA DE SÍMBOLOS

V	Volts
MHz	Mega Hertz
MB	Megabyte
Mbps	Megabits por segundo
GHz	Gigahertz
bps	Bits por segundo
ºC	Graus Celsius
kB	Kilobyte
Kbps	Kilobits por segundo

Sumário

1 INTRODUÇÃO 14

1.1 OBJETIVOS 15

1.1.1 Geral 15

1.1.2 Específicos 16

1.2 ORGANIZAÇÃO DO TRABALHO 16

2 FUNDAMENTAÇÃO TEÓRICA 17

2.1 SISTEMAS AUTOMOTIVOS 17

2.1.1 Controller Area Network (CAN) 17

2.1.2 Arquitetura de Hardware para interfaceamento da rede CAN 18

2.1.3 On-Board Diagnostics II (OBD-II) 19

2.2 SISTEMAS EMBARCADOS 22

2.2.1 Microcontroladores 22

2.2.2 Sistema Operacional em Tempo Real (RTOS) 23

2.3 PROTOCOLOS DE COMUNICAÇÃO E INTERNET DAS COISAS 24

2.3.1 Bluetooth Low Energy 24

2.3.2 MQTT 25

2.4 METODOLOGIA DE PROJETO EM V 26

3 DESENVOLVIMENTO DO TRABALHO 28

3.1 PROJETO DO SISTEMA 29

3.1.1 Escolha das Tecnologias de Comunicação 30

3.1.2 Hospedagem de Serviço em Nuvem 32

3.1.3 Seleção de Componentes de Hardware 33

3.1.4 Ferramentas de Depuração e Monitoramento 36

3.2 PROJETO DETALHADO 37

3.2.1 Projeto de Firmware 37

3.2.2 Projeto de Hardware 42

3.3 IMPLEMENTAÇÃO 45
3.3.1 Codificação 45
3.3.2 Montagem de Hardware 46
3.4 TESTES DE UNIDADE 49
3.5 TESTES DE INTEGRAÇÃO 52
3.6 TESTES DE SISTEMA 56
4 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE 57
5 REFERÊNCIAS 59

1 INTRODUÇÃO

Impulsionado pela digitalização, o setor automotivo passa por uma transformação estrutural. A convergência entre conectividade embarcada, sensores e análise de dados amplia as possibilidades de mobilidade e de gestão de frotas. Nesse contexto, a telemetria veicular torna-se um componente central, visto que, coleta e transmite, em tempo real, indicadores operacionais do veículo, sustentando desde o monitoramento e diagnóstico básicos até aplicações avançadas baseadas em IoT e aprendizado de máquina (GUBBI, 2013).

A importância desses sistemas vai muito além do simples acompanhamento de veículos, eles atuam como a base para a mobilidade inteligente, coletando e transmitindo dados cruciais sobre desempenho do motor, eficiência no consumo de combustível, hábitos de direção e condições mecânicas (ALAM, 2022). Essas informações são valiosas não só para proprietários de veículos, mas também para empresas de transporte, seguradoras, fabricantes e gestores de trânsito urbano (ZHANG, 2019).

A disponibilidade de dados de telemetria abre amplas oportunidades de pesquisa em áreas como segurança viária, eficiência energética e planejamento urbano (LI, 2021). Ao aplicar reconhecimento de padrões e aprendizado de máquina a esses conjuntos de dados, é possível identificar perfis de condução, mapear fatores de risco para acidentes, otimizar rotas e até antecipar necessidades de manutenção preditiva, com impactos diretos em segurança, consumo e tempo de viagem (CHEN, 2020).

A evolução dos sistemas embarcados e da Internet das Coisas para automóveis mostra um progresso constante nas últimas décadas. A relevância prática desses sistemas aparece claramente quando olhamos para seus benefícios em diferentes áreas. Para a segurança no trânsito, a telemetria permite identificar comportamentos de direção perigosos, detectar problemas mecânicos com antecedência e até intervir em tempo real para evitar acidentes (VICTOR, 2022). No aspecto ambiental, o monitoramento preciso do motor e das condições de operação ajuda a melhorar o consumo de combustível e reduzir a poluição.

Economicamente, a telemetria veicular oferece oportunidades reais de reduzir custos operacionais através de manutenção preventiva, otimização de rotas e gestão eficiente de frotas. Do ponto de vista tecnológico, esses sistemas representam um campo cheio de possibilidades para inovação, unindo avanços em computação embarcada, comunicação sem fio, inteligência artificial e análise de dados (CHEN, 2020). Neste cenário, empresas como CSS Electronics oferecem hardware especializado, enquanto soluções como Geotab e Sascar dominam o mercado com plataformas completas. Estes sistemas apresentam custos elevados, arquitetura fechada e limitada flexibilidade para customizações.

Esta lacuna motiva o desenvolvimento de uma arquitetura aberta que mantém a confiabilidade do padrão *On Board Diagnostics* (OBD-II) com custo reduzido, inserindo-se no contexto de inovação e transformação digital do setor através da criação de uma arquitetura integrada para capturar, processar e transmitir dados veiculares. Este trabalho busca demonstrar as possibilidades tecnológicas atuais, contribuindo para o avanço dos sistemas de telemetria com aplicações práticas e acessíveis.

1.1 Objetivos

1.1.1 Geral

O principal objetivo deste trabalho é desenvolver um sistema embarcado de baixo custo, capaz de fornecer dados veiculares estruturados e acessíveis através de múltiplas plataformas e protocolos de comunicação. O sistema deve incorporar requisitos de expansibilidade, permitindo a integração futura de novos sensores, funcionalidades e protocolos de comunicação. Utilizando componentes amplamente disponíveis no mercado e de custo acessível, este trabalho busca desenvolver uma plataforma para acesso e compartilhamento de dados veiculares, tornando viável o uso em diferentes aplicações e segmentos.

1.1.2 Específicos

1. Projetar uma arquitetura de sistema modular e expansível: Definir uma estrutura de tarefas e dispositivos concorrentes, garantindo escalabilidade para futuras ampliações funcionais.
2. Implementar uma camada robusta de aquisição de dados: Desenvolver mecanismos para leitura periódica de parâmetros veiculares via barramento CAN, utilizando a porta OBD-II.
3. Integrar múltiplos protocolos de comunicação: Configurar interfaces diversificadas para transmissão de dados. Garantir interoperabilidade e capacidade de adaptação a diferentes cenários de conectividade.

1.2 Organização do Trabalho

O trabalho está organizado em quatro capítulos principais.

No Capítulo 1, são apresentados o contexto e a motivação para o desenvolvimento do sistema de telemetria veicular, seguidos pelos objetivos gerais e específicos, e por fim uma visão geral da estrutura do trabalho.

O Capítulo 2 apresenta fundamentação teórica, onde revisam-se os conceitos que sustentam o projeto. Abordou-se temas relacionados à sistemas automotivos como o protocolo CAN e o funcionamento do padrão OBD-II, sistemas embarcados como microcontroladores, o sistema operacional de tempo real RTOS, a comunicação Bluetooth Low Energy (BLE), o protocolo MQTT, e a metodologia Ciclo V para projetos.

O Capítulo 3 apresenta o desenvolvimento do Trabalho, onde detalha-se o acompanhamento da metodologia adotada, abordando a estratégia de seleção de componentes e meios de comunicação, o projeto de arquitetura do sistema e fluxos de controle, implementação de ferramentas de codificação e projetos de hardware, além dos testes e resultados atingidos.

Finalmente, o Capítulo 4 apresenta a Conclusão e Propostas de Continuidade, onde avalia-se o grau de atendimento dos objetivos, e apresentam-se sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas Automotivos

A eletrônica veicular moderna é composta por uma rede complexa de *Electronic Control Units* (ECU) responsáveis por gerenciar funções críticas como injeção de combustível, freios ABS, *airbags* e controle de estabilidade. Essas unidades eletrônicas são distribuídas por todo o automóvel, propiciando uma comunicação confiável e em tempo real um requisito fundamental para o funcionamento seguro e eficiente do veículo. Dessa forma, o protocolo CAN surgiu como a solução dominante para este desafio.

2.1.1 Controller Area Network (CAN)

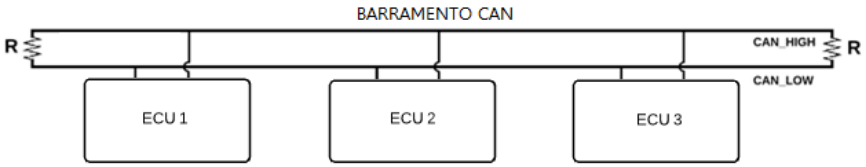
O *Controller Area Network* (CAN) é um protocolo de comunicação serial multimestre, amplamente reconhecido por sua robustez e elevada confiabilidade. Desenvolvido pela Bosch na década de 1980 para atender às demandas do setor automotivo, o CAN surgiu como alternativa ao complexo sistema de conexões ponto a ponto então utilizado. Sua adoção possibilitou a redução significativa do peso, do custo e da complexidade da fiação veicular, ao mesmo tempo em que assegurou uma comunicação eficiente e imune a interferências eletromagnéticas, condições comuns no ambiente automotivo (BOSCH, 1991).

A principal característica do CAN é sua operação baseada em mensagens, onde a prioridade de transmissão é determinada pelo identificador da mensagem (CAN ID), e não por um endereço nodal. Isso significa que todas as ECUs na rede recebem todas as mensagens, cabendo a cada uma decidir, com base no CAN ID, se a mensagem é relevante para suas operações. Este modelo *broadcast* simplifica a adição de novos nós à rede, promovendo escalabilidade e flexibilidade (DAVIS, 2013).

O barramento CAN físico é composto por um par de fios trançados, denominados CAN_H (Can_High) e CAN_L (Can_Low), que operam de forma diferencial. Esta configuração oferece alta imunidade a ruídos eletromagnéticos, pois a interferência afeta ambos os fios igualmente, e o receptor interpreta a diferença de

potencial entre eles. Como apresentado na Figura 1, a topologia de barramento requer a instalação de resistores de terminação (tipicamente 120 Ω) em cada extremo do barramento para evitar reflexões de sinal e garantir a integridade da comunicação, conforme ISO 11898-2 (ISO, 2016).

Figura 1 - Barramento CAN

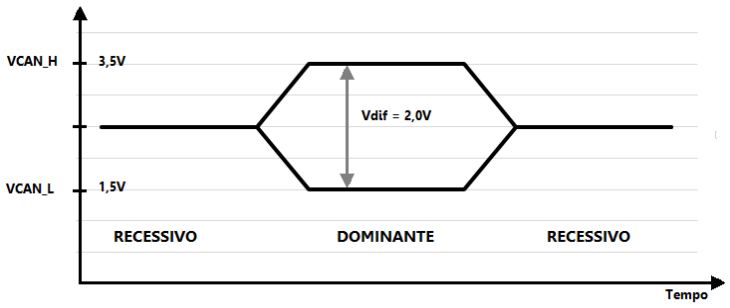


Fonte: (MARINA LACERDA, 2019).

2.1.2 Arquitetura de Hardware para interfaceamento da rede CAN

A implementação prática da comunicação com o barramento CAN veicular requer mais do que apenas o conhecimento do protocolo. É necessária uma arquitetura de hardware que faça a interface entre o microcontrolador, que opera em níveis de tensão de 0V a 3.3V ou 5V, e o barramento CAN diferencial (CAN_H e CAN_L), que opera em níveis de tensão diferentes e em um ambiente eletricamente ruidoso, como mostrado na Figura 2 . Esta arquitetura é composta essencialmente por dois componentes integrados: o Controlador CAN e o Transceptor CAN (ISO, 2016).

Figura 2 - Níveis de Sinal diferenciais CAN



Fonte: (MARINA LACERDA, 2019).

O controlador CAN é um circuito integrado que implementa as camadas de Enlace de Dados e de Controle de Acesso ao Meio do protocolo CAN, conforme definido no modelo OSI. Sua função principal é aliviar a carga de processamento da CPU principal ao lidar com as complexidades do protocolo, como a montagem e desmontagem de quadros, verificação de CRC, arbitragem, e filtragem de mensagens (MICROCHIP, 2019).

O transceptor CAN atua como a interface física entre o controlador CAN e o barramento de pares trançados do veículo. Ele implementa a camada física do protocolo, convertendo os sinais lógicos digitais TX e RX do controlador nos sinais diferenciais do barramento CAN. (NXP, 2016)

2.1.3 On-Board Diagnostics II (OBD-II)

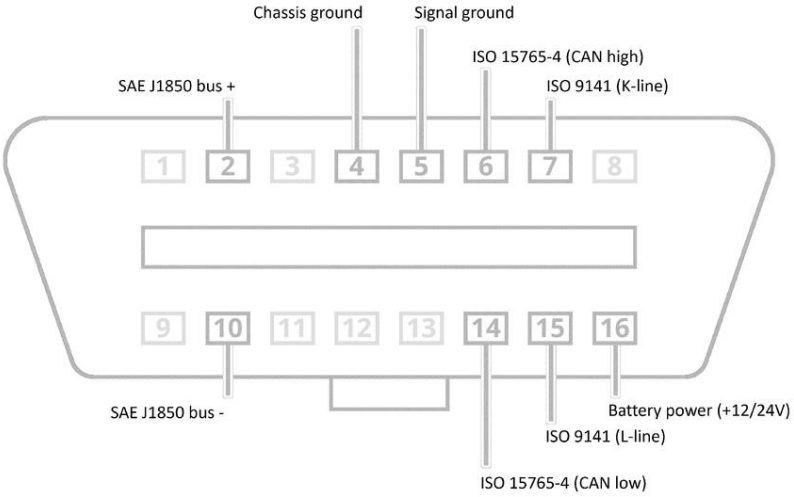
Enquanto o protocolo CAN estabelece o "meio de comunicação" entre as ECUs de um veículo, o padrão OBD-II define uma "linguagem" padronizada para acessar dados de diagnóstico e desempenho das ECUs. Trata-se de um sistema de monitoramento obrigatório regulamentado globalmente, incluindo agências como a *Environmental Protection Agency* (EPA) nos EUA e o Conselho Nacional do Meio Ambiente (CONAMA) no Brasil, cuja adoção tornou-se mandatória para veículos comercializados no país a partir de 2009. Seu objetivo primordial é o controle de emissões de poluentes, estabelecendo um protocolo universal de diagnóstico que permite a interoperabilidade entre diferentes fabricantes e sistemas de inspeção veicular (EPA, 2020) (CONAMA, 2009).

A normatização OBD-II substituiu uma série de protocolos proprietários e conectores diversos das montadoras por um único padrão, garantindo que equipamentos de diagnóstico independentes pudessem se conectar a qualquer veículo vendido no mercado a partir de 1996 nos EUA e 2001 na Europa, desde que compatível, para ler dados padronizados, principalmente relacionados ao sistema de trem de força e às emissões (BOSCH, 2014).

O ponto de acesso físico para o sistema OBD-II é o conector de 16 pinos J1962, mostrado na Figura 3, geralmente localizado na área do motorista, sob o painel. Este

conector fornece acesso à alimentação da bateria (pino 16), ao terra (pinos 4 e 5), e aos barramentos de comunicação do veículo, incluindo os pinos específicos para CAN (pino 6: CAN_H e pino 14: CAN_L). A presença do barramento CAN nestes pinos é obrigatória para veículos leves tornou-se obrigatória no Brasil seguindo o cronograma estabelecido pela Resolução CONAMA 418/2009 (CONAMA, 2009), além dos produzidos a partir de 2008 nos EUA e 2001 para veículos a diesel na Europa (ISO, 2016).

Figura 3 - Conector OBD-II



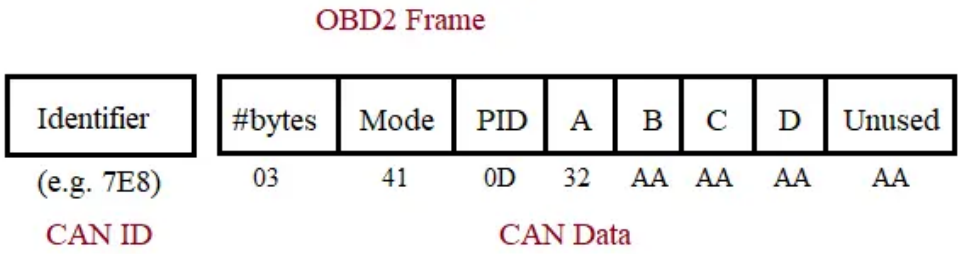
Fonte: o Autor.

A parte principal do padrão OBD-II reside em seu conjunto de Modos de Serviço e Identificadores de Parâmetros (PIDs). Os modos de serviço, definidos pela SAE J1979, são categorias de operações de diagnóstico. O modo mais relevante para a telemetria em tempo real é o Modo 01, que solicita e retorna dados do trem de força enquanto o veículo está em operação. Dentro de cada modo, os PIDs funcionam como chaves que desbloqueiam parâmetros específicos do veículo. Cada PID é um código hexadecimal que, quando solicitado, instrui a ECU a retornar um valor específico (BOSCH, 2014).

A comunicação via OBD-II segue uma estrutura de requisição-resposta, onde o dispositivo que deseja obter a informação deve enviar uma mensagem CAN para o endereço de diagnóstico do veículo. Como mostrado na Figura 4 , a mensagem de requisição contém o modo de serviço e o PID desejado. A ECU responsável responde

com outro frame CAN contendo o modo de serviço, o PID e os dados solicitados, que devem ser decodificados conforme a fórmula padrão definida para aquele PID, como o exemplo exibido na Tabela 1. O ID de requisição geralmente segue o padrão 0x7DF (11 bits) ou 0x18DB33F1 (29 bits).

Figura 4 - Formato de mensagem OBD



Fonte: (RFWIRELESS-WORLD.com).

Tabela 1 - Tabela de Conversão de PIDs

PID (Hex)	Parâmetro	Fórmula de Conversão (Exemplo)	Unidade
0C	Rotação do Motor (RPM)	((A * 256) + B) / 4	rpm
0D	Velocidade do Veículo	A	km/h
5	Temperatura do Líquido de Arrefecimento	A - 40	°C
2F	Nível de Combustível no Tanque	(100 * A) / 255	%

*A: Primeiro byte da resposta; B: Segundo byte da resposta.

Fonte: (SAE, 2021)

2.2 Sistemas Embarcados

Sistemas embarcados são sistemas computacionais completos projetados para desempenhar uma ou poucas funções específicas, com restrições de custo, consumo energético, tamanho físico e desempenho em tempo real. Diferentemente de um computador de propósito geral, um sistema embarcado é tipicamente parte de um dispositivo maior, operando de forma autônoma e sem intervenção humana direta (LAKHTAR, 2023). No contexto automotivo moderno, dezenas de sistemas embarcados (ECUs) gerenciam desde o entretenimento até a segurança do veículo (BOSCH, 2014).

2.2.1 Microcontroladores

Microcontroladores são circuitos integrados que reúnem, em um único chip, todos os componentes de um computador como unidade central de processamento (CPU), memória RAM e *Flash*, e periféricos de entrada/saída (I/O). Essa característica os torna ideais para sistemas embarcados, oferecendo um equilíbrio entre desempenho, consumo e custo para aplicações dedicadas (BARR, 2022).

A funcionalidade de um microcontrolador é drasticamente ampliada pelo seu conjunto de periféricos. Periféricos são circuitos especializados integrados ao chip que gerenciam a interação com o mundo externo, liberando a CPU para tarefas de processamento mais complexas (VALDERRÁBANO, 2023).

Entre os periféricos mais comuns e essenciais para aplicações de interação com o mundo físico, destacam-se:

- Interface Serial Periférica (SPI): Um barramento síncrono de alta velocidade e *full-duplex*, utilizado para comunicação com dispositivos próximos. É caracterizado pelo uso de quatro linhas: SCLK (*clock*), MOSI (dados do mestre para o escravo), MISO (dados do escravo para o mestre) e SS/CS (seleção de escravo). É amplamente usado para conectar sensores, memórias e controladores de interface.
- Comunicação Serial Assíncrona Universal (UART): Um protocolo de comunicação serial assíncrono ponto a ponto que utiliza duas linhas: TX

(transmissão) e RX (recepção). A comunicação é baseada em um acordo pré-estabelecido de velocidade (*baud rate*) entre os dispositivos. É comumente empregado para comunicação com módulos GSM/GPRS, GPS e para debug via console serial.

- *Inter-Integrated Circuit* (I^2C): Um barramento serial síncrono multi-mestre, multi-escravo que utiliza apenas duas linhas bidirecionais: SDA (dados) e SCL (*clock*). É ideal para conectar múltiplos dispositivos de baixa velocidade em um mesmo barramento, como sensores de temperatura, umidade e pressão.
- Conversor Analógico-Digital (ADC): Um periférico crucial que converte tensões analógicas do mundo real em valores digitais que podem ser processados pela CPU. Sua resolução é medida em bits, determinando a precisão da leitura.

2.2.2 Sistema Operacional em Tempo Real (RTOS)

Conforme a complexidade do software em sistemas embarcados aumenta, gerenciar de forma eficiente e confiável múltiplas funções ou tarefas como aquisição de dados e gerenciamento de comunicação, torna-se um desafio significativo. Esta abordagem pode levar a problemas de responsividade, bloqueio de funções de baixa prioridade por outras de alta prioridade e dificuldade em garantir tempos de resposta previsíveis. Para superar estas limitações, recorre-se a um Sistema Operacional em Tempo Real.

Um RTOS é um sistema operacional especializado projetado para gerenciar os recursos de hardware de um microcontrolador e executar aplicações com *timing* preciso e previsível. A característica definidora de um RTOS não é sua velocidade, mas sua capacidade determinística, ou seja, a capacidade de garantir que as operações sejam realizadas dentro de um intervalo de tempo estritamente definido (LABROSSE, 2022).

A programação com um RTOS introduz um paradigma diferente, baseado em concorrência e paralelismo. Seus conceitos mais relevantes para este trabalho incluem as tarefas que são unidades independentes de execução que encapsulam uma função específica, por exemplo, tarefa_leitura_can ou tarefa_comunicacao_ble.

O RTOS é responsável por simular a execução paralela dessas tarefas em um único núcleo de processamento.

Dentre as opções disponíveis, o FreeRTOS destaca-se como uma solução amplamente adotada na indústria, conhecida por sua robustez, portabilidade e licença de código aberto (AWS, 2023). Seu ecossistema inclui portes para diversos microcontroladores, incluindo o ESP32, e implementa todos os mecanismos essenciais de um RTOS.

2.3 Protocolos de Comunicação e Internet das Coisas

A Internet das Coisas (IoT) refere-se a uma infraestrutura composta por sensores, softwares e demais tecnologias embarcadas, cujo propósito é conectar dispositivos físicos à internet, possibilitando o monitoramento, a troca e o processamento distribuído de dados. No contexto automotivo, a IoT viabiliza a concepção de veículos conectados, ampliando o escopo de funcionalidades disponíveis. Entre as aplicações mais relevantes destacam-se a manutenção preditiva, a gestão inteligente de frotas, os serviços baseados em localização e a integração com sistemas de automação residencial (AL-FUQAHA, 2015).

O sistema desenvolvido neste trabalho insere-se nesse cenário, operando como um nó IoT de sensoriamento responsável pela coleta de informações de um ativo físico e pelo seu compartilhamento com plataformas externas. Nesse tipo de arquitetura, a escolha dos protocolos de comunicação desempenha papel central, pois determina como os dados serão transmitidos de forma eficiente, segura e confiável entre o dispositivo e o usuário final.

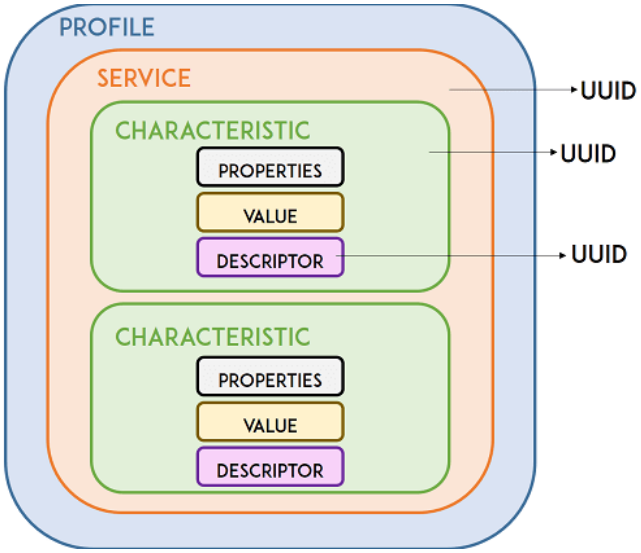
2.3.1 Bluetooth Low Energy

O Bluetooth Low Energy (BLE) é um protocolo de comunicação sem fio de curto alcance, parte da especificação Bluetooth 4.0 e posteriores, projetado especificamente para aplicações que demandam baixo consumo de energia. Diferente do Bluetooth Clássico, focado em transmissão contínua de dados, como áudio, o BLE é otimizado para operar em rajadas curtas de transmissão de pequenos pacotes de

dados, permanecendo a maior parte do tempo em modo de baixo consumo (Bluetooth Core Specification Version 5.3, 2023).

O BLE opera sob um modelo cliente-servidor baseado em atributos. Como pode ser visto na Figura 5 o dispositivo periférico atua como um Servidor *Generic Attribute Profile* (GATT) onde hospeda um conjunto de Serviços, que são coleções lógicas de características. Uma característica é um contêiner para um valor de dado e seus descritores, que configuram como o dado pode ser acessado ou notificado (NORDIC, 2023).

Figura 5 - Arquitetura BLE



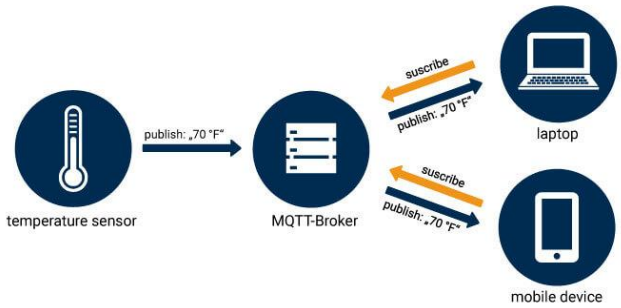
Fonte: (AKB, 2024)

2.3.2 MQTT

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de transporte de mensagens leve, projetado para comunicação eficiente em dispositivos IoT com recursos computacionais limitados, bem como em redes com largura de banda reduzida ou latência variável (BANKS, 2019). Baseado no paradigma publicador/assinante, o MQTT possibilita uma comunicação assíncrona e altamente escalável entre dispositivos distribuídos.

Nesse modelo arquitetural, o cliente responsável pelo envio das mensagens denominado publicador, permanece desacoplado daqueles que as recebem, denominados assinantes. Como ilustrado na Figura 6, a interação entre esses elementos é intermediada por um servidor central denominado broker, o qual gerencia o roteamento, o armazenamento temporário e o direcionamento das mensagens. Os clientes podem publicar ou receber informações por meio de tópicos, que representam canais de comunicação identificados por *strings* hierárquicas, como por exemplo /dadosVeiculo (LIGHT, 2017).

Figura 6 - Arquitetura MQTT



Fonte: (GABRIEL, 2023)

2.4 Metodologia de Projeto em V

O desenvolvimento de um sistema embarcado requer uma abordagem sistemática e metodológica. A adoção de uma metodologia de projeto proporciona um roteiro para guiar o processo de desenvolvimento, mitigar riscos, gerenciar complexidade e aumentar as chances de sucesso do projeto (VALDERRÁBANO, 2023).

Uma das metodologias tradicionais amplamente empregadas na engenharia de sistemas embarcados é o Modelo em V, ou Ciclo de Vida em V. Essa abordagem destaca a importância da verificação e da validação ao longo de todas as etapas do desenvolvimento, assegurando que os requisitos sejam rastreados de forma consistente desde a concepção até os testes finais. Conforme ilustrado na Figura 7, o

modelo organiza-se em duas fases que convergem para a implementação, assumindo a forma da letra “V”: no lado esquerdo, encontram-se as atividades de definição de requisitos e elaboração do projeto; no vértice, é realizada a implementação; e no lado direito estão as etapas de integração, testes e validação. Cada fase de teste corresponde diretamente a uma fase de especificação, garantindo a coerência entre o que foi projetado e o que é efetivamente entregue.

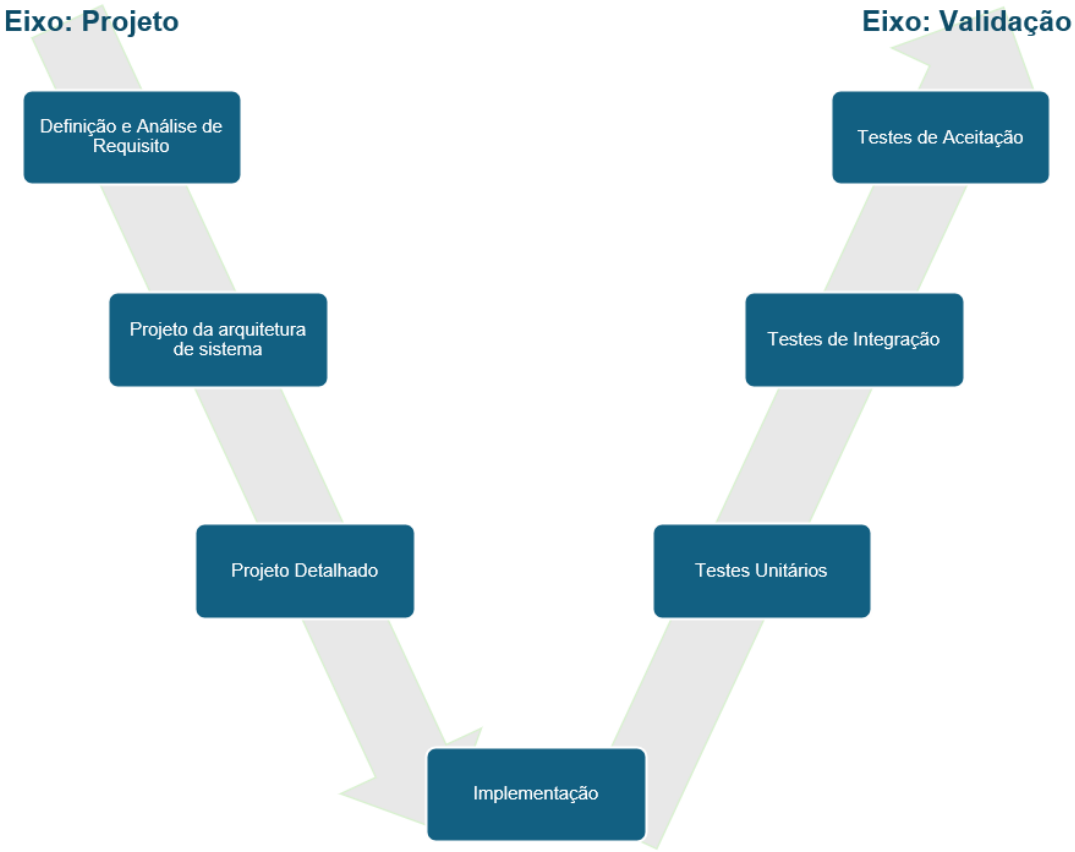
A metodologia se inicia com a Definição e Análise de Requisitos, onde são estabelecidas de forma clara e documentada as funcionalidades que o sistema deve executar, suas restrições e seu desempenho esperado. Em seguida, avança-se para o Projeto de arquitetura do sistema, onde o sistema global é decomposto em subsistemas constituintes de hardware e software, com a definição das interfaces entre eles.

A metodologia continua no projeto detalhado de software/firmware e hardware, onde a arquitetura de software é definida em minúcias, incluindo a modularização, o diagrama de tarefas concorrentes sob o FreeRTOS, o formato das estruturas de dados e os algoritmos de comunicação, enquanto a arquitetura de hardware é especificada através da seleção de componentes, diagramas esquemáticos e definição de interfaces físicas.

O fundo do V representa a fase de Implementação ou Codificação. É nesta etapa que o projeto detalhado é traduzido em código-fonte, soldagem de componentes e montagem do hardware. As decisões e especificações definidas no lado esquerdo do V são materializadas em software e hardware funcionais.

O lado direito se inicia com os testes de unidade, nos quais cada módulo ou função de software é testado individualmente de forma isolada. Superada essa etapa, realizam-se os testes de integração, onde os módulos previamente testados são combinados progressivamente e suas interações são validadas. Com o sistema integrado, executa-se o teste de sistema, que valida o comportamento do produto completo contra todos os requisitos definidos inicialmente, envolvendo testes em bancada e em ambiente real. Finalmente, o ciclo se encerra com o teste de aceitação, que é a validação final perante o usuário ou cliente para confirmação de que o sistema atende plenamente às suas necessidades e expectativas (SOMMERVILLE, 2019).

Figura 7 - Ciclo de Projeto em V



Fonte: O autor.

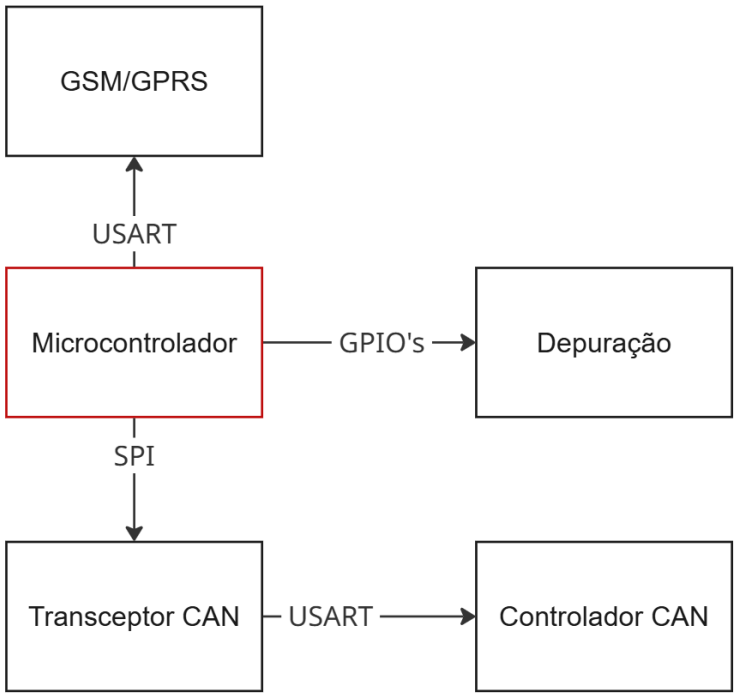
3 DESENVOLVIMENTO DO TRABALHO

Este capítulo detalha o processo de desenvolvimento do sistema de telemetria veicular, seguindo uma adaptação da metodologia em V. A partir dos objetivos e requisitos do sistema definidos no Capítulo 1, são apresentados a arquitetura de hardware e software projetada para atendê-los, a implementação prática e, por fim, a estratégia de testes e validação.

3.1 Projeto do Sistema

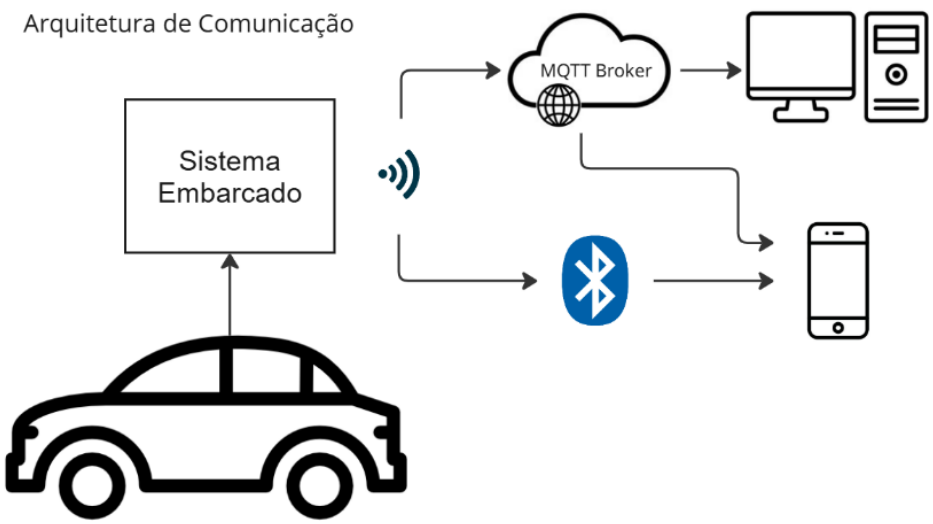
Com base nos requisitos estabelecidos no tópico 1.1, procedeu-se com a etapa de projeto de arquitetura do sistema, onde foram definidos a arquitetura de hardware, Figura 8, e a arquitetura de comunicação necessária para implementar as funcionalidades especificadas, Figura 9.

Figura 8 - Arquitetura de Hardware



Fonte: O autor.

Figura 9 - Arquitetura de Comunicação



Fonte: O autor.

3.1.1 Escolha das Tecnologias de Comunicação

Para atender ao requisito de transmissão remota de dados, avaliou-se soluções baseadas em tecnologias como LoRa, Sigfox e redes celulares, os parâmetros estão resumidos na Tabela 2.

Tabela 2 - Comparação de Tecnologias de Comunicação

Tecnologia	Alcance	Taxa de Dados	Consumo Energético	Custo de Implantação	Custo Operacional
GSM/GPRS	Nacional	~50-100 kbps	Moderado-Alto	Baixo	Moderado
LTE-M/NB-IoT	Nacional	~100-300 kbps	Baixo	Moderado-Alto	Baixo-Moderado
LoRaWAN	Regional	~0.3-50 kbps	Muito Baixo	Moderado	Baixo
Sigfox	Nacional	~100 bps	Muito Baixo	Baixo-Moderado	Por mensagem
Wi-Fi	Local	~10-100 Mbps	Alto	Muito Baixo	Baixo
Bluetooth (BLE)	Local	~1 Mbps	Muito Baixo	Baixo	Nulo

Fonte: O autor.

A seguir uma breve analise de adequação das tecnologias de comunicação elencadas na Tabela 2.

A tecnologia LoRaWAN destaca-se por seu alcance de quilômetros em área rural e consumo energético extremamente reduzido, sendo ideal para sensores estáticos com transmissão esporádica de dados. Contudo, sua arquitetura baseada em gateways fixos cria uma dependência crítica da existência de infraestrutura de cobertura na região de operação do veículo.

A rede Sigfox opera em banda estreita e oferece excelente eficiência energética para transmissão de pequenos pacotes de dados. Similarmente ao LoRaWAN, sua viabilidade está condicionada à disponibilidade de cobertura na área de operação. Apesar de possuir abrangência nacional, sua cobertura é inferior à rede GSM, além de operar em modelo de negócio baseado no número de mensagens transmitidas, o que poderia inviabilizar economicamente o envio contínuo de dados telemáticos (MOUNA, 2022).

O Wi-Fi apresenta limitações críticas que o tornam inadequado para o cenário proposto. A tecnologia depende inteiramente da disponibilidade de redes externas e da inserção de credenciais de acesso para funcionar, o que é impraticável durante o deslocamento do veículo ou em locais sem infraestrutura preexistente (AL-FUQAHA, 2015).

As tecnologias LTE-M (*Long Term Evolution for Machines*) e NB-IoT (*Narrowband IoT*) representam a evolução das redes celulares para aplicações de IoT. Oferecem vantagens significativas em consumo energético quando comparadas ao GSM/GPRS tradicional, além de maior penetração de sinal em ambientes internos e subterrâneos. Entretanto, o custo dos módulos especializados é substancialmente superior ao dos módulos GSM/GPRS, impactando negativamente o custo total da solução (RATAJ, 2022).

Dessa forma, diante da análise realizada, a tecnologia GSM/GPRS mostrou-se como a opção mais adequada ao projeto em sua fase atual devido a fatores como a cobertura geográfica, volume de dados e custo operacional do sistema de telemetria veicular proposto. É necessário destacar que a rede GSM/2G ainda mantém uma ampla cobertura no território nacional, mas esta infraestrutura está em processo progressivo de desativação pelas operadoras. Assim, a implementação adotada utiliza módulo SIM800L comunicando-se com o microcontrolador principal via interface UART, empregando protocolo de comandos AT para estabelecimento de conexão e transmissão de dados através de conexões TCP/IP (WIRATAMA, 2021).

Em complemento, o Bluetooth Low Energy (BLE) foi integrado ao sistema para prover conectividade de curto alcance com dispositivos móveis. Diferente das soluções de longo alcance, o BLE opera em uma faixa de até dezenas de metros com consumo energético extremamente reduzido, sendo ideal para cenários de configuração local.

3.1.2 Hospedagem de Serviço em Nuvem

Em um sistema de telemetria móvel, a escolha da infraestrutura para hospedagem do *broker* MQTT é decisiva para garantir uma comunicação confiável e escalável. Dispositivos móveis, como veículos rastreados ou sensores em trânsito, operam em redes com endereços IP dinâmicos e conexões instáveis, o que impossibilita que atuem como servidores fixos. Essa limitação exige um ponto central permanente na internet, capaz de agregar e gerenciar o fluxo de dados de fontes distribuídas e geograficamente dispersas. Sem um *endpoint* estável, a comunicação em tempo real se tornaria inviável, comprometendo todo o funcionamento do sistema.

Nesse contexto, uma máquina virtual (VM) em cloud, como as oferecidas pela DigitalOcean, Amazon Web Service (AWS), Microsoft Azure ou Google Cloud, surge como uma boa solução técnica. Ao provisionar uma VM, obtém-se um endereço IP público fixo e recursos computacionais dedicados, criando um hub sempre acessível para onde todos os dispositivos móveis direcionam seus dados. Essa centralização assegura alta disponibilidade, mesmo com flutuações na conectividade dos dispositivos, e permite o uso de mecanismos robustos de segurança, como firewalls configuráveis e autenticação por chave. Além disso, a possibilidade de aumentar CPU, RAM ou armazenamento sob demanda, torna a máquina virtual adequada para cenários de crescimento progressivo da frota ou do volume de dados. Entretanto, esta abordagem apresenta a desvantagem do custo operacional contínuo, que pode tornar-se significativo conforme a escala do projeto aumenta, especialmente para aplicações com grande número de dispositivos transmitindo dados constantemente.

Portanto, a escolha da DigitalOcean como provedor em nuvem para hospedar o broker MQTT foi motivada não apenas por seus atributos técnicos como simplicidade, custo acessível e desempenho estável, mas também pela grande documentação e exemplos práticos disponíveis para implementações similares.

3.1.3 Seleção de Componentes de Hardware

A seleção dos componentes de hardware foi orientada pelos requisitos previamente estabelecidos, priorizando o custo-benefício, a disponibilidade no mercado nacional, a robustez para o ambiente automotivo e a adequação técnica às funcionalidades do sistema.

3.1.3.1 Seleção do Microcontrolador

A análise para a escolha do microcontrolador central, mostrado na Tabela 3, considerou plataformas amplamente utilizadas, como Arduino Uno (baseado no ATmega328P), STM32 (família ARM Cortex-M) e ESP32.

O Arduino Uno, embora amplamente difundido devido ao seu baixo custo e ao vasto ecossistema de desenvolvimento, apresenta limitações significativas de memória, apenas 2 KB de RAM e de capacidade de processamento, baseada em uma arquitetura de 8 bits operando com um *clock* de apenas 16 MHz. Tais restrições o tornam inadequado para aplicações que demandam execução de um sistema operacional de tempo real e gerenciamento concorrente de múltiplas pilhas de comunicação (ARDUINO, 2023).

Os microcontroladores da família STM32, por sua vez, oferecem desempenho superior, ampla variedade de periféricos avançados e, em algumas variantes, controlador CAN integrado. Porém, a necessidade de módulos adicionais para prover conectividade sem fio acarretaria maior complexidade de integração, aumento no consumo energético e elevação do custo total da solução (STMICROELECTRONICS, 2022).

Diante desse cenário, o ESP32 foi selecionado por apresentar o melhor equilíbrio entre desempenho, custo e funcionalidades integradas. Seu processamento em 32 bits, aliado à arquitetura dual-core, garante recursos suficientes para a execução do FreeRTOS e para o gerenciamento eficiente de tarefas concorrentes, como aquisição e transmissão de dados (ESPRESSIF, 2023). Além disso, a integração nativa de Wi-Fi e Bluetooth Low Energy (BLE) elimina a necessidade de módulos externos, simplificando o projeto de hardware e reduzindo custos.

Tabela 3 - Comparação de Microcontroladores

Característica	Arduino Uno (ATmega328P)	STM32F103C8T6 (ARM Cortex-M3)	ESP32-WROOM-32 (Xtensa LX6)	Melhor Caso de Uso
Arquitetura	AVR 8-bit	ARM Cortex-M3 32-bit	Xtensa LX6 32-bit (Single/Dual-Core)	-
Freq. Clock	16 MHz	72 MHz	160 ou 240 MHz	ESP32
Memória Flash	32 KB	64 KB	4 MB	ESP32 (Armazenamento de código)
Memória RAM	2 KB	20 KB	520 KB	ESP32 (Manipulação de dados)
Conectividade Nativa	UART, I²C, SPI	UART, I²C, SPI, I²S, USB	Wi-Fi, Bluetooth/BLE, UART, I²C, SPI, I²S	ESP32 (Conectividade sem fio)
Periféricos Avançados	ADC 10-bit	ADC 12-bit, DAC, DMA, CAN	ADC 12-bit, DAC, DMA	STM32 (Controle preciso e interface CAN nativa)
Custo (Aprox.)	Médio (R$ 30-40)	Baixo (R$ 20-30)	Médio (R$ 30-40)	STM32 (Custo inicial)
Facilidade de Desenvolvimento	Excelente (Arduino IDE)	Moderada (STM32CubeIDE, Mbed OS)	Boa (Arduino IDE ou ESP-IDF)	Arduino/ESP32 (Prototipagem rápida)
Ecossistema	Maior número de bibliotecas e tutoriais	Robusto, mas mais complexo	Muito vasto e em crescimento	Arduino (Simplicidade e comunidade)

Fonte: O autor

3.1.3.2 Seleção de Controlador CAN

Para a interface com o barramento veicular, optou-se pela solução consolidada baseada no controlador CAN MCP2515 e no transceptor TJA1050. O controlador MCP2515 comunica-se com o microcontrolador através de interface SPI, gerenciando autonomamente as camadas de enlace de dados do protocolo CAN. O transceptor TJA1050 desempenha função crucial na interface física, convertendo os sinais lógicos do controlador para os níveis diferenciais do barramento CAN e providenciando isolamento elétrico e proteção contra transientes de tensão.

3.1.4 Ferramentas de Depuração e Monitoramento

Para garantir a robustez e confiabilidade do sistema, devem ser implementadas ferramentas de depuração e monitoramento que permitam verificar o funcionamento em tempo real e diagnosticar possíveis falhas. A escolha das ferramentas considerou a simplicidade de implementação, o baixo custo e a eficácia na identificação de problemas durante as fases de desenvolvimento e operação.

A principal ferramenta de depuração adotada foi a comunicação serial, utilizada para envio de mensagens de depuração e informações de status do sistema. Através do monitor serial, foi possível acompanhar o fluxo de execução das tarefas, verificar valores de variáveis críticas e identificar eventuais erros nas operações de comunicação. A implementação incluiu diferentes níveis de mensagens de depuração como informacional e erro. Adicionalmente, foram implementados três LEDs indicadores com funções específicas de monitoramento visual:

- LED de Alimentação: Indicador de alimentação do sistema, permanece continuamente aceso quando o sistema está devidamente energizado e operacional.
- LED de Status CAN: Atua como indicador de funcionamento do barramento CAN. Em condições normais de operação, pisca periodicamente indicando a recepção de mensagens do veículo. Em situações de erro na comunicação CAN (perda de conexão, falha na decodificação de mensagens), permanece aceso continuamente, sinalizando a necessidade de intervenção.
- LED de Atividade BLE: Indicador de comunicação Bluetooth Low Energy, pisca sempre que uma mensagem é transmitida via protocolo BLE, permitindo verificar visualmente a atividade de comunicação com dispositivos móveis.

3.2 Projeto Detalhado

3.2.1 Projeto de Firmware

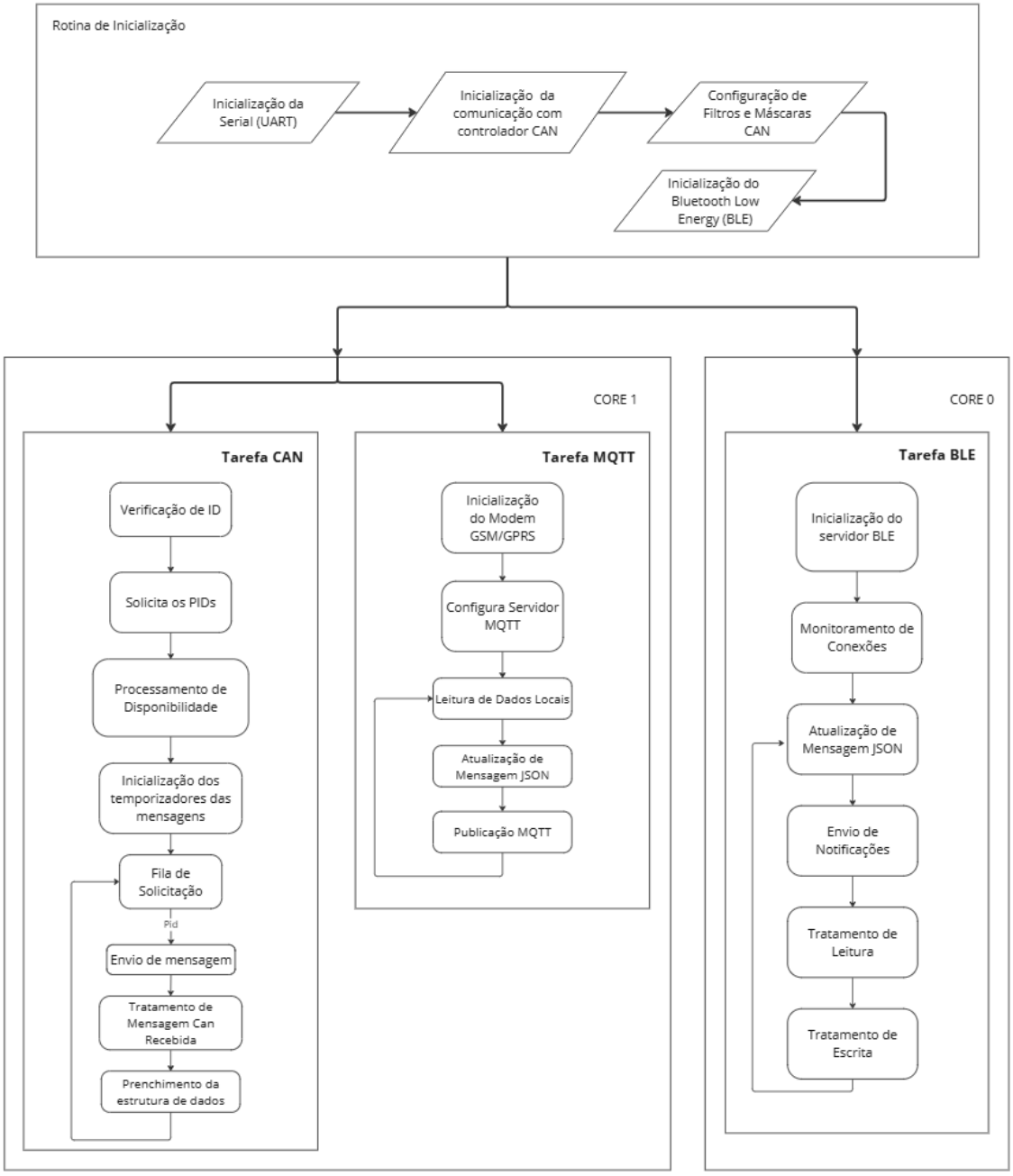
O firmware corresponde ao software embarcado que atua diretamente sobre o hardware, implementando as funcionalidades essenciais do dispositivo e coordenando o funcionamento de seus periféricos. No sistema desenvolvido, o fluxo operacional do firmware, ilustrado na Figura 10, inicia-se com a configuração e inicialização dos módulos e interfaces de comunicação, progredindo para a execução concorrente de tarefas que realizam a aquisição, o processamento e o envio dos dados coletados.

Uma alternativa comum em sistemas embarcados é a abordagem *bare metal*, na qual não há um sistema operacional, todo o controle de tempo e lógica de execução depende exclusivamente do código da aplicação, frequentemente estruturado em laços principais e rotinas de interrupção. Embora seja apropriada para aplicações simples e com número reduzido de eventos concorrentes, essa abordagem tende a tornar-se complexa e menos previsível quando a aplicação exige múltiplas tarefas de forma simultânea e com diferentes prioridades.

Diante disso, optou-se pelo uso do FreeRTOS, um sistema operacional de tempo real que permite estruturar o firmware em tarefas independentes, com escalonamento preemptivo baseado em prioridades. Essa característica garante que rotinas críticas, tais como, a leitura do barramento CAN mantenham comportamento determinístico e não sejam interrompidas por operações secundárias. Além disso, a organização modular do código facilita a manutenção, a depuração e a expansão futura do sistema.

A arquitetura dual-core da ESP32 foi estrategicamente utilizada para otimizar o desempenho do sistema, alocando a tarefa crítica de aquisição CAN no *Core* 1 (*Application Core*) para garantir temporalidade precisa e baixa latência, enquanto a tarefa de comunicação BLE foi designada ao *Core* 0 (*Protocol Core)* para aproveitar as dependências intrínsecas da função Bluetooth da fabricante Espressif. A tarefa de comunicação MQTT opera através de comandos AT para controle do modem GSM/GPRS, compartilha o *Core* 1 sem competição significativa por recursos.

Figura 10 - Arquitetura de Firmware



Fonte: O autor.

3.2.1.1 Rotina de Inicialização

1. Inicialização da Serial (UART): Configura a portas e *baud rate* da comunicação serial para depuração e troca de mensagens com o módulo GSM.

2. Inicialização da comunicação com controlador CAN: Configura o controlador CAN MCP2515 via SPI e o inicializa para operação no barramento veicular.

3. Configuração de Filtros e Máscaras CAN: Ajusta os filtros de hardware do controlador CAN para aceitar apenas as mensagens com IDs relevantes, ignorando o tráfego irrelevante do barramento.

4. Inicialização do Bluetooth Low Energy (BLE): Configura o servidor BLE, definindo o serviço principal e suas características para permitir comunicação sem fio com um aplicativo.

3.2.1.2 Tarefa CAN

1. Verificação de ID: Verifica o tipo de protocolo CAN utilizado pelo veículo (Padrão ou Estendido) para ajustar a configuração do controlador CAN e o formato das mensagens de requisição.

2. Solicita os PIDs Disponíveis no OBD-II: Envia mensagens de requisição OBD-II para descobrir quais parâmetros estão disponíveis para consulta no veículo.

3. Processamento de Disponibilidade: Processa a resposta da ECU, convertendo os dados hexadecimais recebidos em uma máscara binária onde cada bit representa a disponibilidade (1) ou não (0) de um PID específico.

4. Inicialização dos temporizadores das mensagens: Configura temporizadores individuais para cada PID disponível, definindo períodos de solicitação

personalizados com base na taxa de atualização necessária para cada tipo de dado (ex: RPM com atualização mais frequente que temperatura).

5. Fila de Solicitação: Preenche a fila com os PID solicitados para aquisição.

6. Envio de mensagem: Rotina que monta e envia mensagens de requisição OBD-II para o barramento CAN, utilizando o formato correto (Padrão ou Estendido) conforme detectado.

7. Tratamento de Mensagem CAN Recebida: Acionada por interrupção, verifica o ID da mensagem recebida, extrai o conteúdo da mensagem recebida.

8. Preenchimento da estrutura de dados: Decodifica os dados hexadecimais da mensagem de resposta conforme a fórmula definida pelo padrão OBD-II para o PID em questão e atualiza uma estrutura global de dados do veículo, compartilhada com outras tarefas do sistema.

3.2.1.3 Tarefa MQTT

1. Inicialização do Modem GSM/GPRS: Verifica as informações básicas do módulo e do estado do cartão SIM. Configura parâmetros de acesso específicos de cada operadora, incluindo o ponto de acesso (APN) e as credenciais de autenticação correspondentes.

2. Configura servidor MQTT: Estabelece conexão GPRS para obter conectividade com a Internet, configura parâmetros de conexão com o broker MQTT (endereço do servidor, porta, credenciais de acesso e tópicos).

3. Leitura de Dados Locais: Acessa a estrutura global de dados para obter os valores mais recentes dos parâmetros capturados do barramento CAN.

4. Atualização de Mensagem JSON: Converte os dados telemétricos em um formato JSON. Exemplo: {"rpm": 2100, "vel": 85, "temp": 92}.

5. Publicação MQTT: Envia a mensagem JSON para o *broker* MQTT no tópico configurado.

3.2.1.4 Tarefa BLE

1. Inicialização do Servidor BLE: Configuração do servidor BLE com os serviços e características GATT definidos para a aplicação. Definição das propriedades das características (leitura, escrita, notificação).

2. Monitoramento de Conexões: Verificação constante do estado das conexões BLE. Gerenciamento de eventos de conexão e desconexão de clientes.

3. Atualização de Mensagem JSON: Acesso à estrutura global de dados do veículo para obter valores atualizados. Atualização dos valores das características GATT com novos dados telemétricos.

4. Envio de Notificações: Envio de notificações para todos os clientes inscritos quando as características GATT são atualizadas com novos dados.

5. Tratamento de Leitura: Resposta a solicitações de leitura das características GATT por dispositivos clientes. Retorno dos valores atuais dos parâmetros veiculares.

6. Tratamento de Escrita: Processamento de comandos recebidos via características GATT com permissão de escrita.

3.2.2 Projeto de Hardware

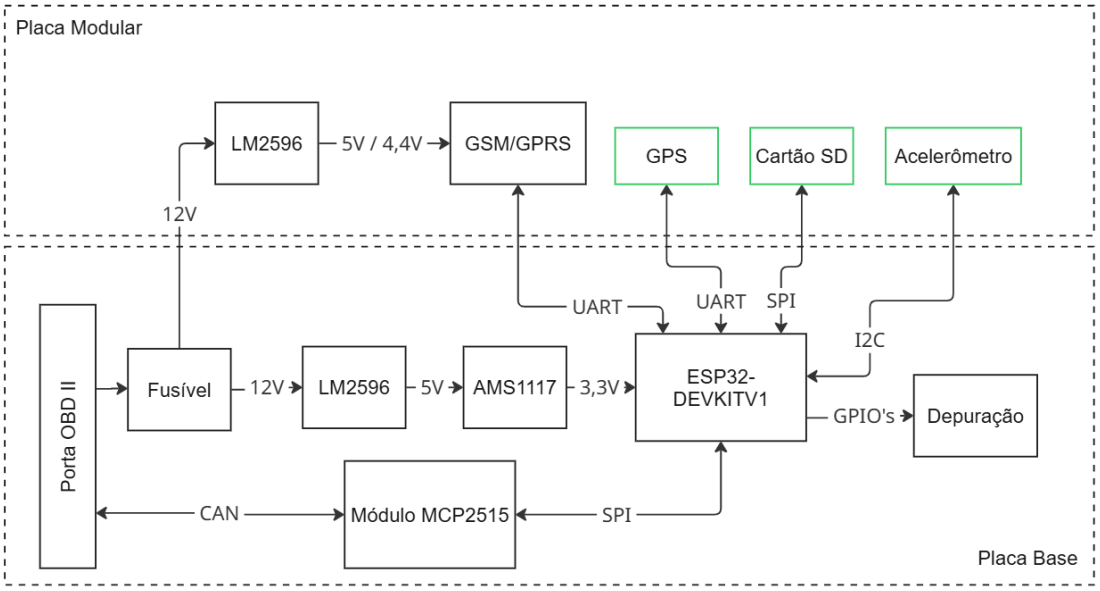
Baseado nos componentes escolhidos em 3.1.3, o projeto da placa de circuito impresso (PCI) foi desenvolvido no software KiCad, uma plataforma de desenvolvimento gratuita de fácil utilização e grande número de usuários. Foi adotada uma arquitetura de sistema modular em duas camadas, buscando proporcionar maior flexibilidade para futuras expansões, permitindo que diferentes aplicações possam utilizar e processar os dados veiculares adquiridos.

A arquitetura da PCI foi organizada em dois blocos principais:

Placa base que é responsável pela execução central do sistema, integrando o microcontrolador ESP32, a interface CAN e os circuitos essenciais de alimentação e proteção e a Placa modular que é dedicada à comunicação através da rede móvel celular, contendo o módulo SIM800L e dispondo de conectores compatíveis com protocolos como I²C e SPI, possibilitando a integração com módulos de comunicação alternativos ou sensores adicionais conforme a necessidade da aplicação.

A Figura 11 mostra a arquitetura física projetada, dando como sugestões alguns componentes complementares que podem ser utilizados futuramente como: GPS para aplicações de rastreamento, acelerômetros para análises de impacto e comportamento do motorista, e Cartão SD para aplicações de armazenamento local de dados.

Figura 11 - Arquitetura Física de Hardware



Fonte: O autor.

Para o sistema de alimentação, foram implementados dois conversores buck LM2596 independentes. O primeiro fornece 5V regulados para a placa principal, enquanto o segundo é dedicado exclusivamente ao módulo SIM800L fornecendo 4.4V ou 5V, a depender do modelo, garantindo estabilidade da alimentação do módulo Sim800L. Esta separação previne interferências e queda de tensão durante os picos de corrente característicos das transmissões GSM.

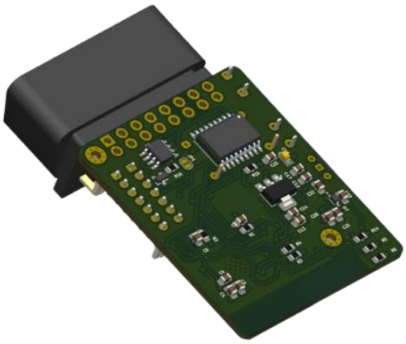
Com o objetivo de minimizar a área ocupada pela eletrônica, a placa base foi projetada utilizando componentes do tipo *Surface-Mount Device* (SMD), ou seja, dispositivos montados diretamente sobre a superfície da placa sem a necessidade de perfurações, o que favorece maior compacidade e melhor desempenho elétrico. A Figura 12 e Figura 13 ilustram o resultado dessa abordagem.

Já a subplaca modular foi desenvolvida empregando componentes *Plated Through-Hole* (PTH), que utilizam terminais inseridos em furos metalizados da placa. Esse tipo de montagem proporciona maior robustez mecânica e facilita a substituição de módulos durante testes e experimentações, como mostrado na Figura 14. A Figura 15 apresenta o arranjo físico planejado para o conjunto das placas.

Figura 12 - Parte Superior da Placa Base

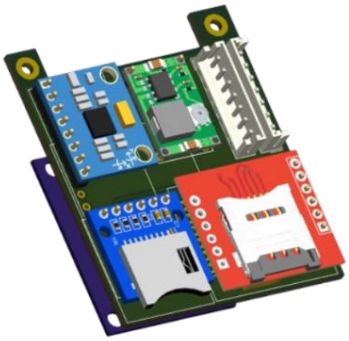
Fonte: O autor.

Figura 13 - Parte Inferior da Placa Base



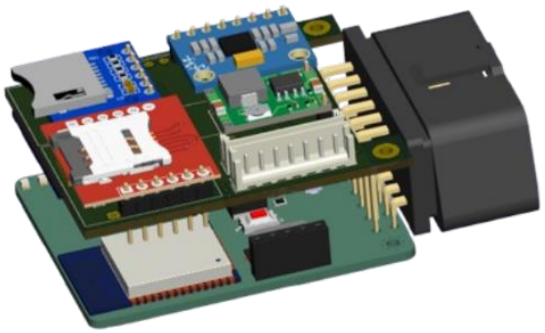
Fonte: O autor.

Figura 14 - Placa Modular



Fonte: O autor.

Figura 15 - Montagem das Placas



Fonte: O autor.

3.3 Implementação

3.3.1 Codificação

O código-fonte foi estruturado em módulos especializados, seguindo uma abordagem de design orientado a objetos mesmo utilizando linguagem C/C++. A organização em pastas proporcionou separação clara de responsabilidades:

- Módulo CAN: Responsável por toda a comunicação com o barramento veicular
  - CanMsgHandling.hpp: Implementa a máquina de estados para processamento de mensagens OBD-II e cálculo de valores físicos mediante fórmulas específicas de cada PID
  - CollectedDataStruct.h: Define a estrutura de dados global para armazenamento dos valores convertidos, permitindo acesso thread-safe pelas demais tarefas
  - Definitions.h: Centraliza todos os “*defines*”, constantes e mapeamentos de PIDs e periféricos
  - CANFunctions.cpp/h: Contém as funções de baixo nível para controle do MCP2515, formatação e envio de mensagens
- Módulo Circular Buffer: Implementa o buffer circular para gestão da fila de PIDs a serem processados, incluindo verificação de disponibilidade no veículo.

- Módulo TickerISR: Utiliza a biblioteca Ticker para gerar interrupções temporizadas que disparam a aquisição dos PIDs em diferentes intervalos.
- Módulo GPRS: Gerencia toda a comunicação com o módulo SIM800L, incluindo inicialização, comandos AT e protocolo de comunicação

A seleção das bibliotecas e do framework de desenvolvimento foi realizada com base nos critérios de documentação e suporte da comunidade. Foi escolhido trabalhar pelo framework Arduino, em detrimento do ESP-IDF nativo, devido à maior simplicidade de desenvolvimento, vasto ecossistema de bibliotecas compatíveis.

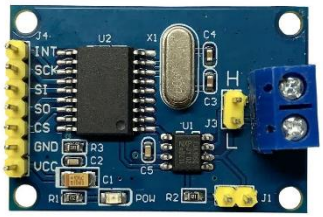
3.3.2 Montagem de Hardware

Para realização de testes, a implementação física do sistema utilizou uma placa perfurada para a montagem dos componentes principais: a placa de desenvolvimento ESP32 DevKit V1, o módulo MCP2515 para interface CAN e o módulo SIM800L para comunicação GSM (Figura 16, Figura 17 e Figura 18, respectivamente).

Figura 16 - ESP32 DEV KIT V1

Fonte: O Autor.

Figura 17 - Módulo MCP2515



Fonte: O Autor.

Figura 18 - Módulo Sim800L

Fonte: o Autor

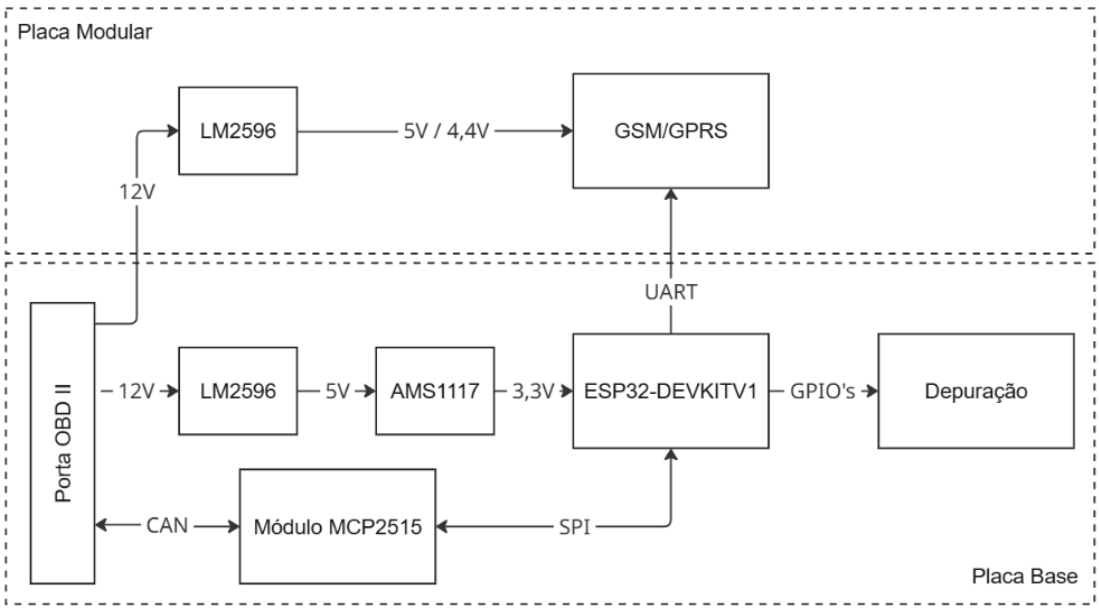
A alimentação do sistema foi baseada no regulador buck LM2596, Figura 19, responsável por converter os 12V da bateria do veículo para 5V, seguido por um regulador linear AMS1117-3.3V para fornecer a tensão de 3,3V requerida pelo ESP32.

Figura 19 - Conversor Buck LM2596

Fonte: (Mercado Livre)

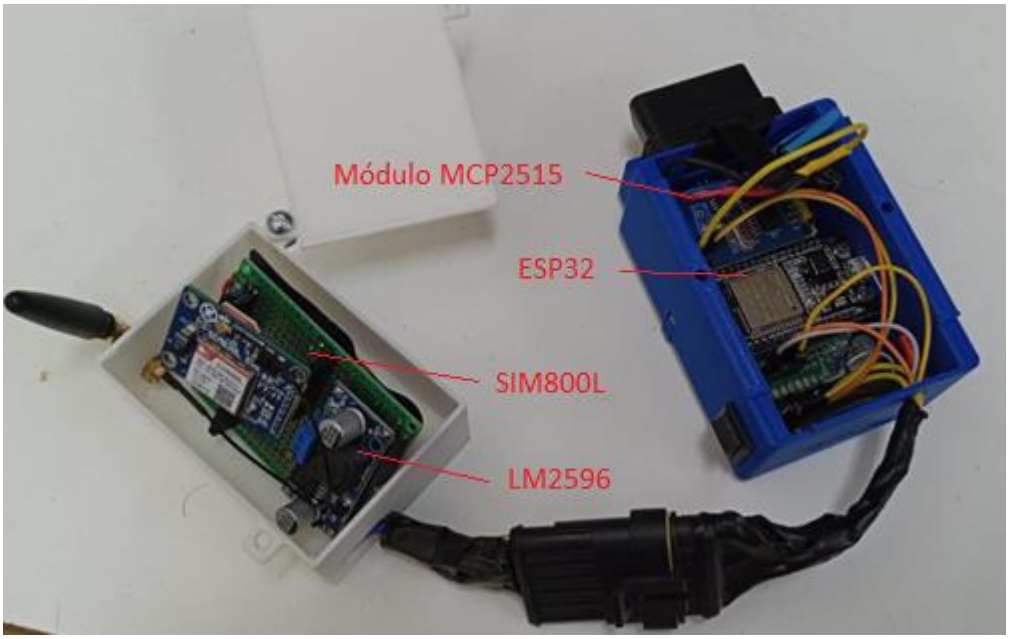
Foi adaptado um cabo com conector OBD-II, que forneceu tanto acesso ao barramento CAN quanto à alimentação de 12V, permitindo uma instalação simplificada. A Figura 20 mostra a arquitetura de montagem executada, já a Figura 21 mostra a montagem dos componentes selecionados.

Figura 20 - Arquitetura de Montagem para Teste de Hardware



Fonte: O autor.

Figura 21 - Montagem Inicial do Sistema



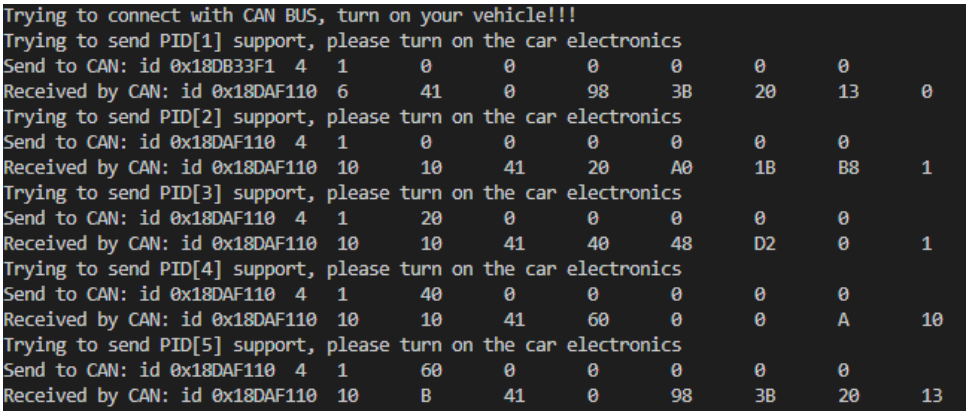
Fonte: O autor.

3.4 Testes de Unidade

Seguindo o lado direito do ciclo V, inicialmente, foi utilizado o hardware montado em placa perfurada e módulos comerciais, sendo a PCI projetada no Kicad utilizada posteriormente. Nos testes de unidade, cada tarefa desenvolvida foi testada de forma individual com seu componente de hardware. Realizou-se o teste de comunicação com o veículo, observando o envio e recebimento de mensagens utilizando a porta OBD-II.

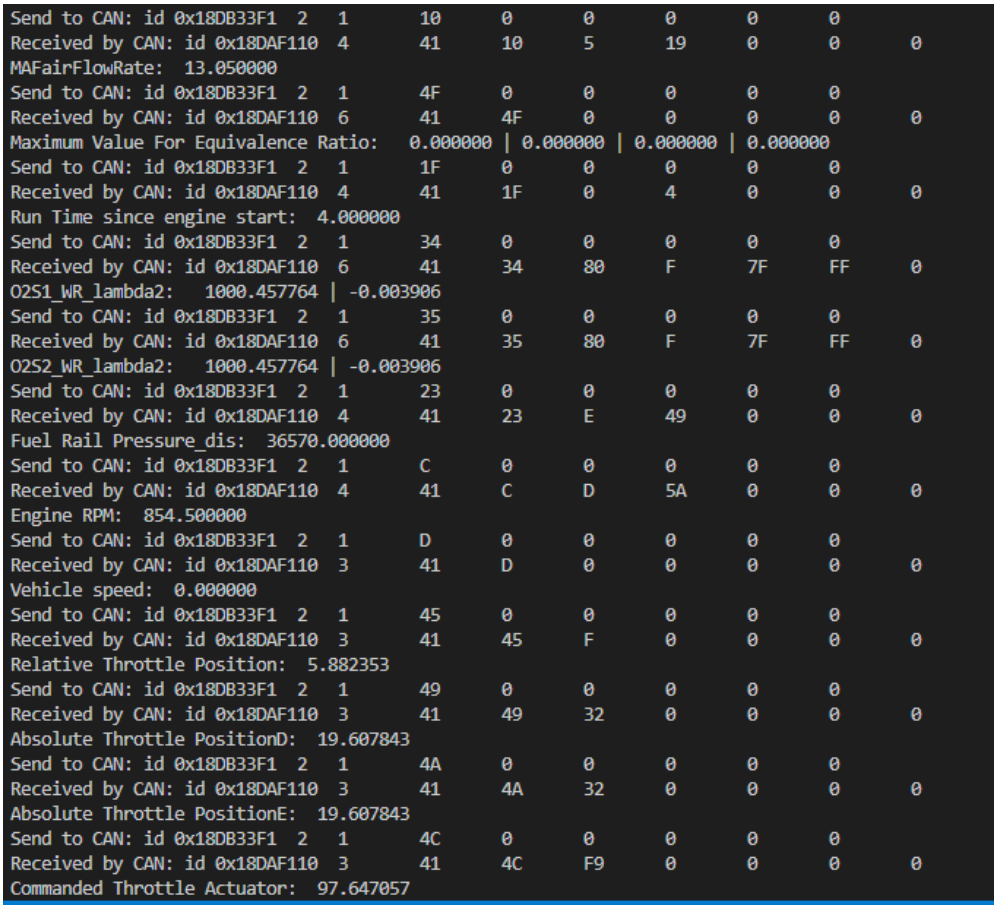
A Figura 22 mostra o resultado do teste de detecção de PIDs e a Figura 23 mostra a resposta à solicitação de mensagens no Modo 01. Já na Figura 24 é possível observar a decodificação do conteúdo das mensagens recebidas. Esses testes foram realizados em dois automóveis, um Jeep Renegade 2015 e um Fiat Pulse 2023, Figura 25, sendo obtidos resultados positivos em ambos.

Figura 22 - Solicitação e Resposta de PIDs disponíveis no barramento



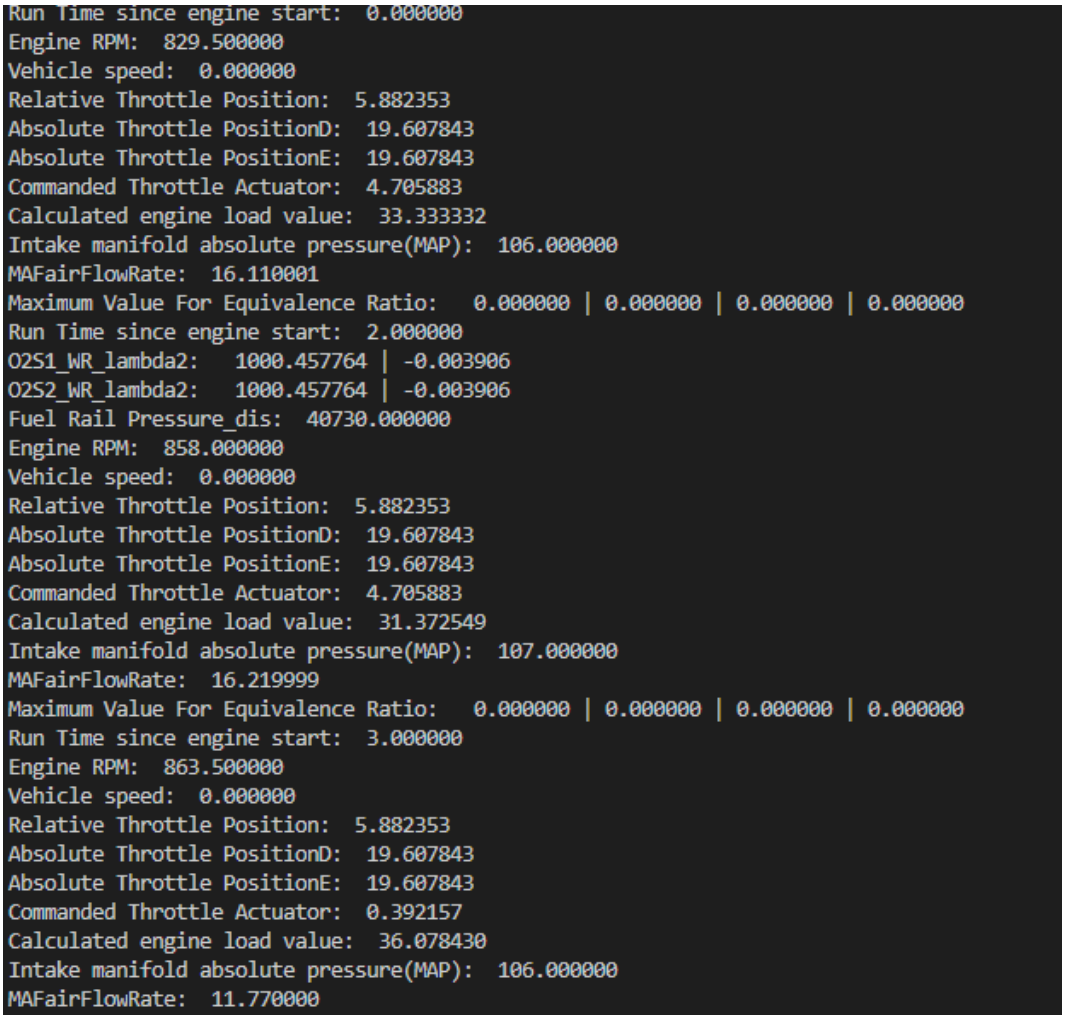
Fonte: O autor.

Figura 23 - Envio e Recebimento de Mensagens no Modo 01



Fonte: O autor.

Figura 24 - Decodificação do conteúdo das mensagens



Fonte: O autor.

Figura 25 - Veículos Testados

Fonte: O autor.

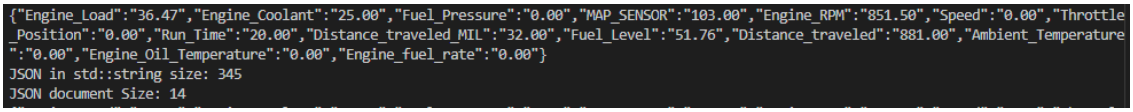
3.5 Testes de Integração

Nos testes de integração, os módulos previamente testados foram combinados, sendo testadas as Tarefas CAN + BLE e CAN + MQTT. A comunicação BLE foi testada utilizando o aplicativo nRF Connect (disponível na Play Store) em um smartphone Android. Como mostra a Figura 26, o dispositivo móvel conseguiu identificar e conectar-se ao servidor BLE do ESP32, assinando a característica responsável pelos dados OBD-II, enviados no formato exibido na Figura 27. Os valores transmitidos foram exibidos em tempo real no aplicativo, comprovando a eficácia da comunicação sem fio e a estruturação correta do pacote de dados.

Figura 26 - Recebimento de Mensagem BLE em aparelho celular

Fonte: O autor.

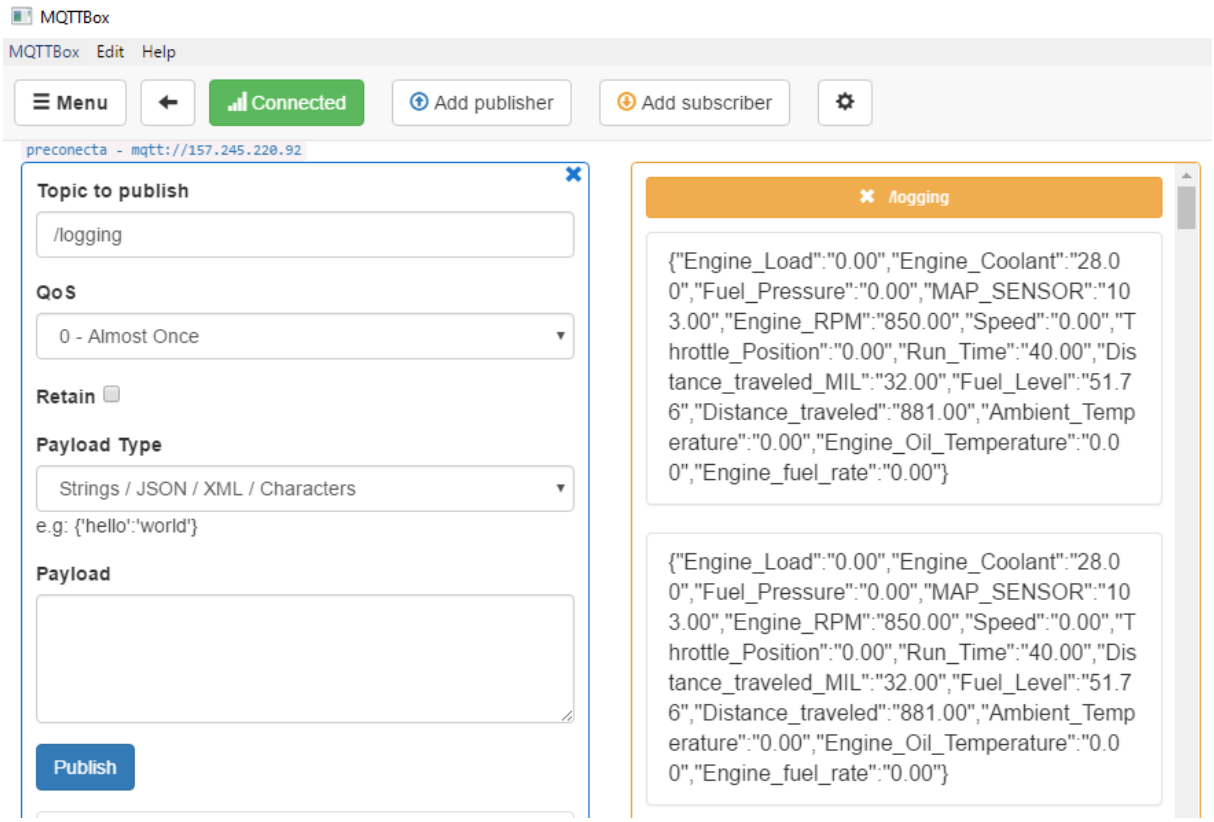
Figura 27 - Formatação de mensagem JSON com os dados veiculares



Fonte: O autor.

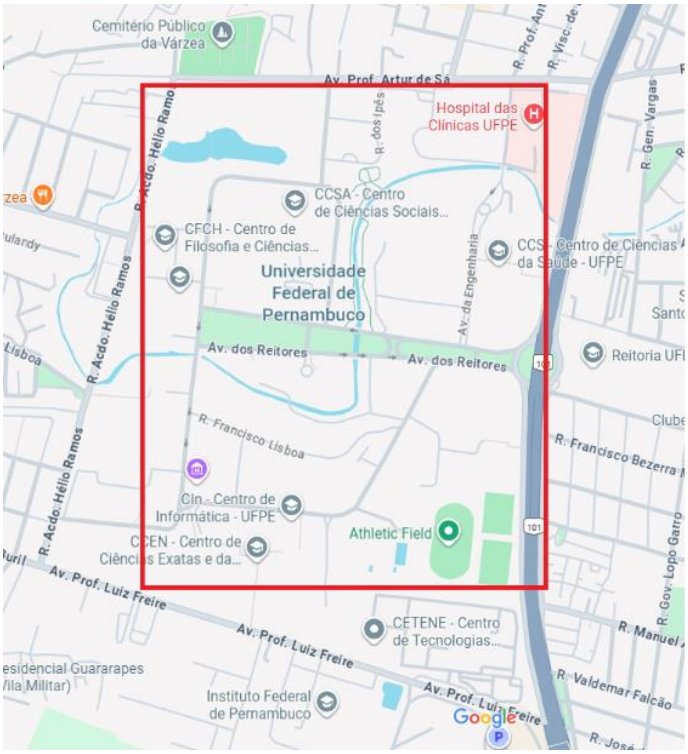
Logo após, avaliou-se o recebimento e a publicação dos dados no *broker* MQTT hospedado na nuvem. Como mostrado na Figura 28, foi utilizado o programa MQTTBox para monitorar as mensagens no tópico assinado. É necessário destacar que foi embarcado no sistema um SIM card da operadora de telefonia CLARO, sendo assim, espera-se que o sistema apenas envie dados dentro da área de cobertura da operadora. Devido à restrições de utilização dos veículos fora do limites da Universidade Federal de Pernambuco, o teste foi realizado dentro do espaço da Universidade, Figura 29.

Figura 28 - Recebimento de Mensagens no Tópico Assinado



Fonte: O autor.

Figura 29 - Área de testes delimitada



Fonte: O autor.

3.6 Testes de Sistema

Com o sistema completo, foi executado o teste de sistema com a PCI projetada, Figura 30, averiguando a execução de todas as tarefas paralelamente. Como esperado, o funcionamento do sistema atendeu as expectativas.

Figura 30 - Sistema Instalado na Porta OBD-II

Fonte: O autor.

4 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

Este trabalho apresentou uma abordagem metodológica completa para o desenvolvimento de sistemas embarcados aplicados à telemetria veicular, abrangendo desde a seleção de componentes até a arquitetura de firmware. Os testes realizados demonstraram que o sistema é capaz de realizar a captura, tratamento e transmissão de dados veiculares. A decodificação OBD-II mostrou-se precisa, a comunicação BLE estável e a integração com MQTT eficaz.

Esses resultados validam a arquitetura proposta e abrem caminho para expansões futuras que podem ser exploradas em continuidades deste trabalho, como:

- Implementação de Armazenamento de Dados Locais: Para aumentar a confiabilidade em cenários com intermitência de conexão, pode-se implementar um banco de dados local com a utilização de cartões SD. Isso permitiria o armazenamento temporário dos dados em caso de perda de conectividade com a nuvem, com sincronização posterior assim que a comunicação for restaurada.

- Integração de Serviços de Localização em Tempo Real: A fusão de dados de um GPS com os parâmetros OBD-II possibilitaria análises mais ricas, como monitoramento de rotas, detecção de frenagens bruscas ou acelerações excessivas.

- Desenvolvimento de *Dashboard* Analítico Personalizado: A criação de uma interface poderia exibir históricos de telemetria, relatórios de consumo de combustível, eficiência do motor e demais métricas relevantes.

- Aprimoramentos de Segurança: A adoção de certificados SSL/TLS para criptografia *end-to-end* na comunicação MQTT, além da implementação de autenticação mais robusta entre o dispositivo móvel e o *broker*, são essenciais para proteger os dados contra interceptações ou acessos não autorizados.

- Testes em Escala e em Condições Reais Diversas: Validar o sistema em uma frota de veículos, sob diferentes condições de tráfego, clima e conectividade, será crucial para refinar a estabilidade e o desempenho da solução em produção.

5 REFERÊNCIAS

AKB. What is Bluetooth Low Energy (BLE)? Everything You Need to Know. **campuscomponent**, 2024. Disponível em: <https://www.campuscomponent.com/blogs/post/what-is-ble-bluetooth-low-energy-explained?srsltid=AfmBOoqJxYaWty27p-2MFQYBPP2N1cpUDbJA91m8VnnaCEl54zLVwNbJ>. Acesso em: 05 ago. 25.

ALAM, M. M. **A Survey on Automotive Telematics:** Past, Present, and Future. 8. ed.

AL-FUQAHA, A. **Internet of Things:** A Survey on Enabling Technologies, Protocols, and Applications. 4. ed.

ARDUINO. **Arduino Uno Rev3 Technical Specifications**. [S.l.]. 2023.

AWS. **FreeRTOS Reference Manual**. AMAZON WEB SERVICES. [S.l.]. 2023.

BANKS, A. **MQTT Version 5.0.** OASIS Standard. [S.l.]. 2019.

BARR, M. **Programming Embedded Systems:** With C and GNU Development Tools.

Bluetooth Core Specification Version 5.3. BLUETOOTH SIG. [S.l.]. 2023.

BOSCH. **CAN Specification Version 2.0**. [S.l.]. 1991.

BOSCH. **Automotive Electrics and Automotive Electronics**.

BOSCH, Robert. **Diagnostic Systems for OBD-II**. Bosch. [S.l.]. 2014.

CHEN, L. **Machine Learning for Vehicle Telematics:** A Review.

CONAMA. **Resolução CONAMA**. Conselho Nacional do Meio Ambiente. [S.l.]. 2009. (nº 418).

DAVIS, R. A Review of Controller Area Network. Computing & Control Engineering Journal. **Computing & Control Engineering Journal**, 2013.

DEVMEDIA. Ciclos de Vida do Software. **devmedia.com**, 2011. Disponível em: <https://www.devmedia.com.br/ciclos-de-vida-do-software/21099>.

EPA. **OBD (On-Board Diagnostics) Regulations and Requirements**. Environmental Protection Agency. Washington, DC: United States. 2020.

ESPRESSIF. **ESP32-WROOM-32**. [S.l.]. 2019.

ESPRESSIF. **ESP32 Series Datasheet Version 4.6.** [S.l.]. 2023.

GABRIEL, Leonardo E. L. MQTT. **ufrj.br**, 2023. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2023-1/trabalhos/Grupo01>. Acesso em: 05 ago. 2025.

GUBBI, J. **Internet of Things (IoT):** A vision, architectural elements, and future directions. 7. ed.

ISO. **ISO 11898-2:2016. Road vehicles — Controller area network (CAN)**. [S.l.]. 2016.

ISO. **ISO 15031-3:2016. Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics**. International Organization for Standardization. [S.l.]. 2016.

LABROSSE, J. J. **MicroC/OS-II:** The Real-Time Kernel.

LAKHTAR, M. **Embedded Systems:** Design and Applications.

LI. **Big Data Analytics in Intelligent Transportation Systems:** A Survey. 6. ed.

LIGHT, R. A. **Introduction to MQTT for IoT**. INTERNATIONAL CONFERENCE ON INTERNET OF THINGS. [S.l.], p. 1-6. 2017.

MARINA LACERDA, Tamine A. Y. N. UFRJ. **ufrj.br**, 2019. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/can/arquitetura>. Acesso em: 01 ago. 2025.

MICROCHIP. **AN713: Controller Area Network (CAN) Basics**. Microchip Technology Inc. [S.l.]. 2019.

MOUNA, M. **Performance Analysis of Sigfox Technology for IoT Applications.** IEEE. [S.l.]. 2022.

NIKOLOV, Neven; GOTSEVA, Daniela. **Make a prototype of IoT connected diagnostic tool using ESP32 and MQTT for reading data from car CAN bus OBD2.** 59th International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST 2024. [S.l.]: IEEE. 2004.

NORDIC. **nRF5 SDK Documentation: Bluetooth Low Energy Applications**. NORDIC SEMICONDUCTORS. [S.l.]. 2023.

NXP. **Application Note: TJA1050 High-Speed CAN Transceiver**. NXP Semiconductors. [S.l.]. 2016.

RATAJ, A. **Cellular IoT in the 5G Era: An Overview.**. IEEE. [S.l.]. 2022.

RFWIRELESS-WORLD.COM. **RF WIRELESS WORLD**. Disponível em: <https://www.rfwireless-world.com/terminology/obd2-frame-format-message-structure>. Acesso em: 01 ago. 2025.

RODRÍGUEZ, Armando; RAÚL, José; INOUYE, Ricardo. Implementation of an OBD-II Diagnostics Tool over CAN-BUS with Arduino, Pinar del Río, 02 Fevereiro 2018.

SAE. **SAE J1979: Diagnostic Test Modes**. SAE INTERNATIONAL. [S.l.]. 2021.

SOMMERVILLE, I. **Engenharia de Software**.

STMICROELECTRONICS. **STM32 Microcontrollers Technical Documentation**. [S.l.]. 2022.

VALDERRÁBANO, J. L. **Embedded Systems Architecture:** Explore architectural concepts and the practical implementation of embedded systems.

VALDERRÁBANO, J. L. **Embedded Systems Architecture:** Explore architectural concepts and the practical implementation of embedded systems.

VICTOR, T. **Safety Benefits of Vehicle Telematics:** A Meta-Analysis.

WIRATAMA. **Design and Implementation of Low-Cost Vehicle Tracking System Using SIM800L GSM/GPRS Module**. IEEE. [S.l.]. 2021.

ZHANG, T. **Urban Traffic Management Based on Vehicle Telemetry Data.**