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

Pedro Alves de Oliveira Neto

Desenvolvimento de um Sistema de Aquisição e Telemetria de Dados para Aplicações Automotivas

Recife
2024

Pedro Alves de Oliveira Neto

Desenvolvimento de um Sistema de Aquisição e Telemetria de Dados para Aplicações Automotivas

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 em Engenharia de Controle e Automação.

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

Recife
2024

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

Oliveira Neto, Pedro Alves de.
Desenvolvimento de um sistema de aquisição e telemetria de dados para aplicações automotivas / Pedro Alves de Oliveira Neto. - Recife, 2024.
79 p. : il., tab.

Orientador(a): Márcio 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, 2024.
Inclui referências, apêndices.

1. Telemetria. 2. MQTT. 3. CAN. 4. Engenharia Automotiva. 5. Aquisição de Dados. I. Brito, Márcio Evaristo da Cruz. (Orientação). II. Título.

620 CDD (22.ed.)

Pedro Alves de Oliveira Neto

Desenvolvimento de um Sistema de Aquisição e Telemetria de Dados para Aplicações Automotivas

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 em Engenharia de Controle e Automação.

Aprovado em: 14/03/2024

BANCA EXAMINADORA

_____

Prof. Dr. Márcio Evaristo da Cruz Brito
Universidade Federal de Pernambuco

_____

M.Sc Néstor Iván Medina Giraldo
Universidade Federal de Pernambuco

_____

Prof. Dr. Geraldo Leite Maia
Universidade Federal de Pernambuco

Dedico este trabalho ao Departamento de Engenharia Elétrica, que proporcionou todo o suporte necessário para minha formação excepcional. Em particular, expresso minha gratidão aos professores Dr. Fabrício Bradaschia e Dr. Márcio Evaristo pela prontidão em me ajudar. À equipe Mangue Baja, pelos valores técnicos e morais que obtive tão cedo, tornando-se minha família na engenharia. À minha noiva Marcella Cavalcanti, por sempre me apoiar e estar do meu lado em conquistas tão significantes. E em especial ao meu avô, Dr. Pedro Alves de Oliveira Filho, por ter me transmitido o valor do conhecimento. Sem sua influência, não teria alcançado tanto.

“O âmago do espírito do homem pede novas experiências.” (Jon Krakauer)

RESUMO

Nos últimos anos, a indústria automotiva tem se destacado pela rápida evolução tecnológica, especialmente no que tange à digitalização e à conectividade dos veículos. A integração de sistemas eletrônicos mais sofisticados tem se tornado um diferencial competitivo para as montadoras, os passos recentes dados em direção à Internet das Coisas e à Inteligência Artificial beneficiam esse ramo marcado pela reinvenção. Esse trabalho tem como objetivo o desenvolvimento de centrais eletrônicas automotivas, as quais serão modulares com o intuito de adquirir parâmetros de todo o veículo e transmitir via telemetria para o servidor, buscando a otimização do projeto automotivo. O trabalho também vai detalhar o dimensionamento dos canais de comunicação *MQTT* assim como os bancos de dados envolvidos, os quais serão essenciais para garantir a integridade dos dados. Além disso, será dissertada a implementação de algoritmos de inteligência artificial de aprendizado por reforço e seu papel na otimização dos sistemas trabalhados através da interação com modelagens matemáticas não-lineares do sistema real, conhecidas por gêmeos digitais.

Palavras-chave: Telemetria; *MQTT*; *CAN*; Engenharia Automotiva; Aquisição de Dados.

ABSTRACT

In recent years, the automotive industry has been distinguished by its rapid technological evolution, especially in terms of digitalization and vehicle connectivity. The integration of more sophisticated electronic systems has become a competitive advantage for manufacturers, with recent strides towards the Internet of Things and Artificial Intelligence benefiting this sector marked by reinvention. This work has the goal to develop automotive electronic central units, that are modulated in order to acquire parameters from all of the vehicle and transmit it through telemetry to the server, seeking the automotive project optimization. The thesis will also detail the implementation of the MQTT communication channels as well as the databases involved, which will be essential to ensure the data integrity. Furthermore, it will be disserted the implementation of artificial intelligence reinforcement learning algorithms and its role in optimizing the systems in question through iterations with non-linear mathematical models of the real system, known as digital twins.

Keywords: Telemetry; MQTT; CAN; Automotive Engineering; Data Acquisition.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	*Antilock Braking System*
API	*Application Programming Interface*
CAN	*Controller Area Network*
CC	Corrente Contínua
DC-DC	*Direct Current to Direct Current*
ECU	*Electronic Control Unit*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
IMU	*Inertial measurement unit*
IoT	*Internet of Things*
I²C	*Inter-Integrated Circuit*
JSON	*JavaScript Object Notation*
LDO	*Low Dropout Regulator*
LED	*Light-emitting Diode*
LIN	*Local Interconnect Network*
LTE	*Long-Term Evolution*
LWT	*Last Will and Testament*
MDP	*Markov's Decision Process*
MOSFET	*Metal Oxide Semiconductor Field Effect Transistor*
MQTT	*Message Queuing Telemetry Transport*
OTA	*Over the Air*
PCI	Placa de Circuito Impresso
QoS	*Quality of Service*
RTOS	*Real-time Operating System*
SAE	*Society of Automotive Engineers*
SMD	*Surface-mount Device*
SPI	*Serial Peripheral Interconnect*
UART	*Universal Asynchronous Receiver Transmitter*
UI	*User Interface*
UX	*User Experience*

SUMÁRIO

1 INTRODUÇÃO

Nesse capítulo, serão introduzidos os conceitos que motivaram e definiram o projeto. Serão abordados os objetivos propostos para solucionar o problema levantado, preparando para uma análise detalhada nas seções subsequentes.

1.1 Delimitação do problema

Alguns projetos de engenharia são caracterizados por um processo evolutivo, onde o aprimoramento ocorre através de sucessivas iterações. Durante estas iterações, os engenheiros se deparam com diversos desafios e problemas que precisam ser resolvidos para alcançar os objetivos de projeto. Esta abordagem é fundamental para o refinamento do projeto e para garantir que o produto final atenda tanto aos requisitos técnicos quanto às expectativas do mercado (ANGHEL et al, 2017).

No entanto, no caso de projetos automotivos, um dos grandes desafios encontrados é o custo elevado associado a cada iteração. Diferentemente de outros sistemas de engenharia, onde simulações digitais e prototipagens rápidas podem ser mais econômicas, no setor automotivo, a complexidade e o custo dos componentes, além dos testes extensivos necessários, tornam cada etapa do processo significativamente mais cara (ANGHEL et al, 2017). Isso pode limitar o número de iterações que podem ser realizadas, exigindo uma eficiência ainda maior na fase de planejamento e execução do projeto.

Outra característica marcante dos sistemas automotivos é a complexidade física e a dificuldade em modelar matematicamente o comportamento do veículo de forma precisa (PETTY, 2018). As equações que representam o sistema em simulações muitas vezes não conseguem representar todos os aspectos do funcionamento real dos veículos. Isso ocorre devido à variedade de interações que ocorrem em um veículo em movimento. Essa lacuna entre a simulação e a realidade aumenta a complexidade dos métodos de otimização utilizados, pois os modelos

podem não refletir totalmente as condições reais de operação (HADSELL, 2019). Além disso, os ambientes nos quais os veículos automotivos operam são extremamente variados e apresentam interações difíceis de prever. As condições de estrada, o clima, o tráfego e as interações com outros veículos e com o ambiente são apenas alguns dos fatores que influenciam a modelagem do sistema (LINNHOFF et al, 2022). Esta variedade de condições torna quase impossível representar todas as interações em uma simulação. Como resultado, o modelo digital acaba sendo uma aproximação que pode não capturar totalmente as nuances do comportamento do veículo no mundo real.

1.2 Justificativa

Este trabalho descreve um sistema de melhoria de projeto capaz de resolver os problemas descritos, por meio de uma captação robusta dos dados envolvidos no protótipo automotivo (FATHIZADEH e AYYAD, 2019). Esse monitoramento detalhado do projeto em operação permite a fácil disponibilização dos dados, fornecendo entendimentos valiosos para o processo de otimização.

O sistema embarcado no veículo permite a coleta de uma vasta gama de dados operacionais, como rotação do motor, velocidade do veículo, temperatura do óleo de motor, aceleração nos eixos longitudinal, lateral e vertical, entre outros (FATHIZADEH e AYYAD, 2019). Estes dados são transmitidos para uma rede de computadores, que possui uma melhor capacidade computacional, onde podem ser processados e analisados. Esta infraestrutura permite o uso de técnicas computacionais avançadas, abrindo caminho para métodos de otimização mais sofisticados e precisos (ALI et al, 2022), tanto para acompanhamento do projetista, quanto para uso de análises automatizadas.

Uma forma de otimização que será analisada é a possibilidade de implementação de algoritmo de aprendizado por reforço, o qual será responsável por otimizar um sistema de gêmeo digital (VOOGD et al, 2023). Ou seja, uma simulação

simples do sistema real será explorada pelo algoritmo, o que servirá de auxílio para otimizar os parâmetros do sistema real ao comparar os resultados reais e digitais.

1.3 Objetivos

A proposta deste Trabalho de Conclusão de Curso (TCC) apresenta uma estrutura de objetivos meticulosamente organizada, que inicia com objetivos gerais, seguida pelos objetivos específicos, culminando em uma discussão sobre a importância desses objetivos e como eles contribuem para a resolução do problema abordado.

O objetivo geral deste trabalho é desenvolver um sistema embarcado automotivo robusto, capaz de captar dados eficientemente em tempo real. Esse sistema é composto por centrais eletrônicas especializadas, configuradas em uma topologia otimizada. Adicionalmente, é proposto de maneira genérica o desenvolvimento do sistema automotivo por meio da incorporação de um gêmeo digital e um algoritmo de aprendizado por reforço, visando a melhoria contínua do veículo através de simulações e ajustes baseados em dados reais. Os objetivos específicos para o projeto são:

- Desenvolvimento de 3 centrais eletrônicas especializadas, a serem implementadas em um protótipo Baja SAE, facilitando assim os testes e a validação do sistema proposto (SAE BRASIL, [s.d.]).
- Implementação de uma central eletrônica de telemetria para a coleta e envio de dados via rede CAN, utilizando tecnologias de IoT e módulos GPRS para conexão à rede de telefonia móvel (ELSAADANY; ALI; HAMOUDA, 2017).
- Desenvolvimento de infraestrutura em servidor para coleta dos dados enviados.
- Desenvolvimento de interface para acompanhamento dos dados recebidos pelo servidor.

- Estrutura para desenvolvimento de um gêmeo digital do veículo para simular seu comportamento e testar diferentes configurações e estratégias de otimização (BATTY, 2018).

A importância desses objetivos reside na capacidade de enfrentar e superar os desafios na engenharia automotiva, especialmente no desenvolvimento de veículos. A utilização de um sistema embarcado robusto para aquisição de dados em tempo real permite uma análise precisa do desempenho do veículo, enquanto a implementação de um gêmeo digital e algoritmos de aprendizado por reforço possibilita uma evolução baseada em iterações rápidas, reduzindo custos e riscos associados a testes em cenários reais.

Essa abordagem não só aprimora significativamente o processo de desenvolvimento de veículos, como também abre novas possibilidades para inovação na engenharia automotiva. Ao integrar esses elementos — dados reais e simulação digital — o trabalho proposto oferece um modelo exemplar para futuras investigações e desenvolvimentos na área, destacando-se pela sua relevância e potencial impacto na indústria automotiva.

1.4 Estrutura do trabalho

A seguir são apresentadas as introduções de cada capítulo do trabalho.

- Capítulo 1 – Introdução: Este capítulo apresenta os conceitos centrais e a motivação do projeto, definindo seus objetivos para solucionar o problema identificado, como prelúdio para uma análise mais aprofundada nos capítulos seguintes.
- Capítulo 2 - Referencial Teórico: Este capítulo aborda os fundamentos essenciais para o desenvolvimento do projeto. Serão detalhados conceitos cruciais para alcançar os objetivos propostos e assegurar a eficácia dos *hardwares* e *softwares* utilizados.

- Capítulo 3 – Desenvolvimento: Este capítulo descreve os procedimentos para desenvolver os sistemas, cobrindo áreas de *hardware*, *firmwares*, *softwares,* prototipagem e fabricação.
- Capítulo 4 – Resultados: Aqui são mensurados e apresentados os resultados obtidos com a implementação do trabalho.
- Capítulo 5 – Conclusões e Considerações Finais.
- Capítulo 6 – Referências.

2 REFERENCIAL TEÓRICO

Nesse capítulo serão tratados os fundamentos que formaram a base para o desenvolvimento do projeto do sistema embarcado, o servidor *MQTT* e o algoritmo de otimização. Serão explicados conceitos importantes para atingir os objetivos descritos e garantir o funcionamento dos *hardwares* e *softwares* envolvidos.

2.1 Sistemas embarcados

Os sistemas computadorizados foram foco de grande desenvolvimento desde o surgimento da microeletrônica, uma vez que, esses sistemas passaram a ocupar espaços menores com uma performance extremamente superior aos da geração passada (SOUZA, [s.d.]). Nesse contexto, na década de 70 foram surgindo os primeiros microcontroladores que tiravam proveito do avanço da computação para executar tarefas específicas, focando o seu processamento na execução de tarefas focadas em um objetivo específico (GARCIA, 2018), ao contrário dos computadores de uso pessoal que buscavam uma generalização e o cumprimento de multitarefas simultâneas.

2.1.1 Definição

É possível definir os sistemas embarcados pela sua principal característica de executar dedicadamente tarefas específicas em dispositivos eletrônicos incorporados (GARCIA, 2018). Tais sistemas são projetados para atender necessidades particulares e são otimizados para isso, em eficiência, desempenho, consumo de energia e geralmente operam em tempo real, ou seja, com resposta sem atrasos perceptíveis.

2.2 Indústria automobilística

Com essa evolução dos sistemas embarcados, os sistemas automotivos que eram constituídos de projetos puramente analógicos eletromecânicos sofreram uma grande evolução (DOS ANJOS, 2011). A introdução de microprocessadores e microcontroladores nesses projetos ao final da década de 70 possibilitou o avanço de sistemas de controle que hoje são considerados críticos no cenário global de um veículo. Inicialmente foram implementados sistemas de controle digital em projetos como a injeção de combustível e ignição, mas que foram evoluídos para uma eletrônica mais sofisticada, passando por freios *ABS*, sistemas de *airbag*, controle de estabilidade e monitoramento de parâmetros (DOS ANJOS, 2011) e resultando em projetos atuais como o uso de câmeras e radares para veículos cada vez mais autônomos (LUNDQUIST, 2011) e internet embarcada para monitoramento individual e de frota em tempo real (GOEL, 2007).

2.2.1 ECUs

Nesse cenário, a crescente complexidade dos sistemas automotivos, devido ao aumento do número de sensores e da demanda por processamento, levou os projetos eletrônicos a buscar a modularização e descentralização do processamento (CONTESINI, 2017).

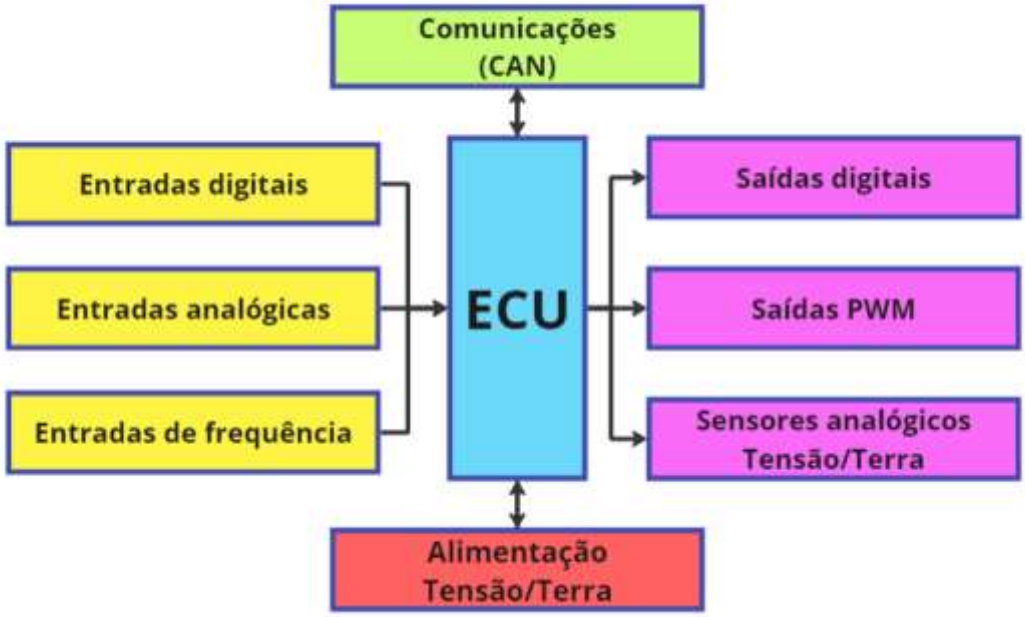
As *ECU*s, ou centrais eletrônicas, são placas com circuitos e microcontroladores que possuem o intuito de cumprir tarefas especializadas em um veículo (CONTESINI, 2017), sendo assim, essencialmente, sistemas embarcados. A convergência da indústria para esse tipo de componente surgiu com o aumento de parâmetros monitoráveis em um veículo e a performance crescente dos microcontroladores, resultando em topologias que facilitam a manutenção do cabeamento de um veículo (FANTON, 2022) ao mesmo tempo em que divide o processamento de diferentes dados em módulos mais especializados e mais críticos quando necessário.

Essas centrais são comumente formadas por um microcontrolador, que processa os dados de interesse para aquela *ECU*, circuitos de condicionamento de sinais, que processam os dados dos sensores para cumprir os requisitos que o microcontrolador suporte, e os componentes necessários para realizar a tarefa na qual a central seja especializada, como os módulos de armazenamento, telemetria, interface, entre outros (CONTESINI, 2017).

Na Figura 01 é possível observar o diagrama de *hardware* genérico de uma *ECU* (WILLIAMS, Samuel V. et al, 2021). Ele consiste de entradas analógicas, digitais e de frequência, as quais representam a função de sensoriamento de uma *ECU*. São essas entradas que recebem dados de sensores dos mais diversos tipos, os quais por muitas vezes possuem circuitos de condicionamento que possibilitam sua aplicação (KAISER, 2014). Ainda, o sistema consiste de saídas digitais, saídas de alimentação dos sensores analógicos, uma vez que esses sensores costumam retornar sua medição nessa faixa de tensão fornecida (HOSTICKA, 2007), saídas *PWM*, que funcionam como saídas analógicas ao variar a largura dos pulsos digitais de alta e baixa tensão em um ciclo específico quando aplicados a dispositivos com inércia ou capacitância (PALACHERLA, 1997).

Por fim, a imagem descreve a alimentação do *hardware*, geralmente executada pela tensão da bateria de 12 V do veículo e regulada para as tensões usuais, como 5 V e 3,3 V, por reguladores como *LDO* ou *DC-DC* (KUECK, 2013). A interface de comunicação padrão para as *ECU*s é a *CAN*, a qual será descrita em seu próprio capítulo. É importante notar que por muitas vezes as centrais possuem comunicações alternativas como *LIN*, *UART*, *I²C*, *SPI* entre outras (VAUSDEVH, 2020).

Figura 1 - Diagrama de *hardware* genérico para uma *ECU*.



Fonte: WILLIAMS, Samuel V. et al. (2021. p. 24-25).

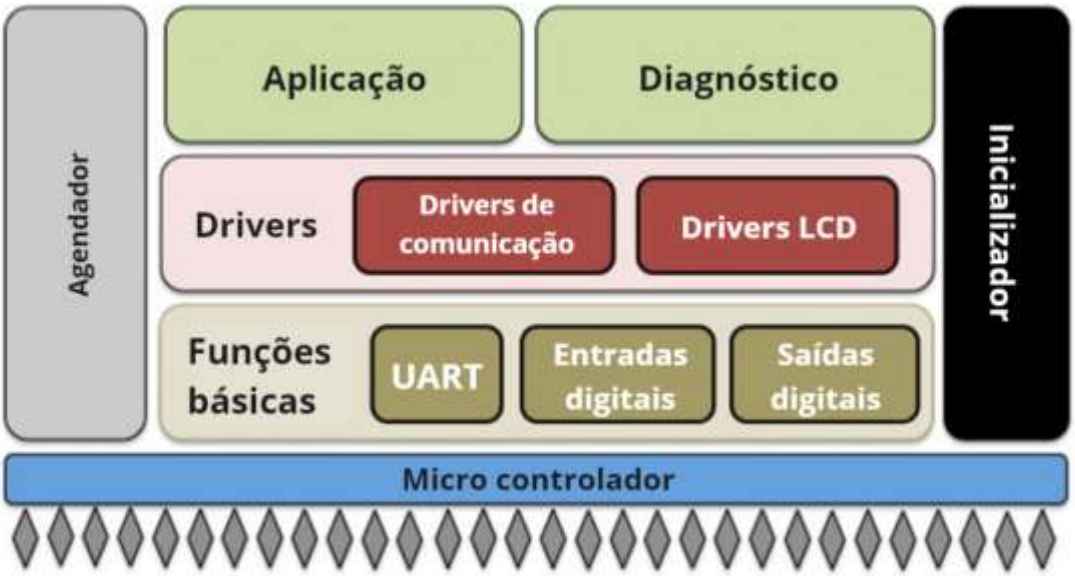
Em seguida, a Figura 02 demonstra uma arquitetura de *software* genérica que define os componentes de *software* utilizados por uma *ECU* (MANSOUR, Karim; FARAG, Wael; ELHELW, Mohamed. 2012). Nessa imagem é possível notar como a estrutura de *software* é construída, partindo da camada de aplicação no topo do diagrama, a qual possui a característica de ser uma camada de alto nível, disponível para o projetista utilizar funções com maiores níveis de abstração, sem preocupação com especificações do microcontrolador.

Em seguida, a camada de *drivers* serve como intermediária, conectando as operações fundamentais do microcontrolador à camada de aplicação. Essa camada é essencial para configurar o *hardware* para funcionar conforme o esperado, detalhando minuciosamente o comportamento específico de cada função.

Por fim, a camada de funções básicas de um microcontrolador engloba o conjunto de operações de baixo nível diretamente relacionadas ao *hardware*, como manipulação de registradores, controle de periféricos e rotinas de interrupção. Essa

camada abstrai a complexidade do *hardware*, fornecendo uma interface simplificada para as camadas superiores, permitindo que o *software* interaja com o microcontrolador de maneira eficiente sem necessitar de um conhecimento profundo sobre os detalhes do *hardware*.

Figura 2 - Arquitetura de *software* genérico para uma *ECU*.



Fonte: MANSOUR, Karim; FARAG, Wael; ELHELW, Mohamed. (2012. p. 1-7).

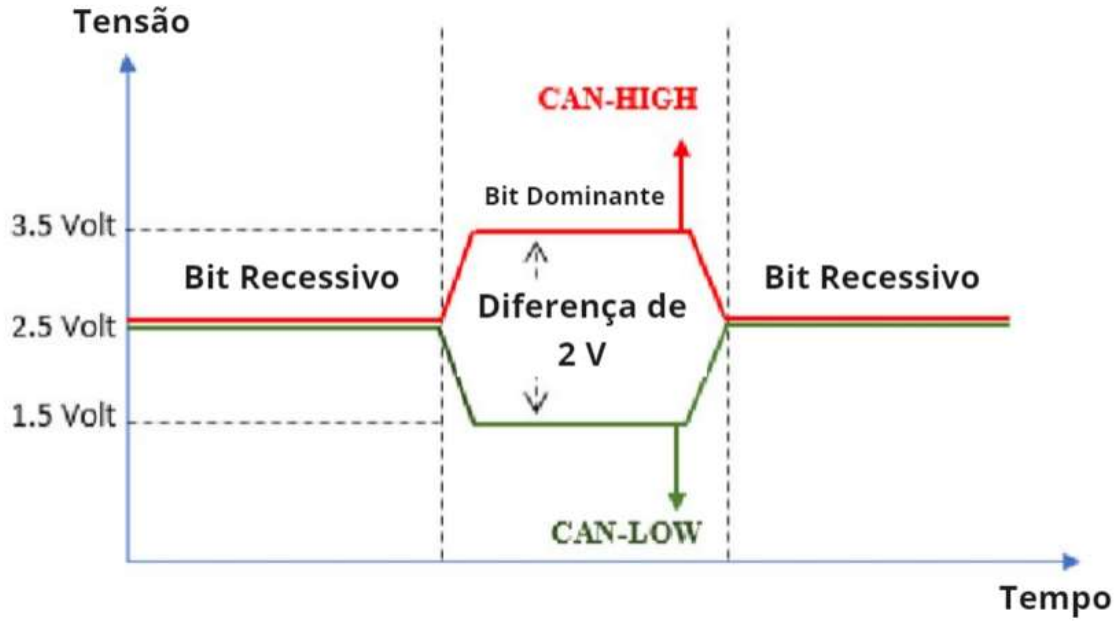
2.2.2 Arquitetura CAN

Como observado, as centrais eletrônicas operam de forma modular e descentralizada. Para viabilizar isso, foi necessário estruturar uma rede de comunicação que atendesse a essas demandas, permitindo a priorização das centrais mais críticas para o funcionamento do veículo e a segurança do condutor. Isso garante a robustez necessária para o bom desempenho em diversos ambientes automotivos, tudo operando em tempo real (JUNIOR, 2023).

Nesse cenário, na década de 80 a empresa alemã Bosch desenvolveu a rede *CAN* atendendo a esses requisitos (JUNIOR, 2023). O funcionamento da rede *CAN* é

implementado em um esquema de transmissão diferencial em pares trançados (MATTEDE, [s.d].), resultando na robustez requisitada para a rede visto que essa aplicação é mais robusta em meios sujeitos a interferências eletromagnéticas (PING et al. 2015), a Figura 03 mostra como são obtidos os bits recessivos e dominantes no funcionamento diferencial (AVATEFIPOUR et al. 2017).

Figura 3 – Identificação de bits em par trançado da rede *CAN*.



Fonte: AVATEFIPOUR, Omid et al. (2017. p. 1-6).

A rede *CAN* utiliza um protocolo orientado a mensagens, em que cada mensagem possui um identificador de 11 ou 29 bits. A rede possui vários modos de operação, podendo chegar a velocidades de transferência de até 10 Mbps na aplicação de *CAN FD.* O funcionamento é executado pelo controlador *CAN*, incumbido de interpretar e enviar as mensagens, em conjunto com o transceptor, que efetua a interface entre o barramento e o controlador (SMITH, 2021).

Uma vez que a rede *CAN* permite a conexão de várias centrais em um único barramento, ela se tornou fundamental na indústria automotiva, sendo regulada pela norma SAE J1939. Para viabilizar essa conexão, a concorrência de mensagens na

rede ocorre com base na prioridade de cada *ECU*, garantindo que as mensagens mais críticas tenham precedência para manter a característica de tempo real da rede (SMITH, 2021).

2.2.3 Software Automotivo

Nessa subseção são avaliadas e selecionadas as ferramentas de desenvolvimento de código embarcado que possibilitam o cumprimento dos objetivos para um sistema robusto em tempo real, como foi visto anteriormente.

2.2.3.1 Microcontrolador

Para estabelecer os requisitos de *software*, configurações, tecnologias e bibliotecas a serem utilizadas, é essencial primeiro definir o microcontrolador, buscando compatibilidade com as características determinadas posteriormente. Nesse sentido, foram avaliados diversos aspectos relevantes para a seleção do microcontrolador. A Tabela 01 apresenta uma matriz de decisão desenvolvida para orientar essa escolha (MARTINS, 2021), identificando os principais critérios considerados: custo, frequência de operação, número de entradas e saídas genéricas (*GPIO*), facilidade de implementação, memória e aplicabilidade em *IoT.*

Considerando que o exemplo de aplicação do projeto atual envolverá 3 centrais em um contexto de menor escala, os principais critérios ponderados para a matriz de decisão foram o custo, devido à sua importância decisiva em projetos menores, e o número de *GPIO*s, que se torna relevante quando um número reduzido de *ECU*s ainda precisa lidar com a implementação de vários sensores. Os outros fatores considerados na matriz são relacionados ao desempenho, ajudando na escolha de microcontroladores similares entre si.

Assim, é possível verificar na tabela que tanto o ESP32 quanto o STM32 atenderam os requisitos. Acessando a documentação de ambos os microcontroladores foi possível notar que o ESP32 possui maior conectividade

(ESPRESSIF SYSTEMS, 2023), o que foi crucial para defini-lo como o microcontrolador utilizado pela *ECU* de telemetria, enquanto o STM32 foi selecionado para as outras duas *ECUs,* viabilizando o custo do projeto e atendendo o número de *GPIOs* necessários (STMicroelectronics, 2023).

Tabela 1 - Matriz de decisão do microcontrolador

	Custo	**Frequência de operação**	**Número de *GPIO***	**Facilidade de implementação**	**Memória**	**Aplicabilidade em *IoT***	**Total**
Pesos	**Peso 5**	**Peso 4**	**Peso 5**	**Peso 3**	**Peso 2**	**Peso 4**	
STM32	5	4	5	4	4	3	98
ATMega328p	3	2	3	5	3	2	73
PIC16	5	2	2	3	2	1	69
ATMega2560	2	3	5	4	4	4	73
ESP32	3	5	4	5	5	5	100

Fonte: O Autor, 2024.

2.2.3.2 Linguagem de programação

Além dos requisitos já citados, outra característica importante dos sistemas embarcados é a utilização de linguagens de programação de baixo nível. Os sistemas embarcados em sua maioria são implementados utilizando a linguagem C (ERIKA, [s.d].), que possui abstração apenas para facilitar o entendimento do projetista, mas que possui um nível próximo do código de máquina, que é a linguagem natural do microcontrolador.

No trabalho atual não será diferente, visto que a linguagem C além de possuir uma grande liberdade do projetista em lidar com o microcontrolador, possui um alto número de usuários, o que aumenta o volume de material para estudo e bibliotecas que auxiliam no desenvolvimento. O *framework*, que é um conjunto de ferramentas e bibliotecas que fornece uma estrutura base para o desenvolvimento do *software*, que possibilitará a integração da linguagem C com os microcontroladores escolhidos é a

IDE do Arduino para a ESP32 e o Mbed para a STM32, ambos possuem uma grande facilidade de implementação considerando a escala trabalhada, possuindo uma grande disponibilidade de documentação.

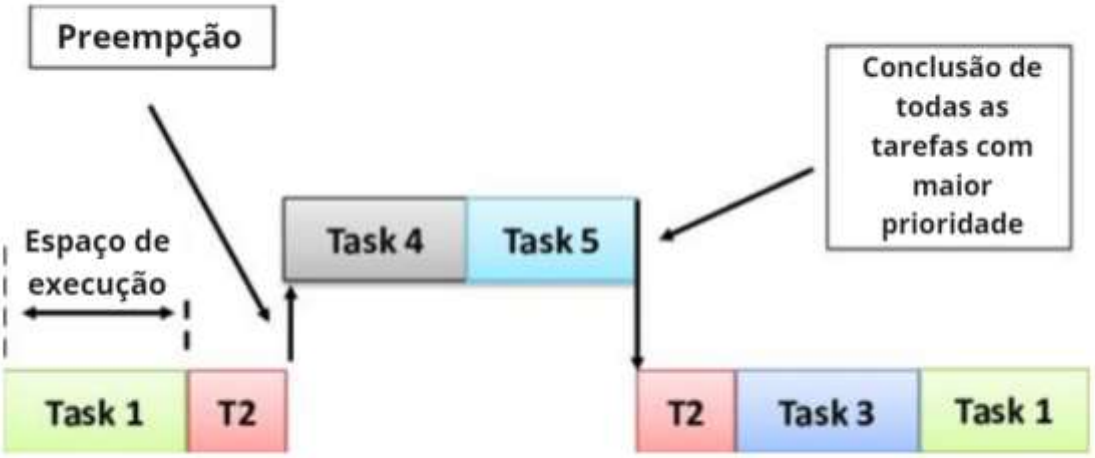
2.2.3.3 Sistema Operacional de Tempo Real

Em aplicações embarcadas que operam em tempo real é indispensável que o código seja implementado em um sistema operacional de tempo real (*RTOS*). Isso se deve ao fato da necessidade de baixos tempos de resposta a ações realizadas pelas *ECUs.*

Um Sistema Operacional em Tempo Real (*RTOS*) se distingue de aplicações convencionais pela sua capacidade de atribuir e gerir prioridades entre as tarefas em execução. Em um *RTOS*, tarefas com alta prioridade são rapidamente atendidas, pois o sistema pode interromper as atividades de menor prioridade em andamento. Esse mecanismo de interrupção e priorização é essencial, pois permite que o *RTOS* reaja de forma eficaz a eventos críticos, redistribuindo recursos de processamento para onde são mais necessários (GILLIS, 2022).

Nos blocos mostrados na Figura 4, é possível observar a execução de várias tarefas. Nesse contexto, torna-se evidente que a tarefa Task 2, ou T2, representada na imagem, tem sua execução interrompida para priorizar as tarefas 4 e 5, sendo retomada logo em seguida (FRIGIERI et al., 2013). Esse funcionamento explica por que as tarefas 4 e 5 têm tempos de execução mais baixos, em detrimento da conclusão da tarefa 2, que é menos prioritária.

Figura 4 – Execução de tarefas prioritárias em um sistema *RTOS*.



Fonte: FRIGIERI, Edielson Prevato et al. (2013)

Nesse sentido, considerando a escolha dos microcontroladores ESP32 e STM32, a escolha do FreeRTOS é lógica por sua portabilidade com os produtos da Espressif, além de possuir grande documentação (ESPRESSIF SYSTEMS, [s.d].). Já para a STM32 foi utilizado o Mbed OS pelos mesmos motivos apontados: portabilidade já definida pela fabricante e grande documentação (ARM, [s.d].).

2.3 Telemetria e Internet das Coisas

A utilização das tecnologias de telecomunicação para monitorar e gerenciar veículos a distância remonta às décadas de 80 e 90, inicialmente usadas em carros de corrida como Fórmula 1, evoluindo significativamente com os avanços em sensores, *GPS*, redes móveis e Internet das Coisas.

A importância da telemetria no setor automotivo é imensa, abrangendo áreas como gerenciamento de frotas, seguros baseados no uso, manutenção preditiva, assistência a veículos autônomos e serviços de conectividade e entretenimento. No contexto deste trabalho, a implementação robusta da telemetria de dados possibilita um monitoramento detalhado do veículo em questão. Os dados são recebidos pelo servidor e interpretados pelos projetistas ou utilizados em dinâmicas de iteração de

projeto de forma a otimizar o veículo em poucas iterações (FATHIZADEH, 2017), sendo extremamente eficiente.

A tecnologia de telemetria considerada para o trabalho é o *GPRS*, que permite a conexão dos dados aquisitados com uma cobertura nacional utilizando um chip de dados móveis comum. O módulo estudado para ser implementado nesse projeto foi o SIM800L (MUSTAFA, Ali; AL-NOUMAN, Mohammed IA; AWAD, Osama A, 2019), que é um módulo bastante difundido em seu uso para projetos de baixa escala com uma grande facilidade de implementação, uma vez que suas bibliotecas possuem um ótimo suporte da comunidade.

2.3.1 MQTT

O protocolo *Message Queuing Telemetry Transport* (*MQTT*) é frequentemente a preferência na telemetria automotiva devido à sua concepção especialmente voltada para cenários com recursos limitados, como largura de banda e bateria, o que é comum em veículos (TRACY, 2016). Sua leveza e eficiência são cruciais para uma transmissão eficaz de dados. Além disso, o *MQTT* oferece baixa latência, possibilitando uma comunicação rápida e eficiente necessária para a transmissão em tempo real de dados do veículo.

Um dos pontos fortes do *MQTT* é sua confiabilidade, oferecendo várias opções de qualidade de serviço (*QoS*) que garantem a entrega de mensagens em diferentes níveis, adaptando-se a várias necessidades e assegurando a comunicação mesmo em condições de rede instáveis. Além disso, a funcionalidade de "*Last Will and Testament*" (*LWT*) do *MQTT* é particularmente útil em telemetria automotiva, pois permite que um veículo comunique um problema ou uma desconexão inesperada, o que é vital para diagnósticos e situações de emergência (HIVEMQ, 2015).

O protocolo também é adaptado para lidar com conexões intermitentes, uma realidade comum quando se trata de veículos em movimento, que podem passar por áreas com cobertura de rede limitada. Sua escalabilidade é outro ponto forte,

permitindo o gerenciamento eficiente de um grande número de dispositivos conectados, o que é essencial à medida que aumenta o número de veículos conectados.

2.3.2 TCU

As Centrais de Telemetria Automotiva, conhecidas como Unidades de Controle de Telemática (*TCU*), surgiram da necessidade de acompanhar a evolução tecnológica na indústria automotiva. Com o avanço dos veículos, equipados com sensores e sistemas eletrônicos complexos, tornou-se crucial ter um sistema capaz de monitorar, controlar e comunicar eficientemente esses dados (LOCONAV, 2022).

A transição para veículos conectados e autônomos contribuiu para a necessidade de *TCUs* avançadas, capazes de gerenciar grandes volumes de dados e comunicar-se com outros sistemas, tais como, infraestrutura de tráfego urbano e dispositivos móveis (LOCONAV, 2022).

As *TCUs* funcionam como o cérebro da telemetria automotiva, coletando e processando dados dos sensores do veículo, realizando comunicação sem fio para enviar e receber dados, além de possibilitar controle remoto e atualizações de *software* através da nuvem (*OTA)*. Essas unidades são vitais para atender às expectativas dos consumidores por serviços conectados dentro do veículo e desempenham um papel central na evolução contínua da indústria automotiva em direção a veículos mais seguros, eficientes e conectados.

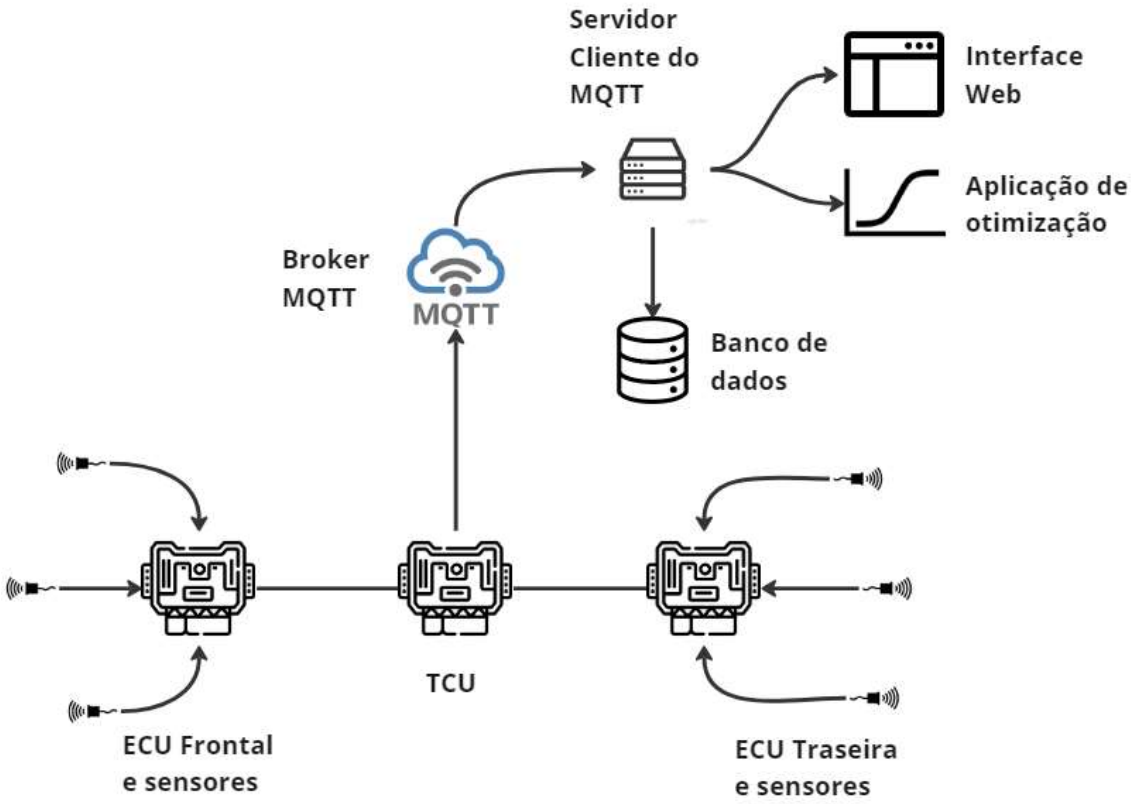
Em nossa aplicação, os componentes essenciais para o cumprimento da *TCU* foram levantados conforme a Tabela 2. Na Figura 5 é possível verificar o diagrama do fluxo que os dados percorrem até serem tratados pelo servidor, assim que ocorre o envio pela *TCU* no *broker MQTT,* a aplicação do servidor recebe os dados e salva no banco de dados, o dado pode ser posteriormente acessado através da interface de usuário pelo projetista, ou por outras aplicações.

Tabela 2 - Componentes selecionados para a *TCU*

Componente	Nome	Função
Módulo de conexão móvel	SIM800L	Conectar ao servidor MQTT para envio dos dados
Módulo *GPS*	NEO-6M	Conexão *GPS* para geolocalização
Transceptor *CAN*	TJA1050	Preparar os dados *CAN* para uso do controlador embutido na ESP32
Microcontrolador	ESP32	Processamento dos dados, executar o código
Fonte chaveada	LM2596	Fornecer a tensão adequada para uma parte dos componentes
Regulador de tensão	LM1117	Fornecer a tensão adequada para outra parte dos componentes

Fonte: O Autor, 2024.

Figura 5 - Fluxo de dados desde a *TCU* até a interface.



Fonte: O Autor, 2024.

Os dados foram estruturados para serem enviados via *MQTT* em pacotes, de modo a atender à frequência de aquisição requerida pela *TCU*, que é de 200 Hz. Esse

valor satisfaz o requisito de frequência de Nyquist de maneira adequada, pois os parâmetros de monitoramento relevantes para uma análise dinâmica do veículo têm periodicidade na ordem dos segundos.

Ao estabelecer o período de envio de mensagens via *MQTT* em 5 segundos, é possível calcular o envio de 1000 pacotes a cada 5 segundos. Na Tabela 3, apresenta-se a descrição do pacote adquirido, indicando quais dados estão sendo obtidos e o tamanho que ocupam em bytes. Dessa forma, um pacote *MQTT* enviado ao servidor terá um tamanho estimado de 32,2 kB.

Tabela 3 - Dados que compõem o pacote transitado pela *CAN* e enviado pelo *MQTT*.

Dado	**Tipo**	**Tamanho**
Aceleração em X	*int16*	2 *Bytes*
Aceleração em Y	*int16*	2 *Bytes*
Aceleração em Z	*int16*	2 *Bytes*
Giroscópio *Pitch*	*int16*	2 *Bytes*
Giroscópio *Roll*	*int16*	2 *Bytes*
Velocidade	*uint16*	2 *Bytes*
Rotação do motor	*uint16*	2 *Bytes*
Temperatura do motor	*uint8*	1 *Byte*
Nível de combustível	*uint16*	2 *Bytes*
Latitude	*double*	8 *Bytes*
Longitude	*double*	8 *Bytes*
	Total:	33 *Bytes*

Fonte: O Autor, 2024.

2.4 Gêmeos digitais e Aprendizado por Reforço

Gêmeos digitais, ou *Digital Twins*, é uma tecnologia emergente que envolve a criação de réplicas digitais de objetos ou sistemas físicos. Estes modelos digitais são

desenvolvidos para simular o comportamento de seus equivalentes no mundo real, utilizando dados coletados de sensores e outras fontes (BATTY, 2018). O conceito de gêmeos digitais tem suas raízes em campos como a modelagem 3D e simulações, mas evoluiu significativamente com o avanço da Internet das Coisas (*IoT*), análise de dados e inteligência artificial.

A principal vantagem dos gêmeos digitais é a capacidade de testar, monitorar, otimizar e prever o comportamento de sistemas físicos sem interferir diretamente neles. No setor automotivo, um gêmeo digital de um veículo pode analisar como diferentes condições de condução afetam o desempenho do carro, permitindo otimizações no design, na manutenção e até na experiência do usuário (VOOGD et al, 2023). Além disso, gêmeos digitais também desempenham um papel importante na área de desenvolvimento de produtos, onde permitem uma iteração mais rápida e eficiente. Ao simular como um produto se comportará sob várias condições, os engenheiros podem fazer ajustes e melhorias antes de criar protótipos físicos, economizando tempo e recursos (XIA et al. 2021).

A combinação de gêmeos digitais com tecnologias avançadas de análise de dados e inteligência artificial abre novas possibilidades para a otimização de sistemas em tempo real, na Figura 6, é possível notar a relação que o sistema real possui com o gêmeo digital, o modelo simulado processa dados de interações que podem encontrar otimizações importantes para o modelo real, enquanto o modelo físico alimenta esse sistemas com dados reais de forma a melhorar os seus resultados (AGNUSDEI, Giulio Paolo; ELIA, Valerio; GNONI, Maria Grazia, 2021).

Figura 6 - Relação entre gêmeo digital e o modelo físico real.

Fonte: AGNUSDEI, Giulio Paolo; ELIA, Valerio; GNONI, Maria Grazia (2021, p. 2767).

O aprendizado por reforço é uma técnica de inteligência artificial amplamente utilizada em sistemas complexos, como os automotivos, onde o foco está em determinar como os agentes devem agir para maximizar uma forma de recompensa cumulativa (LAITANO, 2021). Ao contrário do aprendizado supervisionado, no aprendizado por reforço não há respostas corretas pré-determinadas; em vez disso, um agente aprende por tentativa e erro, interagindo com o ambiente e sendo recompensado por ações bem-sucedidas.

No coração do aprendizado por reforço está o conceito de valor, que é uma estimativa do retorno futuro - ao total de recompensas que um agente espera receber a partir de um estado ou ação específicos. Esse conceito é crucial em sistemas automotivos, onde decisões como a escolha da rota, a otimização do consumo de combustível ou a resposta a condições variáveis de tráfego dependem da maximização desse valor (VOOGD et al. 2023).

Sendo assim, visto que o objetivo do aprendizado por reforço é determinar as ações que maximizam o retorno naquele sistema, são definidos alguns métodos para atingir esse objetivo. O Processo de Decisão de Markov (*MDP*) é um modelo matemático que define que o resultado de uma ação em um determinado estado

depende apenas do estado atual e não do histórico de eventos anteriores (LAITANO, 2021). Ele é caracterizado por um conjunto de estados, um conjunto de ações possíveis, probabilidades de transição entre estados (dadas as ações), e recompensas associadas a essas transições. O objetivo em um *MDP* é encontrar uma política de ações que maximize a soma das recompensas futuras esperadas. Dessa forma, um algoritmo de aprendizado de reforço que encontre essa política de ações deve otimizar o processo em foco. Sendo esse processo um subsistema do gêmeo digital do sistema real, e sendo constantemente comparado com os resultados obtidos no sistema físico, é realista esperar a otimização do processo.

Um gêmeo digital de um carro, por exemplo, pode ser utilizado como um ambiente de teste onde um agente de aprendizado por reforço explora diferentes estratégias de condução ou configurações do sistema para maximizar a eficiência do combustível ou a segurança do veículo. O agente recebe recompensas baseadas no desempenho simulado do carro, aprendendo assim qual abordagem é mais eficaz (VOOGD et al. 2023).

Esta metodologia permite otimizar sistemas automotivos de forma segura e eficiente, sem os riscos associados a experimentos no veículo real. É particularmente útil para testar e desenvolver sistemas avançados de assistência ao motorista (*ADAS*) e tecnologias de veículos autônomos, onde a simulação em um ambiente controlado é essencial antes da implementação prática.

O trabalho atual não tem como objetivo desenvolver o algoritmo de aprendizado por reforço que deve atuar no sistema, visto que há uma complexidade maior que o escopo do projeto, além de ser um algoritmo específico para cada caso de otimização. Ainda assim, será desenvolvida a base em que esses algoritmos irão atuar, ou seja, a estrutura que receberá os dados por telemetria. O ambiente de modelagem dos sistemas em Simulink, plataforma de simulação e modelagem baseada em diagramas de bloco para sistemas dinâmicos, é integrado com Python, uma linguagem de programação de alto nível, amplamente utilizada para desenvolvimento de algoritmos e análise de dados, devido à sua simplicidade e versatilidade, utilizada para a incidência dos algoritmos de aprendizado por reforço. Para a visualização dos

sistemas é utilizado o Unity 3D, uma ferramenta poderosa e flexível para o desenvolvimento de jogos que também é utilizado para criar visualizações de alta qualidade com interação por outros sistemas, que finaliza a implementação de um gêmeo digital com algoritmos de otimização.

3 DESENVOLVIMENTO

Nesse capítulo serão detalhados os passos para o desenvolvimento das centrais eletrônicas envolvidas no projeto, bem como o servidor que recebe os dados enviados por telemetria, além do banco de dados que armazena esses dados de forma a serem disponibilizados com facilidade e robustez. Ainda, é detalhada uma possível aplicação de otimização por gêmeo digital, em que uma simulação de um sistema real será apresentada utilizando tecnologias que permitem uma alta fidelidade na representação do sistema considerado. Dessa forma, será proposta a base para a implementação de um algoritmo de aprendizado por reforço para otimizar esse sistema em comparação com o projeto real.

3.1 Planejamento

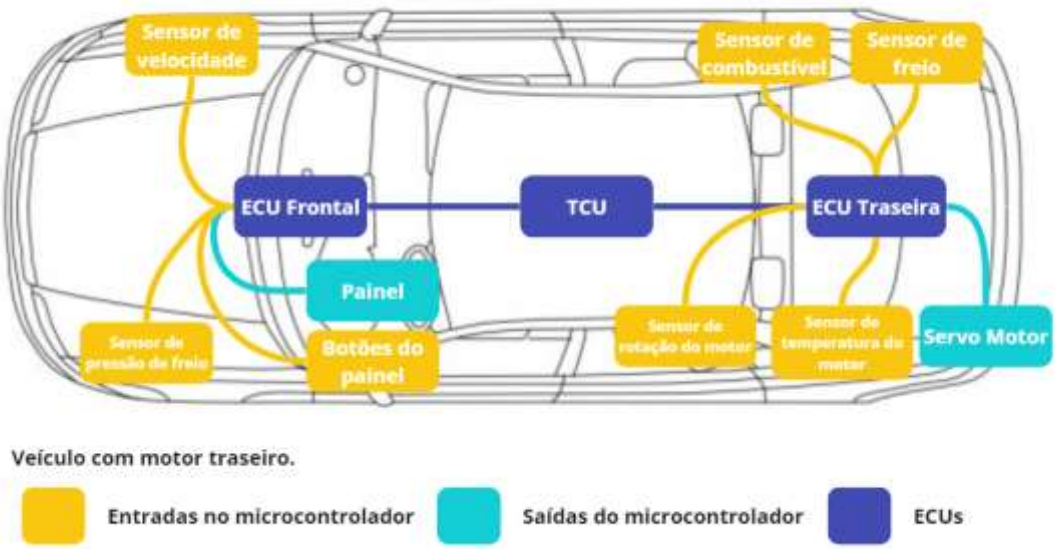
Nessa subseção serão descritas as ferramentas utilizadas para o planejamento do projeto, definindo os objetivos e a forma como eles serão atingidos em cada etapa do seu desenvolvimento.

3.1.1 Topologia do sistema

Na fase de definição do projeto, foi decidido o funcionamento do sistema como descrito no diagrama da Figura 5, já mostrada anteriormente. A interação entre a *TCU* e o servidor *MQTT* foi idealizada de forma que ambos são clientes do mesmo *broker* que transfere os dados. Dessa maneira, o diagrama da Figura 5 foi utilizado no início do projeto para guiar o desenvolvimento, sendo um dos requisitos a ser cumprido. Além disso, foi também definida a topologia para as *ECUs* que serão implementadas no veículo de protótipo, e é nessa fase que é definido o posicionamento de cada central, especificando quais sensores serão responsabilidade de cada *ECU,* bem como o processamento dos dados do mesmo. A Figura 7 mostra a topologia que foi definida no veículo, levando em consideração sua menor escala, possuindo uma central dianteira, a *TCU* posicionada no centro, e uma central traseira. Nas Tabelas 4,

5 e 6 foram descritas as funções de cada uma dessas centrais, e posteriormente os circuitos dessas funcionalidades vão ser detalhados.

Figura 7 - Topologia de rede para as *ECUs*.



Fonte: O Autor, 2024.

Tabela 4 - Funções da *ECU* dianteira.

Funções da *ECU* dianteira
Integração, circuito e processamento do sensor de velocidade
Integração, circuito e processamento do sensor de pressão do freio
Processamento de entradas dos botões do painel
Comunicação com o painel automotivo (*Cluster*)

Fonte: O Autor, 2024.

Tabela 5 - Funções da *TCU*.

Funções da *TCU*
Conexão *GPRS* para envio dos dados de telemetria

Conexão *GPS* para geolocalização

Fonte: O Autor, 2024.

Tabela 6 - Funções da *ECU* traseira.

Funções da *ECU traseira*
Integração, circuito e processamento do sensor de nível de combustível
Integração, circuito e processamento do sensor de freio
Integração, circuito e processamento do sensor de rotação do motor
Integração, circuito e processamento do sensor de temperatura do motor
Operação do servo motor para controle de entrada de ar no motor

Fonte: O Autor, 2024.

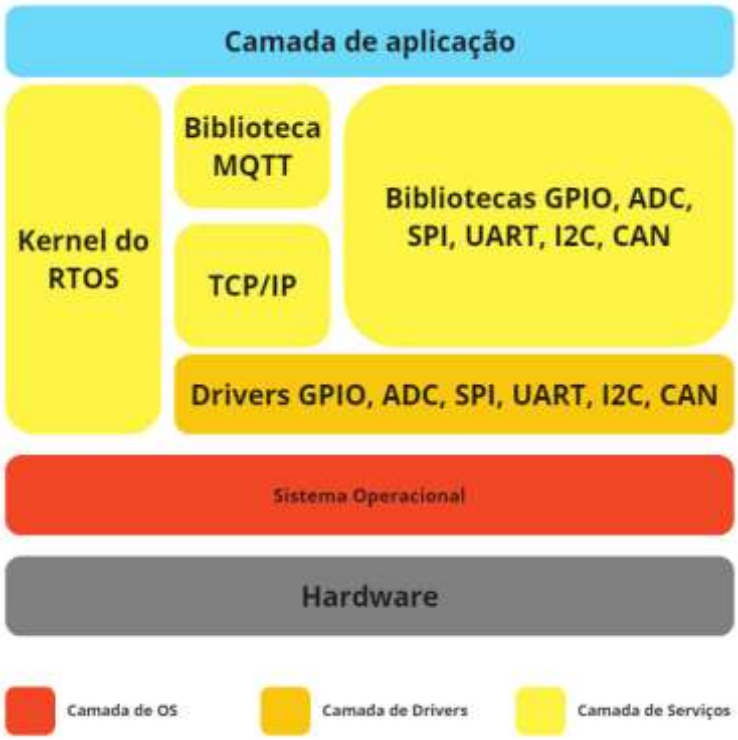
3.1.2 Arquitetura de software embarcado

Como visto na subseção 2.2.1 o projeto do *software* embarcado deve ter sua arquitetura de *software* detalhada. Assim, na etapa inicial, a Figura 8 contém o diagrama de arquitetura de *software*, que foi planejada inicialmente para atender todos os requisitos de uma *ECU* do projeto. Ele oferece uma visão dos principais componentes, sendo assim, foi desenvolvido um diagrama que é suficiente para representar os recursos que serão implementados nas 3 centrais. Na imagem é possível identificar que quanto mais próximo da camada de aplicação, que é onde o código é implementado, maior a abstração do *hardware.* Assim, as camadas foram definidas como:

- Camada de sistema operacional: Onde são gerenciadas as tarefas, temporizações e recursos de sistemas. Aqui são implementadas funcionalidades do FreeRTOS/Mbed OS*,* como tarefas, filas, semáforos e temporizadores.

- Camada de *Drivers*: Oferece uma interface genérica para acesso aos dispositivos periféricos, contendo implementações específicas para controle e comunicação com o *hardware*.
- Camada de serviços: Fornece serviços de alto nível para a aplicação (*middleware*), pode-se destacar que o *Kernel* do *RTOS* possui acesso direto a camada de sistema operacional, visto que o projetista pode controlar como o sistema se comporta em pontos específicos da execução.

Figura 8 – Arquitetura de *software* para as *ECUs* desenvolvidas.



Fonte: O Autor, 2024.

3.1.3 Topologia e protocolo MQTT

Como calculado na subseção 2.3.2, o projeto *MQTT* foi dimensionado para um fluxo de dados de 32,2 kB e a topologia foi definida nessa mesma subseção. Após o dado ser enviado pela *TCU*, um *broker MQTT* recebe esse pacote de dados a cada 5 segundos, tratando em seguida para salvamento em banco de dados.

A tecnologia utilizada para desenvolver o *broker*, que funciona como um canal em que os dados são trafegados, foi o Mosquitto, devido à boa documentação e à facilidade de implementação. Esse *broker* foi implementado utilizando o Node-RED*,* que é uma ferramenta visual de programação em fluxo, sendo uma ferramenta *low code* de fácil edição, possuindo grande praticidade e rapidez na implementação, como visto em VEIGA (2018). Além disso, é necessário definir uma proposta de protocolo para a escalabilidade de um *broker MQTT,* sendo definido os tópicos em que os clientes irão se comunicar. A Tabela 7 cita os tópicos que foram projetados, sendo "*TCU*" o código único de cada central, possibilitando o desenvolvimento em grande escala, tornando esse projeto aplicável em monitoramento de grandes frotas, na aplicação desenvolvida nessa monografia não há grande escala, porém já possui a estrutura para tal. Nessa tabela existe um tópico para que o usuário envie comandos para controlar a *TCU*, e outro tópico para que a *TCU* responda com dados de diagnóstico, respondendo a esse comando. Além disso, foi implementado o tópico de recebimento de dados em representação em fila de bytes, que é utilizado por padrão, evitando o mau uso dos dados fornecidos pela operadora uma vez que só comunica os caracteres essenciais para o entendimento do dado. No entanto, através do tópico de comando, é possível ativar a comunicação via *JSON*, em que outro tópico recebe também esses dados, facilitando a leitura do usuário para diagnosticar eventuais problemas em tempo real.

Tabela 7 - Tópicos *MQTT* e suas funções.

Tópico	**Função**
/*TCU*	Envio de comandos para a *TCU*.
/*TCU/info*	Recebimento de informações de diagnóstico da *TCU*.
/*TCU/data*	Recebimento de dados da *TCU*, em formato de fila de bytes.
/*TCU/user*	Recebimento de dados da *TCU*, em formato *JSON,* para entendimento do usuário.

Fonte: O Autor, 2024.

3.1.4 Banco de dados

O banco de dados é uma parte importante do projeto uma vez que é o ponto de consulta central das aplicações de otimização. De maneira análoga à escolha da tecnologia para o *broker MQTT*, o PostgreSQL foi escolhido como tecnologia de banco de dados, fornecendo uma aplicação fácil e rápida, além de ser conhecido por uma ótima escalabilidade em aplicações de grande porte (ASTERA, 2024).

No PostgreSQL foram definidas colunas conforme os dados que seriam recebidos. Como o banco possui uma coluna referente à data de salvamento do dado por padrão, passou a ser disponível o acesso aos dados de telemetria referente a uma data escolhida. Dessa maneira, é possível que o projetista realize a consulta desses dados referentes ao teste ou validação do sistema projetado de maneira simples, possibilitando a análise de um momento específico do teste/validação, o que serve de grande auxílio no processo de otimização do projeto considerado, uma vez que permite a análise do comportamento obtido e esperado.

3.1.5 Interface de usuário

Uma das aplicações propostas pelo trabalho é a facilitação do acesso aos dados do veículo implementando uma plataforma de monitoramento de dados de fácil acesso. Nesse sentido, a interface de usuário foi desenvolvida pensando em atribuir funcionalidades de design que possibilitam o cumprimento desse objetivo.

Em uma rápida pesquisa de mercado foi possível identificar as tecnologias de maior popularidade para o desenvolvimento de interfaces *HTTP*, que permita o acesso via *browser* dos dados disponibilizados. Assim, as tecnologias implementadas para o cumprimento dessa etapa foram o *framework* React.JS, de desenvolvimento *front-end*, que trabalha utilizando tecnologias bem difundidas, *HTTP, CSS* e JavaScript. Uma vez que a aplicação de interface do usuário deve acessar os dados disponíveis no banco situado na mesma máquina, não foi identificado a necessidade de desenvolvimento de uma *API*, visto que a topologia proposta considera cada máquina

de aplicação como um cliente do *broker MQTT*, permitindo a escalabilidade sem necessidade de um servidor central fornecendo os dados como serviço. Com isso em mente, foram consultadas referências para desenvolvimento de interfaces de monitoramento focando na experiência do usuário e na utilização de artifícios visuais (*UX/UI*). Em ANDERSSON (2013) é possível identificar alguns aspectos que devem conter os *dashboards* de monitoramento, e FARD (2021) e VENDRAMINI (2021) detalham ótimos pontos de melhorias de dashboards clássicos, auxiliando no entendimento da mente do usuário em interação com esse tipo de interface. Por fim, AHMAD (2023) detalha a aplicação automotiva e alguns *feedbacks* valiosos de usuários, assim como os requisitos que esse tipo de *software* deve conter.

Todas as fontes, aliado com as perspectivas da atual monografia, culminaram no cumprimento do design avaliado nas Figuras 9, 10 e 11, sendo as funcionalidades detalhadas na Tabela 8.

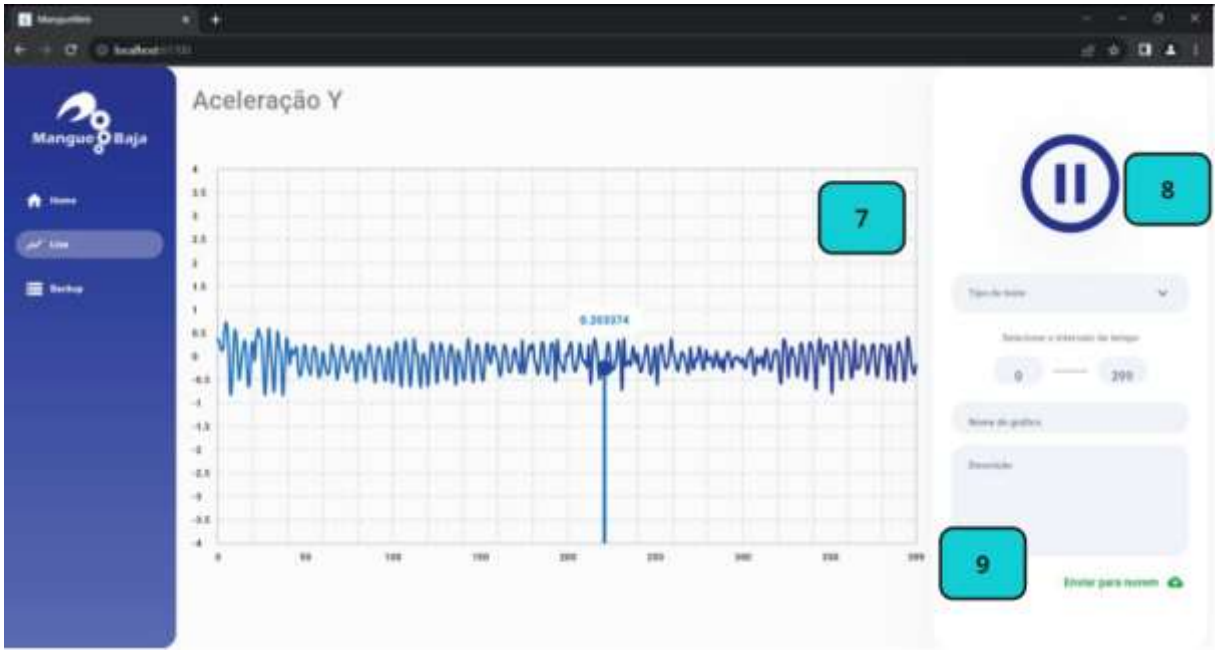
Figura 9 - Design da interface de usuário 1.

Fonte: O Autor, 2024.

Figura 10 - Design da interface de usuário 2.

Fonte: O Autor, 2024.

Figura 11 - Design da interface de usuário 3.



Fonte: O Autor, 2024.

Tabela 8 - Detalhamento das funcionalidades de interface.

Funcionalidade	Detalhamento

1	Acesso aos dados em tempo real
2	Acesso aos dados em nuvem (Google Drive)
3	Acompanhamento dos dados em tempo real (Gráficos)
4	Acompanhamento dos dados em tempo real (Números)
5	Localização no mapa
6	Dados de *Roll* e *Pitch* expressos de forma intuitiva
7	Gráfico com maior detalhamento
8	Opção de pausar o gráfico para capturar momento importante
9	Opção de salvamento de intervalo dos dados para análise posterior

Fonte: O Autor, 2024.

3.2 Desenvolvimento do *hardware*

Essa subseção irá detalhar o desenvolvimento do *hardware* desde o protótipo até a proposta de fabricação em larga escala. Os circuitos são inicialmente dimensionados, assim como, a seleção detalhada dos componentes e o projeto da placa de circuitos impressos.

3.2.1 Circuitos de aquisição e esquemático

Essa seção exibe todos os circuitos desenvolvidos para que sejam aquisitados os parâmetros levantados no projeto. Todos os circuitos descritos são mostrados nos esquemáticos das Figuras 29 e 30, localizadas no Apêndice D.

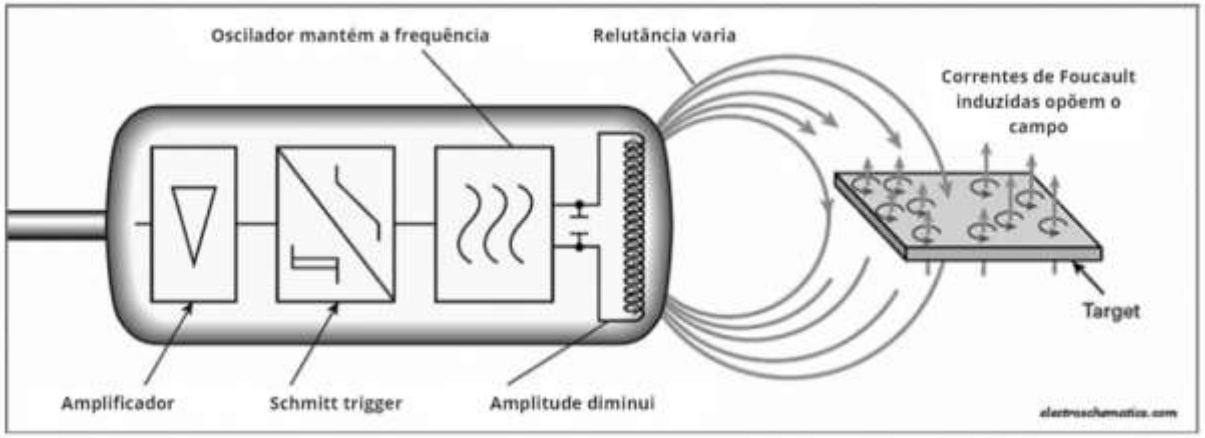
3.2.1.1 ECU Dianteira

Aqui são descritos todos os circuitos implementados na *ECU* dianteira.

3.2.1.1.1 Sensor indutivo de velocidade

Dada a sua proximidade com o pneu dianteiro, onde a velocidade de um veículo de tração traseira é mais precisamente medida, pois o giro da roda geralmente reflete diretamente o movimento real do carro, sem ser afetado por derrapagens, o processamento dos dados de velocidade é atribuição da *ECU* dianteira. Nesse sentido, foi utilizado um sensor de funcionamento indutivo, uma vez que esse tipo de sensor possui a capacidade de detecção de metais ao verificar uma variação na sua relutância pela variação de fluxo magnético que o sensor induz em objetos colocados à sua frente (ROBU.IN, 2020). Conforme ilustrado na Figura 12, o sensor indutivo opera através da indução de uma onda alternada de frequência fixa, cuja amplitude varia ao detectar um objeto metálico. Esse processo ocorre quando o objeto metálico induz uma corrente no metal, gerando um campo magnético de direção oposta. Posteriormente, um circuito Schmitt trigger identifica essa alteração na amplitude, permitindo a detecção de objetos metálicos.

Figura 12 - Funcionamento de um sensor indutivo.



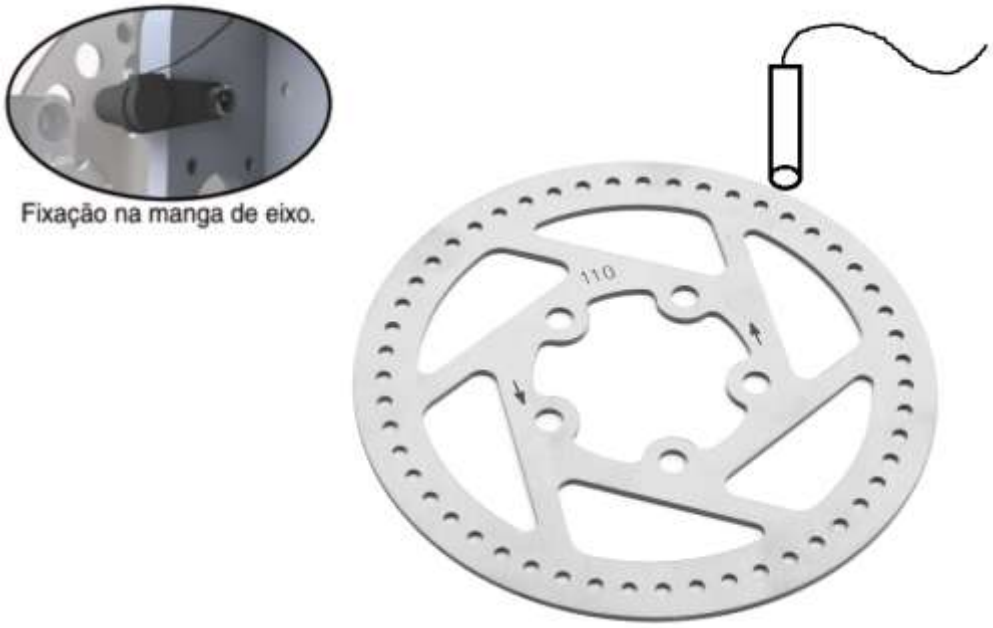
Fonte: ROBU.IN (2020).

Assim, na aplicação automotiva, em especial em motocicletas, é comum que o disco de freio possua furos de alívio de massa igualmente distanciados. Visto que o disco gira em conjunto com a roda do veículo, é possível utilizar um sensor indutivo que obtenha vantagem na detecção desses furos, uma vez que o sensor é capaz de

identificar as porções de metal ao longo do disco, o processo detalhado é mostrado na Figura 13.

O sensor é fixo na manga de eixo, que é um componente fixo em relação ao corpo do veículo (possui mesma velocidade que o veículo em relação a um observador externo). Assim, o sensor aliado ao microcontrolador serve como um contador de furos ao longo do tempo, com essa informação, é possível dividir essa contagem de furos por tempo pela quantidade de furos no disco, obtendo-se o número de voltas da roda por tempo, ou seja, a velocidade angular da roda, que aliado ao valor do raio do centro até os furos resulta na velocidade linear da roda, que possui o mesmo valor da velocidade do veículo.

Figura 13 - Detecção de furos no disco de freio pelo sensor indutivo.



Fonte: O Autor, 2024.

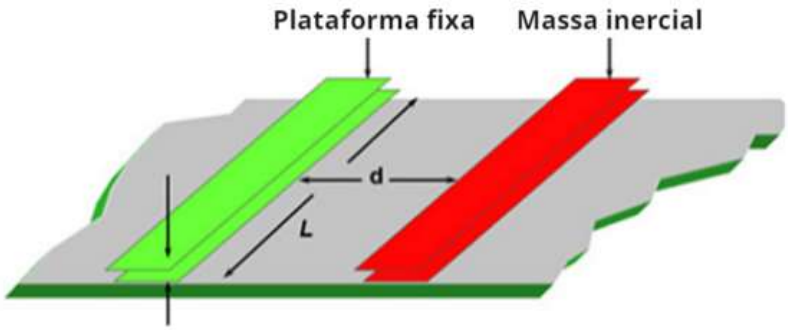
Para utilizar o sensor indutivo com o microcontrolador, foi implementado um circuito para apenas alterar a tensão de nível lógico do sensor que muitas vezes atua em 12 V. Para isso, é utilizado um divisor de tensão simples.

3.2.1.1.2 Acelerômetro e Giroscópio

Na *ECU* dianteira foi integrado internamente um sensor *IMU* que possui a função de adquirir a aceleração tridimensional do veículo bem como o ângulo que o veículo estiver em determinado momento.

O funcionamento da *IMU* é melhor detalhado em BAKER (2018), em que é possível verificar a existência de unidades inerciais e molas que variam um valor de capacitância capaz de ser medido pelos circuitos da unidade inercial, definido pelo posicionamento mecânico das placas fixas no semicondutor em relação às placas variantes da unidade inercial, como visto na Figura 14.

Figura 14 - Posição entre placas que definem a capacitância medida.



Fonte: BAKER, Bonnie (2018).

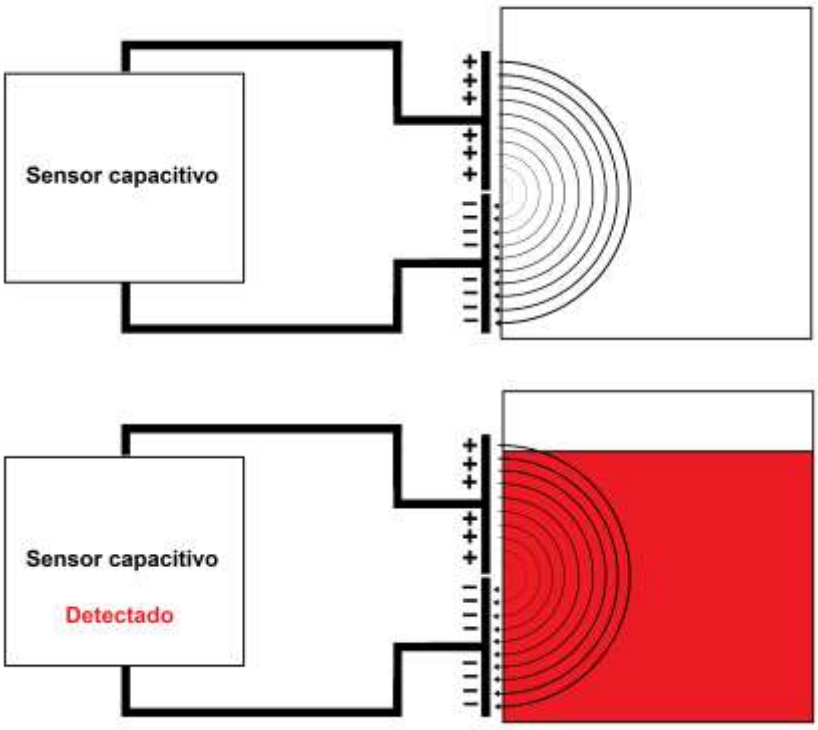
3.2.1.2 ECU traseira

Aqui são descritos todos os circuitos implementados na *ECU* traseira.

3.2.1.2.1 Sensor capacitivo de nível de combustível

De maneira análoga ao circuito mostrado para o sensor indutivo, o sensor capacitivo também explora a mudança do meio colocado à sua frente para detectar um objeto. Como pode ser visto na Figura 15, as placas do capacitor são colocadas de forma que o meio dielétrico é o ambiente externo, ao qual o sensor está apontado, dessa maneira, materiais de diferentes constantes dielétricas irão resultar em uma alteração na capacitância medida pelo dispositivo.

Figura 15 - Posição entre placas que definem a capacitância medida.



Fonte: EQUIPE MOUNTAINBAJA (2018).

Assim, visto que a gasolina em um tanque de combustível possui constante dielétrica diferente do ar, é possível detectar se o tanque atingiu um valor crítico ajustando o sensor para tal medição. O circuito implementado para fazer a leitura desse sensor é também análogo ao circuito do sensor indutivo, necessitando de um divisor de tensão para o sinal gerado pelo sensor, alterando o nível lógico de 12 V para 3,3 V.

3.2.1.2.2 Circuito de medição de rotação do motor

O funcionamento de um motor a combustão depende da inserção de faísca pela vela automotiva para que o combustível entre em combustão e expanda o pistão que resulta no movimento do motor. Assim, a vela automotiva gera um sinal elétrico que representa uma tensão elevada e funciona de maneira cíclica, representando cada rotação executada pelo motor.

Com isso em contexto, basta tratar esse sinal gerado pela vela para que haja a medição de rotação. O circuito para tal utiliza um opto-acoplador que, como descrito em TUNG ([s.d].), funciona como forma de isolar o circuito que gera o sinal com o circuito de aquisição. Logo, o sinal periódico faz com que um *LED* interno ao componente acenda conforme o número de rotações, o que gera um sinal digital com o nível de tensão correto para o microcontrolador implementar um contador de rotações por tempo.

3.2.2 Componentes

A definição de componentes em um projeto automotivo costuma seguir padrões de qualidade avaliados pelas montadoras diretamente com os fornecedores, em que são validadas etapas importantes de certificação de qualidade em uma fábrica de componentes. Nesse projeto, a Tabela 9 cita os componentes envolvidos nas centrais, a tarefa que eles executam e o motivo que definiu o critério de seleção deles entre outras opções.

Tabela 9 - Critério de seleção para os componentes das centrais.

Descrição	Nome	Critério de seleção
Acelerômetro e giroscópio	MPU-6050	Boa disponibilidade no mercado, bom custo e bom suporte de bibliotecas
Módulo *GPS*	NEO 6M	Bom suporte de implementação, disponibilidade de bibliotecas, bom custo em relação a outros módulos

Módulo *GPRS*	SIM800L	Bom suporte de implementação, disponibilidade de bibliotecas, boa disponibilidade de mercado
Opto-acoplador	4N25	Utilizado em todas as aplicações de referência, bom custo e disponibilidade de mercado
Termistor	MTE-3018	Bom custo
Fonte chaveada	LM2596	Boa disponibilidade no mercado, atende os requisitos de tensão e corrente e bom custo
Conector 24 vias	Sicma 24	Possui classificação IP67, garantindo boa robustez
Transceptor *CAN*	TJA1050	Boa disponibilidade no mercado, utilizado na maioria das aplicações pesquisadas, atende os requisitos do sistema
Regulador de tensão	LM1117	Boa disponibilidade no mercado, utilizado na maioria das aplicações pesquisadas, atende os requisitos do sistema

Fonte: O Autor, 2024.

3.2.3 Design da PCI

O projeto de uma placa de circuitos impressos automotiva segue requisitos de robustez tão críticos quanto o seu funcionamento para a segurança de um veículo. Nesse sentido, essa seção segue com uma definição do que foi implementado para as placas dessas centrais.

O *software* utilizado para o projeto das placas foi o KiCad*,* uma vez que ele possui código aberto e um bom suporte da comunidade. Nele foram definidos tanto os esquemáticos quanto o design da placa de circuitos. Como visto em JONES (2004), uma boa configuração padrão para o design de PCIs que pode ser aplicada em diversos projetos assegurando uma robustez são os valores de 25 milésimos de polegada para trilhas de sinais, 50 milésimos de polegada para trilhas de potência, 15 milésimos de polegada de distância para outras trilhas e componentes. Ainda, JONES (2004) define o tamanho do diâmetro da ilha de soldagem de um componente de furo passante em 1,8 vezes o tamanho do diâmetro do furo.

Todas essas configurações foram definidas no KiCad e assim os componentes foram colocados da maneira que JONES (2004) também descreve, os componentes

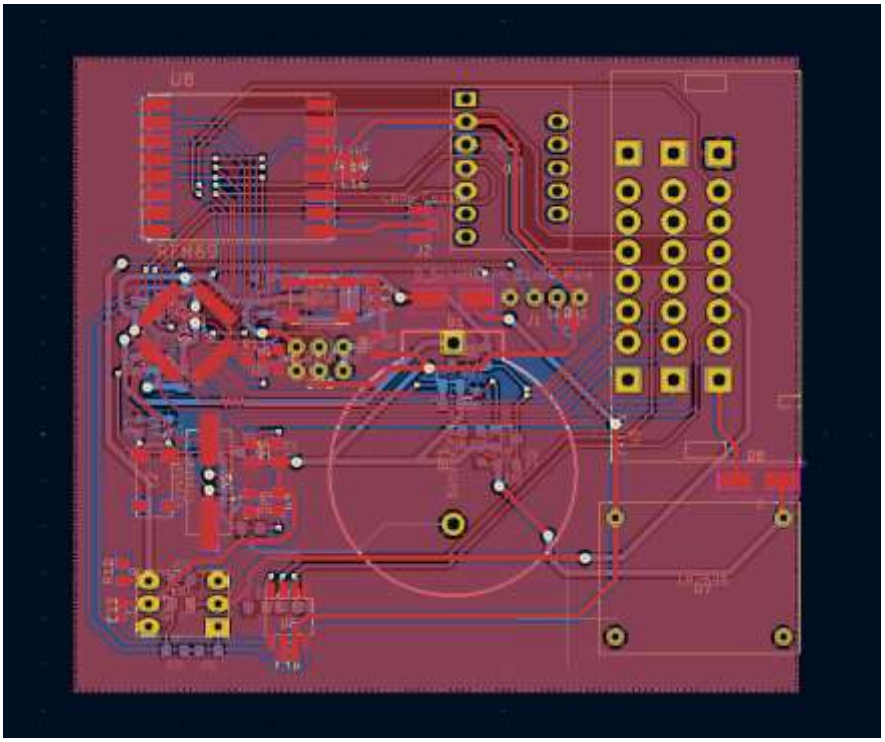
foram então roteados manualmente, estando atento para as trilhas de maiores frequências que poderiam induzir efeitos eletromagnéticos indesejados. Nesse projeto, os circuitos implementados não resultaram em maiores preocupações visto que não apresentam tensões ou frequências expressivas, ou seja, acima de 100 V, ou em altas frequências, especialmente aquelas superiores a 1 GHz, que poderiam levar a emissões eletromagnéticas problemáticas em outros componentes eletrônicos devido ao acoplamento indutivo ou capacitivo JONES (2004).

Além disso, durante o projeto foi notado que as centrais dianteira e traseira possuem circuitos extremamente semelhantes, com microcontroladores iguais e diferenciando apenas nos circuitos. Uma vez que podemos tratar os circuitos como módulos isolados, o projeto realizado uniu ambos os esquemáticos em uma única placa de circuito impresso, visando melhor custo de produção, o que compensa na aplicação de menor escala.

O projeto passou por uma fase inicial de validação, na qual foram empregadas placas em sua versão de desenvolvimento para prototipagem das centrais, como será descrito detalhadamente no capítulo de prototipagem. Este estágio de montagem em placas de circuito indica que o projeto está validado. Buscando aumentar a robustez e desenvolver placas mais sofisticadas e compactas, optou-se por utilizar componentes de montagem de superfície (*SMD*), utilizando diretamente o chip integrado dos microcontroladores.

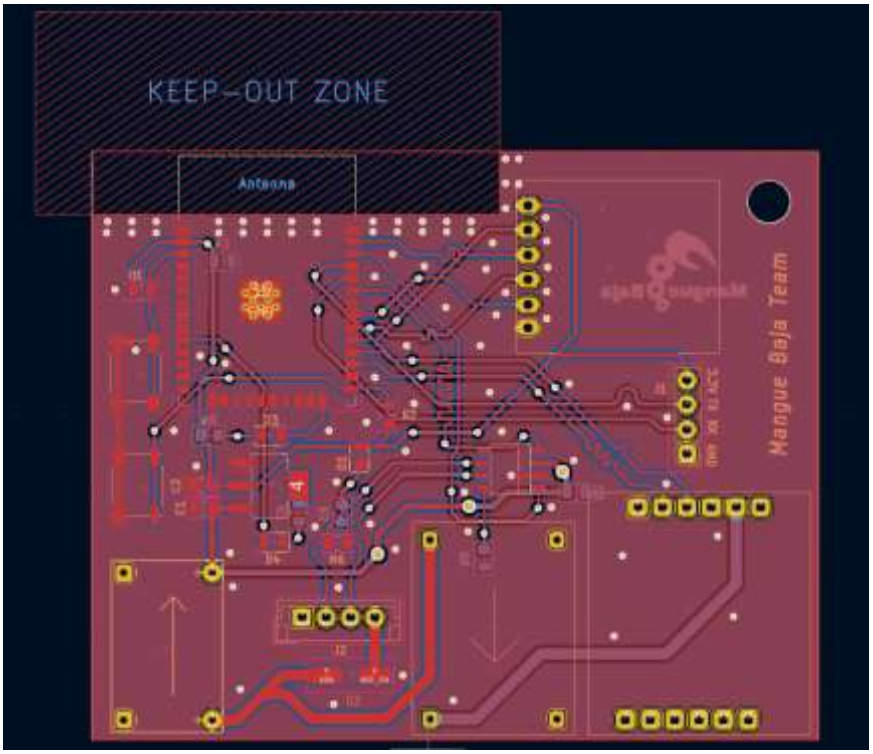
É possível verificar nas Figuras 16 e 17 ambos os resultados de projeto das placas, sendo possível notar tanto a implementação em *SMD* das placas de desenvolvimento quanto a integração dos circuitos dianteiros e traseiros para um melhor custo, soldando os componentes por demanda em estágios mais avançados do projeto.

Figura 16 - Projeto final PCI dianteira e traseira.



Fonte: O Autor, 2024.

Figura 17 - Projeto final PCI da *TCU*.



Fonte: O Autor, 2024.

3.2.4 Prototipagem

Para permitir a iteração e desenvolvimento dos projetos, é essencial realizar a etapa de prototipagem. Isso permite validar as soluções de forma flexível, possibilitando testar diferentes parâmetros com facilidade, sem custos adicionais ou aumento do tempo de desenvolvimento. Dessa forma, todos os circuitos mencionados foram iterados várias vezes para verificar os resultados de diferentes soluções, até alcançar o resultado pretendido nesta monografia. Após o planejamento e o projeto, os protótipos são inicialmente montados em placas de prototipagem, como ilustrado na Figura 18, onde um sensor *IMU* passa pelo processo de prototipagem com o auxílio de um Arduino.

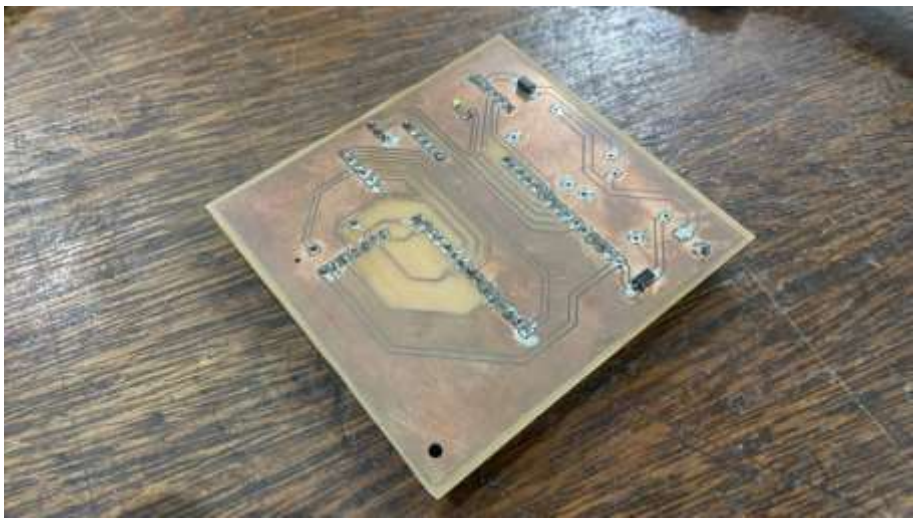
Figura 18 - Prototipação do *IMU*, validação e calibração do sensor.

Fonte: O Autor, 2024.

Para validar o projeto da placa de circuito impresso, após as iterações que definem os circuitos finais e os códigos que atendem aos requisitos, são inicialmente realizados testes utilizando um método de produção manual da PCI. Isso envolve o uso de técnicas que utilizam corrosão com percloreto de ferro em placas de cobre e fenolite, com o objetivo de constatar o bom funcionamento de todas as partes validadas.

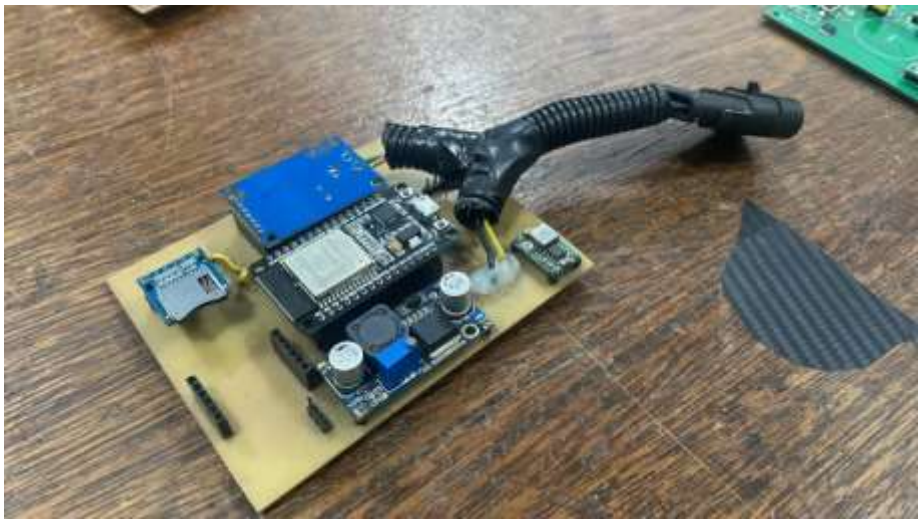
Nesse sentido, a Figura 19 mostra o resultado obtido pelo método manual de corrosão, onde são identificadas algumas falhas estruturais, mas que cumprem com a validação do projeto integrado, além de ser iterado de maneira rápida, em apenas algumas horas. Na Figura 20 os componentes estão montados em conjunto com a placa de desenvolvimento, validando assim o *software* que será utilizado na versão final.

Figura 19 - Placa por fabricação manual de corrosão.



Fonte: O Autor, 2024.

Figura 20 - Montagem dos circuitos na placa de teste.

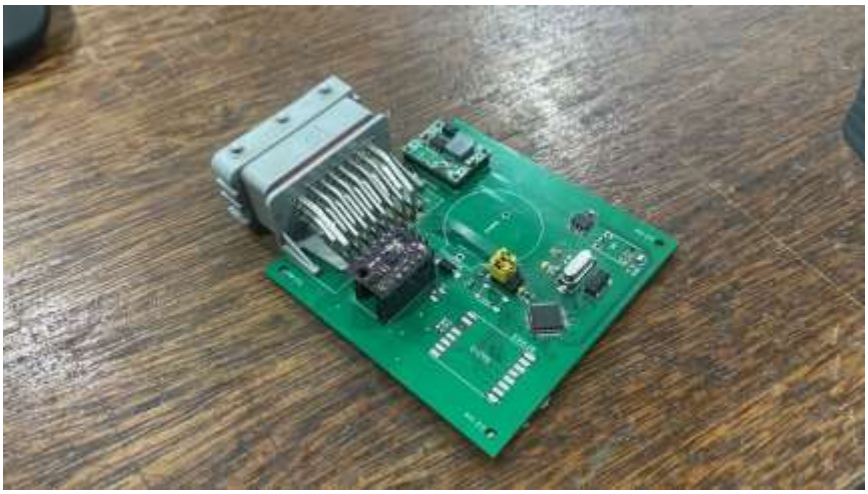


Fonte: O Autor, 2024.

3.2.5 Fabricação

Após a validação dos protótipos na subseção anterior, os componentes são encaminhados para fabricação em empresas especializadas, resultando no *hardware* final do projeto. As Figuras 21, 22 e 23 apresentam o resultado das placas em sua forma final, com as placas dianteira e traseira já testadas, enquanto a *TCU* ainda está em processo de envio pela fabricante. Isso ressalta a eficácia da validação durante a fase de prototipagem.

Figura 21 - Montagem em *hardware* final da *ECU* dianteira.

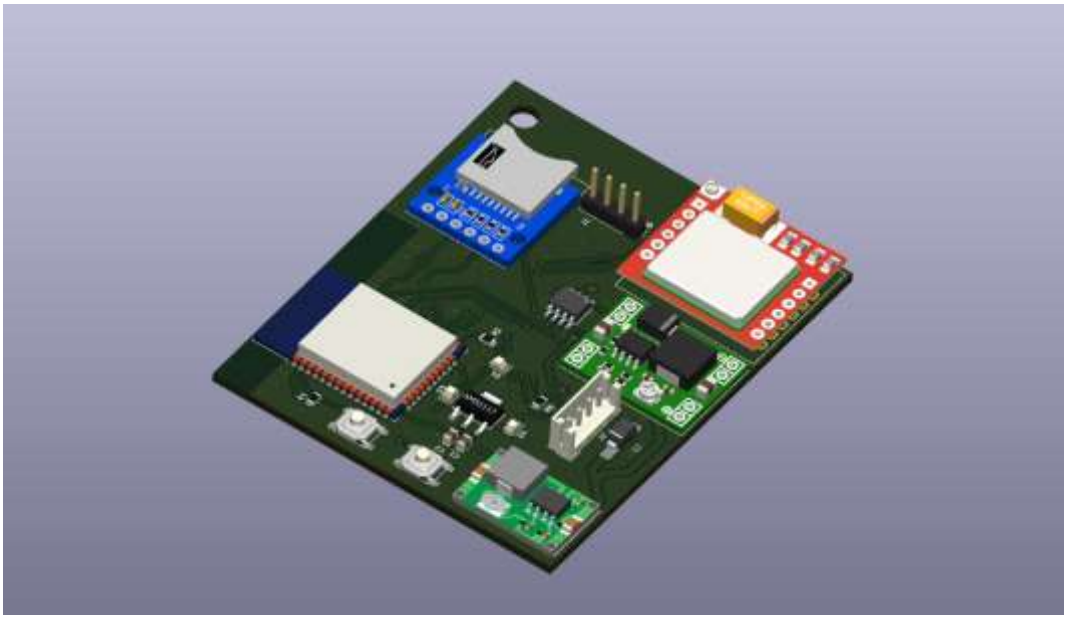


Fonte: O Autor, 2024.

Figura 22 - Montagem em *hardware* final da *ECU* traseira.

Fonte: O Autor, 2024.

Figura 23 - Montagem em *hardware* final da *TCU*.



Fonte: O Autor, 2024.

3.3 Desenvolvimento do *software* embarcado

Nessa etapa serão descritos os métodos empregados para atingir os objetivos para o *software* embarcado. Aqui será caracterizado como a programação das centrais foi executada, desde o ambiente de desenvolvimento até o código.

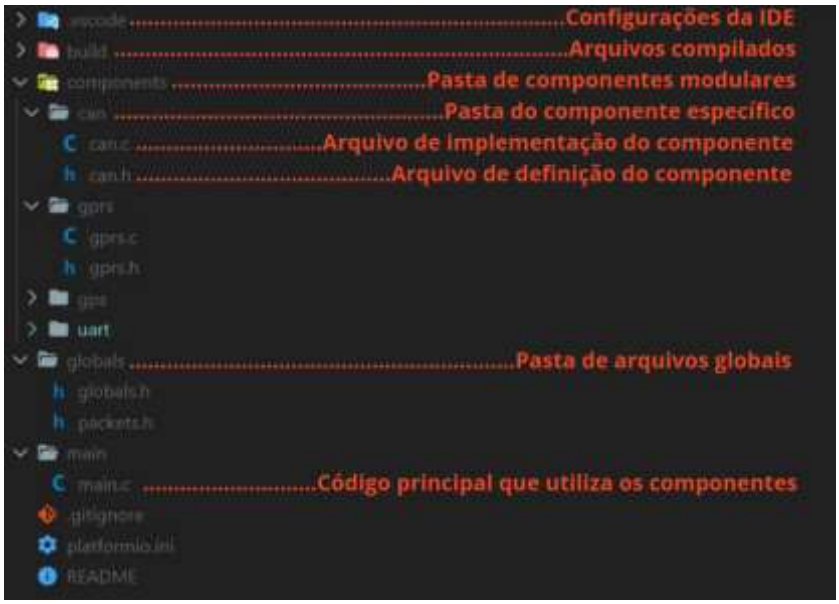
3.3.1 Ambiente de desenvolvimento

HUNT, Andrew; THOMAS, David (1999) definem o programador pragmático como aquele que emprega uma série de boas práticas, as quais proporcionam diversas vantagens, incluindo a capacidade de escalar bem o código. Com esse propósito, eles estabeleceram práticas que promovem a modularização do código, garantindo que o desenvolvimento de novas funcionalidades não resulte em improdutividade devido à complexidade alcançada.

Com isso em mente, o ambiente de desenvolvimento de *software* automotivo precisa ser capaz de receber novas iterações com novas funcionalidades de maneira fácil e organizada. Assim, a estrutura de pastas do projeto foi definida e caracterizada

como mostrado na Figura 24, em que é detalhada a motivação por trás dessa estrutura em cada pasta definida.

Figura 24 - Motivações para a organização de pastas do projeto.



Fonte: O Autor, 2024.

A *IDE* escolhida para desenvolver o código das centrais foi o VS Code, uma vez que permite a adição de extensões que facilitam o desenvolvimento. Tanto a STM32 quanto a ESP32 possuem suporte pela extensão do PlatformIO, que facilita a integração da *IDE* com os *frameworks* de desenvolvimento dessas placas, o Arduino para a ESP32 e o Mbed OS para a STM32.

3.3.2 Implementação do código

Ainda seguindo os conceitos de HUNT, Andrew; THOMAS, David; (1999) para a implementação de um código escalável, os componentes modulares que foram implementados conforme a estrutura da seção anterior devem possuir comportamento independente de outras seções de código que não são relacionados a esses componentes. Isso significa que o arquivo de implementação de um componente deve conter apenas a função de configuração para que seja iniciado o componente, a

função de *call-back* que será chamada de maneira periódica pelo *RTOS* e outras funções que auxiliem o entendimento do componente e o seu funcionamento.

A relação entre componentes, como a obtenção do dado via *CAN* para envio no *MQTT* por exemplo, deve ser definida então no código principal, ou main.c, dessa maneira, qualquer componente que seja adicionado em um momento futuro ao desenvolvimento, e que deve lidar com componentes já utilizados pelo código, não irá causar problemas pelo “efeito espaguete” (HUGO, 2017), em que o código assume um fluxo de controle intrincado, tornando a compreensão e manutenção extremamente difíceis.

No Apêndice A foram colocados exemplos da função main.c e can.c que demonstram a modularidade do componente e como a função principal pode utilizar de maneira a focar na aplicação, sem a necessidade de explicitamente definir suas configurações.

3.3.3 Depuração

Todo o código desenvolvido foi depurado utilizando uma porta *Universal asynchronous receiver-transmitter* (*UART),* que permite utilizar funções como o *print* para obter valores da execução em tempo real. Além disso, foi utilizado um *hardware* próprio para depuração, permitindo a pausa na execução e a consulta de variáveis e registradores. Para a ESP32 foi utilizado o *JTAG,* e a STM32 possui o ST-Link como interface de programação e de *debug* como padrão.

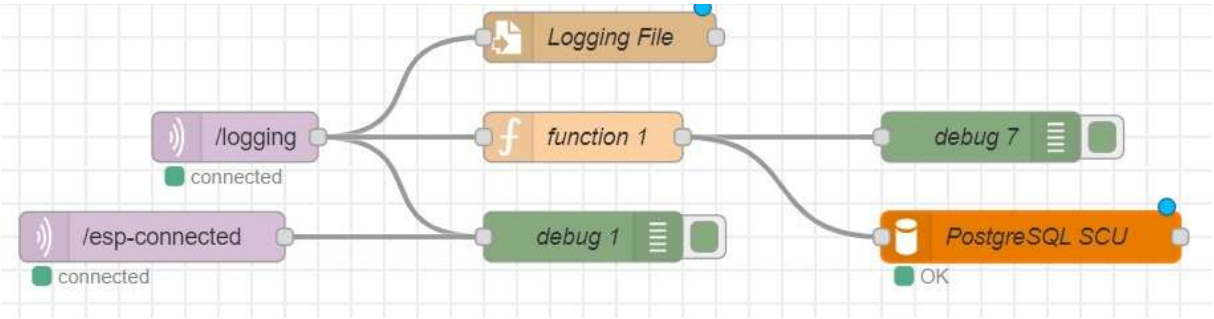
3.4 Desenvolvimento do *software* em nuvem

Aqui serão detalhados os passos tomados para implementar o *software* responsável por receber os dados no servidor, permitindo a consulta e a otimização por meio de algoritmos. Além disso, será detalhado o processo de desenvolvimento do gêmeo digital, bem como a implementação da base do algoritmo de aprendizado por reforço.

3.4.1 Broker MQTT

O *broker MQTT* foi implementado utilizando o *software* Mosquitto, que permite a criação de um túnel de mensagens *MQTT* em um servidor qualquer. Dessa forma, uma aplicação em Node-Red, que já possui bibliotecas para conexão com esse tipo de servidor, recebe os dados enviados nesse túnel e assim pode realizar o salvamento em um banco de dados PostgreSQL, utilizando também bibliotecas já definidas para essa conexão. O fluxograma Node-Red foi definido na Figura 25. Dessa forma, os dados são submetidos a um processo de tratamento para que os bytes sejam convertidos em seus tipos correspondentes e armazenados em colunas da tabela de dados definida no banco. Esse procedimento disponibiliza os dados de um momento específico do teste de forma acessível, permitindo que outras aplicações realizem consultas no banco de dados com facilidade.

Figura 25 - Fluxo Node-Red para o servidor *MQTT*.



Fonte: O Autor, 2024.

3.4.2 Interface gráfica

Como citado anteriormente, foi utilizado o *framework* de JavaScript, React.JS, para o desenvolvimento da interface gráfica mostrada anteriormente, o qual se relaciona com os gráficos analisados e gerados em Python para fácil consulta do projetista.

Essa aplicação, cujo código-fonte da página da Figura 9 está disponível no Apêndice B, permite que os projetistas ajam rapidamente ao avaliar os dados de telemetria recebidos em tempo real, o que facilita a tomada de decisões ao longo das iterações do projeto.

3.4.3 Gêmeo digital e Aprendizado por Reforço

Os avanços recentes nas tecnologias de inteligência artificial e telemetria possibilitaram os avanços descritos nessa seção, sendo proposto um processo de otimização que pode ser aplicado com os avanços decorridos nessa monografia.

3.4.3.1 Simulink e Python

Uma característica fundamental de um gêmeo digital é o quão bem sua simulação representa o sistema real, pois quanto mais precisa essa relação, mais significativos são os resultados obtidos pela análise do gêmeo digital. Portanto, uma dificuldade na implementação de algoritmos de otimização reside na complexidade de alguns sistemas de engenharia, uma vez que muitos desses algoritmos requerem uma simulação linear baseada em equações simplificadas.

Assim, considerando que o ambiente do MATLAB/Simulink é mais utilizado em sistemas mais complexos uma vez que permite a implementação em diagrama de blocos, foi realizada uma interface de comunicação entre essas simulações mais precisas com o enorme suporte que o Python possui em avanços de inteligência artificial, e consequentemente, algoritmos de otimização.

Dessa forma, utilizando a biblioteca “matlab.engine*”,* um ambiente pode ser definido no Simulink em arquivo .slx, e esse arquivo é iniciado pelo algoritmo em Python, o qual pode interagir com o ambiente de simulação configurando os seus parâmetros e rodando as simulações. Essa relação abre oportunidades para

implementações de algoritmos que interajam com o ambiente em Simulink, o que possibilita a implementação de algoritmos de aprendizado por reforço em bibliotecas como o PyTorch que possuem elevada maturidade e desenvolvimento da comunidade nesse sentido.

O Apêndice C contém o código que executa essa relação e ainda comunica via *TCP* com a ferramenta Unity 3D utilizando a biblioteca “unity_comms”, permitindo a visualização de maneira completa dos resultados obtidos otimizando a simulação. Esse código foi realizado utilizando o exemplo de uma simulação de motor CC, uma vez que a simulação completa de um veículo é um objetivo futuro de continuação desse trabalho.

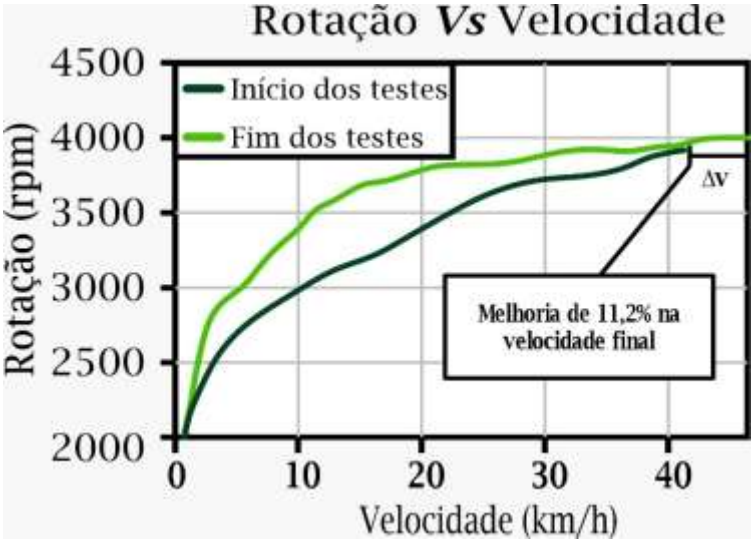
Por fim, a exploração do sistema de simulação por parte do algoritmo de aprendizado por reforço pode utilizar o banco de dados desenvolvido nesse trabalho com o intuito de comparar os resultados em simulação com os resultados do mundo real, trabalho que foi implementado em VOOGD et al. (2023) utilizando a otimização de uma rede neural de veículos autônomos em relação com o gêmeo digital otimizado utilizando aprendizado por reforço.

4 RESULTADOS

O projeto desenvolvido atingiu os objetivos de obtenção e disponibilidade de dados de telemetria em sistemas automotivos. Na aplicação proposta, o acesso de maneira rápida aos dados de projeto contribuiu na validação do desenvolvimento do sistema de transmissão, uma vez que permitiu a iteração no projeto de maneira rápida, atingindo uma melhoria de 11,2% na velocidade final do veículo, como visto na Figura 26.

O teste consistiu de diversas acelerações em pista reta para se obter os dados de rotação, aceleração e velocidade no percurso. O sistema desenvolvido foi utilizado de forma a monitorar em tempo real a dinâmica do veículo nesses testes, ajustando os parâmetros do seu sistema de tração conforme os resultados foram acompanhados pelos projetistas mecânicos. Assim, foi possível atingir os resultados comparando com a dinâmica ideal do gráfico de rotação por velocidade, e em cada iteração o gráfico ficou mais satisfatório. Na Figura 27 é mostrado o teste do veículo em tempo real com a interface de acompanhamento desenvolvida.

Figura 26 - Resultado de melhoria na velocidade final na aplicação do projeto desenvolvido.



Fonte: O Autor, 2024.

Figura 27 – Teste do veículo acompanhado pela interface desenvolvida.

Fonte: O Autor, 2024.

Foi possível verificar a aquisição de mais de 200 mil pacotes de dados enviados e salvos em banco em aproximadamente 20 minutos de validação. Essa análise e monitoramento eficientes confirmam o valor do projeto, validando a teoria discutida na etapa de planejamento.

5 CONCLUSÕES E CONSIDERAÇÕES FINAIS

O projeto concluído nessa monografia demonstra a utilidade dos dados no processo de iteração de projeto em engenharia, e como o monitoramento minucioso e robusto permite o desenvolvimento de análises que acrescentam valor aos resultados.

A estrutura desenvolvida desde a aquisição de dados até a aplicação de algoritmos em simulações mais elaboradas representa um avanço significativo na otimização de sistemas, resultando em maior eficiência no desenvolvimento de projetos e na redução dos custos de iteração nos sistemas automotivos.

5.1 Desenvolvimentos futuros

Ao final do trabalho foi possível identificar oportunidades de desenvolvimentos futuros em campos que não foram detalhados até o momento.

A implementação desse projeto em simulações automotivas complexas é uma das evoluções que permite tanto a validação do que foi proposto em cenários mais desafiantes, como possui grande potencial de evoluir esses projetos automotivos através de comparação com os dados obtidos em tempo real pelo sistema de aquisição e telemetria. É de interesse do autor o melhor detalhamento de como essa comparação entre os cenários real e digital pode ocorrer de forma a obter melhores resultados por parte dos sistemas automotivos, considerando que o projeto virtual pode atingir resultados ótimos de maneira mais eficaz do que iterações físicas do produto.

A proposta de utilização de aprendizado por reforço para atuação nos gêmeos digitais de forma a atingir resultados otimizados também é um foco que vai ser desenvolvido em pesquisas futuras, uma vez que, sendo validada a obtenção de parâmetros ótimos por uso de inteligência artificial, os sistemas reais poderiam se beneficiar do uso dos resultados que o aprendizado por reforço obteria com rapidez nessas simulações, fechando o ciclo mostrado na Figura 6 ao permitir a comunicação de parâmetros úteis entre ambos os sistemas virtual e digital

6 REFERÊNCIAS

AGNUSDEI, Giulio Paolo; ELIA, Valerio; GNONI, Maria Grazia. Is digital twin technology supporting safety management? A bibliometric and systematic review. Applied Sciences, v. 11, n. 6, p. 2767, 2021.

AHMAD, Sadat. Designing a detailed telemetry dashboard for sim-racers. 2023. Trabalho de Conclusão de Curso. University of Twente.

ALI, Mohammed Hasan et al. Big data analysis and cloud computing for smart transportation system integration. Multimedia Tools and Applications, p. 1-18, 2022.

ANDERSSON, Vincent; HESSLUND, Martin. Web-based Real-time Information Dashboard-An event-driven approach to visualize telemetry data. 2013.

ANGHEL, Daniel-Constantin et al. Study of iterations in the design process of a product for automotive industry. In: MATEC Web of Conferences. EDP Sciences, 2017. p. 09011.

ARM. ST | Mbed. [s.d]. Disponível em: <https://os.mbed.com/teams/ST/>. Acesso em: 3 fev. 2024

ASTERA. PostgreSQL vs SQLite: o confronto final do banco de dados. ASTERA, 2024. Disponível em: https://www.astera.com/pt/knowledge-center/postgresql-vs-sqlite/. Acesso em: 3 fev. 2024

AVATEFIPOUR, Omid et al. Linking received packet to the transmitter through physical-fingerprinting of controller area network. In: 2017 IEEE workshop on information forensics and security (WIFS). IEEE, 2017. p. 1-6.

BAKER, Bonnie. Apply Sensor Fusion to Accelerometers and Gyroscopes. DigiKey, 2018. Disponível em: https://www.digikey.com/en/articles/apply-sensor-fusion-to-accelerometers-and-gyroscopes. Acesso em: 3 fev. 2024

BATTY, Michael. Digital twins. Environment and Planning B: Urban Analytics and City Science, v. 45, n. 5, p. 817-820, 2018.

CONTESINI, Leonardo. Afinal o que é ECU? Como elas funcionam?. FlatOut Brasil, 2017. Disponível em: https://flatout.com.br/afinal-o-que-e-ecu-como-elas-funcionam/. Acesso em: 2 fev. 2024

CORRÊA, Gustavo; FUKUSHIMA, William; MATSUMOTO, Fernando. Aprendizado por Reforço #3 — Processo de Decisão de Markov (Parte 2). Medium, 2020. Disponível em: https://medium.com/turing-talks/aprendizado-por-refor%C3%A7o-3-processo-de-decis%C3%A3o-de-markov-parte-2-15fe4e2a4950. Acesso em: 3 fev. 2024

DOS ANJOS, Eduardo Giovannetti Pereira. A evolução da eletrônica embarcada na indústria automobilística brasileira. 2011.

ELSAADANY, Mahmoud; ALI, Abdelmohsen; HAMOUDA, Walaa. Cellular LTE-A technologies for the future Internet-of-Things: Physical layer features and challenges. IEEE Communications Surveys & Tutorials, v. 19, n. 4, p. 2544-2572, 2017.

EQUIPE MOUNTAINBAJA. Sensor Capacitivo com Arduino. Vida de Silício, 2018. Disponível em: https://portal.vidadesilicio.com.br/sensor-capacitivo/. Acesso em: 3 fev. 2024

ERIKA. A quick guide to embedded programming languages. DeepSea Developments, [s.d]. Disponível em: https://www.deepseadev.com/en/blog/embedded-programming-languages/. Acesso em: 3 fev. 2024

ESPRESSIF SYSTEMS (Xangai). ESP32 Series Datasheet. [S. l.: s. n.], 2023

ESPRESSIF SYSTEMS (Xangai). FreeRTOS Overview. ESPRESSIF, [s.d]. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html. Acesso em: 3 fev. 2024

FANTON, Darek. What is CAN bus and why is it so important?. OnLogic Blog, 2022. Disponível em: https://www.onlogic.com/company/io-hub/what-is-can-bus/. Acesso em: 2 fev. 2024

FARD, Adam. From Good To Great In Dashboard Design: Research, Decluttering And Data Viz. Smashing Magazine, 2021. Disponível em: https://www.smashingmagazine.com/2021/11/dashboard-design-research-decluttering-data-viz/. Acesso em: 3 fev. 2024

FATHIZADEH, Masoud; AYYAD, Anan. Application of Remote Telemetry for Improving Formula SAE Car Performance. In: Transactions on Engineering Technologies: World Congress on Engineering and Computer Science 2017. Springer Singapore, 2019. p. 229-243.

FRIGIERI, Edielson Prevato et al. Embedded Micro Application Server in Intel Mini-ITX DN2800MT for Interaction with the ARM Cortex-M3. 2013.

GARCIA, Fernando D. Introdução aos sistemas embarcados e microcontroladores. Embarcados, 2018. Disponível em: <https://embarcados.com.br/sistemas-embarcados-e-microcontroladores/>. Acesso em: 1 Ago 2023.

GILLIS, Alexander S. Real-time operating system (RTOS). TechTarget, 2022. Disponível em: https://www.techtarget.com/searchdatacenter/definition/real-time-operating-system#:~:text=Often%2C%20embedded%20systems%20are%20used,within%20a%20predictable%20time%20limit. Acesso em: 3 fev. 2024

GOEL, Asvin. Fleet Telematics: Real-time management and planning of commercial vehicle operations. Springer Science & Business Media, 2007.

HADSELL, Raia. Challenges for Deep Reinforcement Learning in Complex Environments. YouTube, 2019. Disponível em: https://www.youtube.com/watch?v=fuBXPmJ_fl4

HIVEMQ. What is MQTT Last Will and Testament (LWT)? – MQTT Essentials: Part 9. HiveMQ, 2015. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/. Acesso em: 3 fev. 2024

HOSTICKA, Bedrich J. Analog circuits for sensors. In: ESSCIRC 2007-33rd European Solid-State Circuits Conference. IEEE, 2007. p. 97-102.

HUGO, Kewerson. Clean Code Matters!. TriadWorks, 2017. Disponível em: http://blog.triadworks.com.br/clean-code-matters. Acesso em: 3 fev. 2024

HUNT, Andrew; THOMAS, David. The Pragmatic Programmer: From Journeyman to Master. 1. ed. [S. l.]. Addison-Wesley Professional, 1999.

JONES, David L. PCB Design Tutorial. [S. l.: s. n.]. 2004

JUNIOR, Adair Montagna. Protocolo CAN: Visão Geral e Aplicações. LRI Automação Industrial, 2023. Disponível em: https://blog.lri.com.br/0-protocolo-can-visao-geral-e-aplicacoes/. Acesso em: 3 fev. 2024

KAISER, Martin. Electronic control unit (ECU). In: Gasoline Engine Management: Systems and Components. Wiesbaden: Springer Fachmedien Wiesbaden, 2014. p. 254-259.

KUECK, Christian. The Right Power Supply Can Simplify Automotive ECU Design. Analog Devices, 2013. Disponível em: https://www.analog.com/en/resources/technical-articles/the-right-power-supply-can-simplify-automotive-ecu-design.html. Acesso em: 2 fev. 2024

LAITANO, Joao. Conceitos Para Entender Reinforcement Learning. Medium, 2021. Disponível em: https://medium.com/datarisk-io/conceitos-para-enteder-reinforcement-learning-2b04d68bf4f9. Acesso em: 3 fev. 2024

LINNHOFF, Clemens et al. Measuring the influence of environmental conditions on automotive lidar sensors. Sensors, v. 22, n. 14, p. 5266, 2022.

LOCONAV. The What, Why, How of a Telematics Control Unit. LocoNav, 2022. Disponível em: https://loconav.com/blog/telematics-control-unit/. Acesso em: 3 fev. 2024

LUNDQUIST, Christian. Sensor fusion for automotive applications. 2011. Tese de Doutorado. Linköping University Electronic Press.

MANSOUR, Karim; FARAG, Wael; ELHELW, Mohamed. AiroDiag: A sophisticated tool that diagnoses and updates vehicles software over air. In: 2012 IEEE International Electric Vehicle Conference. IEEE, 2012. p. 1-7.

MARTINS, Julia. Sete passos simples e rápidos para criar uma matriz de decisões, com exemplos. Asana, 2021. Disponível em: https://asana.com/pt/resources/decision-matrix-examples. Acesso em: 3 fev. 2024

MATTEDE, Henrique. Como seu carro se comunica? Rede CAN e OBD2!. Mundo da Elétrica, [s.d]. Disponível em: https://www.mundodaeletrica.com/como-seu-carro-se-comunica-rede-can-obd2/. Acesso em: 3 fev. 2024

MUSTAFA, Ali; AL-NOUMAN, Mohammed IA; AWAD, Osama A. A Smart real-time tracking system using GSM/GPRS technologies. In: 2019 First International Conference of Computer and Applied Sciences (CAS). IEEE, 2019. p. 169-174.

PALACHERLA, Amar. Using PWM to generate analog output. Microchip Inc. Application Note, AN538, 1997.

PETTY, Mikel D. Modeling and validation challenges for Complex Systems. In: Engineering Emergence. CRC Press, 2018. p. 199-216.

PING, Bibo et al. Research on electromagnetic interference between power cables and shielded twisted-pair bundles. In: 2015 IEEE 6th International Symposium on Microwave, Antenna, Propagation, and EMC Technologies (MAPE). IEEE, 2015. p. 409-414.

ROBU.IN. Inductive Proximity Sensor Working Principle. ROBU.IN, 2020. Disponível em: https://robu.in/inductive-proximity-sensor-working-principle/. Acesso em: 3 fev. 2024

ROSSI, Henrique. Arquitetura de Software em Sistemas Embarcados. Embarcados, 2015. Disponível em: https://embarcados.com.br/arquitetura-de-software-em-sistemas-embarcados/. Acesso em: 1 ago. 2023.

SAE BRASIL. Baja Nacional. Disponível em: https://saebrasil.org.br/programas-estudantis/baja-sae-brasil/. Acesso em: 1 fev. 2024.

SANTOS, Guilherme. Protocolo MQTT: O Que é, Como Funciona e Vantagens. Automação Industrial. 20 de maio de 2022. Disponível em: https://www.automacaoindustrial.info/mqtt/ Acesso em: 1 fev. 2024.

SMITH, Grant Maloy. O que é barramento CAN (Controller Area Network) e como ele se compara a outras redes de barramento de veículos. DEWESoft, 2021. Disponível em: https://dewesoft.com/pt/blog/que-e-barramento-can. Acesso em: 3 fev. 2024

SOLANKI, Vijender Kumar; DHALL, Rohit. An IoT based predictive connected car maintenance approach. 2017.

SOUZA, Thiago. História e Evolução dos Computadores. Toda Matéria, [s.d.]. Disponível em: https://www.todamateria.com.br/historia-e-evolucao-dos-computadores/. Acesso em: 2 fev. 2024

STMicroelectronics. STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs. [S. l.: s. n.], 2023

TRACY, Phillip. MQTT protocol minimizes network bandwidth for the internet of things. RCR Wireless News, 2016. Disponível em: https://www.rcrwireless.com/20161129/featured/mqtt-internet-of-things-tag31-tag99. Acesso em: 3 fev. 2024

TUNG, Megan. What is an Optocoupler and How it Works. Jameco, [s.d]. Disponível em: https://www.jameco.com/Jameco/workshop/Howitworks/what-is-an-optocoupler-and-how-it-works.html Acesso em: 3 fev. 2024

VAUSDEVH, Akhil. Automotive Communication Protocols. Skill Lync, 2020. Disponível em: https://skill-lync.com/blogs/auto-communication-protocols. Acesso em: 2 fev. 2024

VEIGA, Fernando. Utilizando MQTT com Node Red. Medium, 2018. Disponível em: https://medium.com/tht-things-hackers-team/utilizando-mqtt-com-node-red-1b41352e9c85. Acesso em: 3 fev. 2024

VENDRAMINI, Giovanna Schnorr. Dashboards: um guia de boas práticas para UX e UI Designers. Atelieware, 2021. Disponível em: https://ateliware.com/blog/dashboard-design. Acesso em: 3 fev. 2024

VOOGD, Kevin L. et al. Reinforcement learning from simulation to real world autonomous driving using digital twin. IFAC-PapersOnLine, v. 56, n. 2, p. 1510-1515, 2023.

WILLIAMS, Samuel V. et al. Systems analysis and remanufacturing: Testing of automotive electronics. In: Proceedings of the Industrial and Systems Engineering Research Conference, Montreal, Canada. 2021. p. 24-25.

XIA, Kaishu et al. A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence. Journal of Manufacturing Systems, v. 58, p. 210-230, 2021.

APÊNDICE A – Código exemplo da comunicação entre *CAN* e *UART*

Esse apêndice contém a função main.c e a função can.c que foram citadas de exemplo de uso de componentes de maneira modular e escalável.

- **Código main.c**

```
#include "can.h"
#include "uart.h"

QueueHandle_t can_to_uart_queue;

void app_main(void) {
    configure_UART();
    configure_CAN();

    can_to_uart_queue = xQueueCreate(10, sizeof(int));

    xTaskCreate(receive_CAN, "receive_CAN", 10240, (void
*)&can_to_uart_queue,
                9, NULL);
    xTaskCreate(send_UART, "send_UART", 10240, (void *)&can_to_uart_queue,
10,
                NULL);
}
```

- **Código can.c**

```
#include "can.h"
#include "../dbc/dbc_binUtil.h"

void configure_CAN(void) {
    // Initialize configuration structures using macro initializers
    can_general_config_t g_config =
        CAN_GENERAL_CONFIG_DEFAULT(GPIO_NUM_19, GPIO_NUM_21,
CAN_MODE_NORMAL);
    can_timing_config_t t_config = CAN_TIMING_CONFIG_500KBITS();
    can_filter_config_t f_config = {.acceptance_code = 0xA4000000,
                                    .acceptance_mask = 0x001FFFFF,
                                    .single_filter = true};

    // Install CAN driver
    if (can_driver_install(&g_config, &t_config, &f_config) == ESP_OK) {
        printf("CAN driver installed\n");
    } else {
        printf("Failed to install CAN driver\n");
        return;
    }

    // Start CAN driver
    if (can_start() == ESP_OK) {
        printf("CAN driver started\n");
    } else {
        printf("Failed to start CAN driver\n");
```

```
        return;
    }
}

void send_CAN(void *pvParameters) {
    // Configure message to transmit
    can_message_t message;
    message.identifier = 0xAAAA;
    message.flags = CAN_MSG_FLAG_EXTD;
    message.data_length_code = 1;

    QueueHandle_t *uart_to_can_queue;

    uart_to_can_queue = (QueueHandle_t *)pvParameters;

    u_int8_t item;

    while (1) {
        if (xQueueReceive(*uart_to_can_queue, &item, 0) == pdTRUE) {
            message.data[0] = item;

            // Queue message for transmission
            if (can_transmit(&message, pdMS_TO_TICKS(1000)) == ESP_OK) {
                printf("Message queued for transmission\n");
            } else {
                printf("Failed to queue message for transmission\n");
            }
        }

        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}

void receive_CAN(void *pvParameters) {
    // Wait for message to be received
    can_message_t message;

    QueueHandle_t *can_to_uart_queue;
    can_to_uart_queue = (QueueHandle_t *)pvParameters;

    dbc_rx_t holder;

    while (1) {
        if (can_receive(&message, pdMS_TO_TICKS(10000)) == ESP_OK) {

            uint32_t id = message.identifier;
            uint8_t dlc = message.data_length_code;

            dbc_Receive(&holder, message.data, id, dlc);

            int16_t data = holder.RS_Obj_00_Part01.RS_Obj00_XPos;

            // ESP_LOGI("  CAN RECEIVED", "%f", data * 0.02441);

            xQueueSend(*can_to_uart_queue, &data, 0);
        }

        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}
```

APÊNDICE B – Código fonte da página mostrada na Figura 9

Esse apêndice contém o código fonte em React.JS para a interface de monitoramento dos dados que foi detalhada durante a monografia.

- **Código fonte em React.JS**

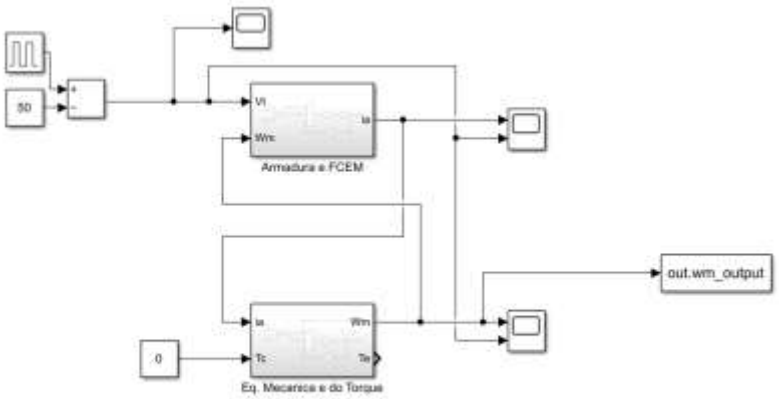
```
import React, { useState } from "react";
import Navfooter from "../components/sidebar/navfooter";
import LoggedData from "../components/logged/loggeddata";
import MapContainer from "../components/maps/MapContainer";
import useWebSocket from 'react-use-websocket';

function Details({ cleanusertoken }) {
    const [gpslist, setGPSlist] = useState([]);

    const [allData, setAllData] = useState([]);

    const [recent, setRecent] = useState({});

    const { lastJsonMessage } = useWebSocket('ws://localhost:8080', {
        onOpen: () => console.log(`Connected to App WS`),
        onMessage: () => {
            if (lastJsonMessage) {

                console.log(lastJsonMessage);

                setRecent(lastJsonMessage);

                allData.unshift(lastJsonMessage);
                setAllData(allData);
                setGPSlist(allData);
            }
        },
        shouldReconnect: (closeEvent) => true,
        reconnectInterval: 3000
    });

    return (
        <div className="hub-body">
            <Navfooter cleanusertoken={cleanusertoken} />
            <div className="main-body">
                <div className="main-header"></div>
                <div className="info-details">
                    <div className="info-widget">
                        <h1 className="info-title">Último dado</h1>
                        {recent ? (
                            <div className="info-block">
                                <h3>
                                    Velocidade:{" "}
                                    {recent.speed}
                                </h3>
                                <h3>
```

```
                                    Rotação:{" "}
                                    {recent.rpm}
                                </h3>
                                <h3>
                                    Aceleração:{" "}
                                    {"["+ recent.accx + ", " + recent.accy
+ ", " + recent.accz + "]"}
                                </h3>
                                <h3>
                                    Temperatura:{" "}
                                    {recent.temp}
                                </h3>
                            </div>
                        ) : (
                            <h1 className="loading">.</h1>
                        )}
                    </div>
                    <div className="map-widget">
                        {gpslist.length > 0 && <MapContainer
latlnglist={gpslist} />}
                    </div>
                </div>
                <div className="logged-general">
                    <div className="logged-frequents">
                        <h1>Dados recebidos</h1>
                        <nav>
                            <ul>
                                {allData === null && <h1>.</h1>}
                                {allData &&
                                    allData.map((detail) => (
                                        <LoggedData
                                            key={detail.time}
                                            speed={detail.speed}
                                            rpm={detail.rpm}
                                            acc={detail.acc}
                                            temp={detail.temp}
                                            lat={detail.lat}
                                            long={detail.long}
                                        />
                                    ))}
                            </ul>
                        </nav>
                    </div>
                </div>
            </div>
        </div>
    );
}

export default Details;
```

APÊNDICE C – Código Python em comunicação com Simulink e Unity 3D

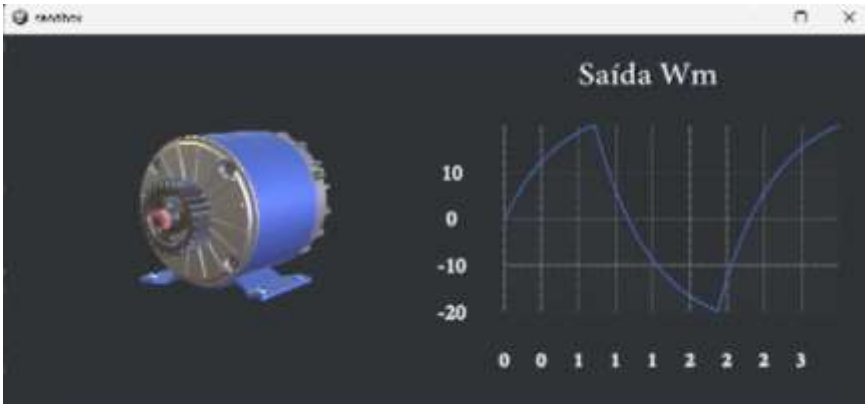
Esse apêndice contém o código fonte em Python que comunica com Simulink para interação com a simulação de motor CC mostrada na Figura 28, além de se comunicar com a ferramenta Unity 3D com o intuito de visualizar a resposta do projeto visto na Figura 29.

Figura 28 - Simulação de motor CC em Simulink.



Fonte: O Autor, 2024.

Figura 29 - Visualização do motor CC utilizando o Unity 3D.



Fonte: O Autor, 2024.

- **Código Python**

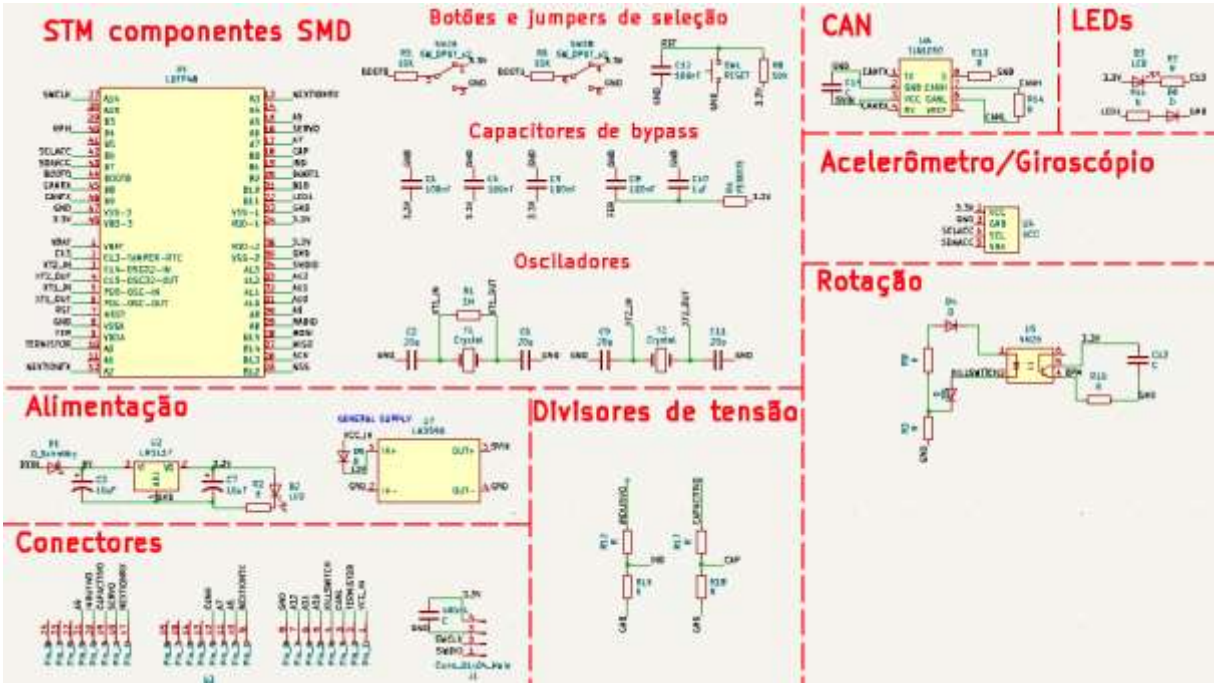
```
from peaceful_pie.unity_comms import UnityComms
import matlab.engine

unity_comms = UnityComms(port=9000)
matlab_engine = matlab.engine.start_matlab()

matlab_engine.eval("b = 0",nargout=0)
matlab_engine.eval("J = 4",nargout=0)
matlab_engine.eval("Km = 2",nargout=0)
matlab_engine.eval("La = 1e-9",nargout=0)
matlab_engine.eval("Ra = 0.5",nargout=0)
matlab_engine.eval("model = 'APS1'",nargout=0)
matlab_engine.eval("load_system('APS1')",nargout=0)
print("Initialized Model")

matlab_engine.set_param('APS1','SimulationCommand','start','SimulationComma
nd','pause',nargout=0)
matlab_engine.set_param('APS1','SimulationCommand','continue', nargout=0)

while (matlab_engine.get_param('APS1','SimulationStatus') != ('stopped' or
'terminating')):
    pass

matlab_engine.eval("wm_output = out.wm_output;",nargout=0)
matlab_engine.eval("tout = out.tout;",nargout=0)

wm_output = matlab_engine.workspace['wm_output']
tout = matlab_engine.workspace['tout']

wm_output = [i[0] for i in wm_output]
tout = [i[0] for i in tout]

def reduce_array_evenly(arr, target_length):
    if len(arr) <= target_length:
        return arr  # No need to reduce if array length is already less
than or equal to the target length

    reduction_factor = len(arr) / target_length
    reduced_arr = [arr[int(i * reduction_factor)] for i in
range(target_length)]
    return reduced_arr

wm_output = reduce_array_evenly(wm_output, 3*60)
tout = reduce_array_evenly(tout, 3*60)

unity_comms.Plot(data=wm_output)
unity_comms.PlotGraph(time=tout, data=wm_output)

matlab_engine.eval("La = 1",nargout=0)

matlab_engine.set_param('APS1','SimulationCommand','start',nargout=0)

matlab_engine.eval("wm_output = out.wm_output;",nargout=0)
matlab_engine.eval("tout = out.tout;",nargout=0)

wm_output = matlab_engine.workspace['wm_output']
tout = matlab_engine.workspace['tout']
```

```
wm_output = [i[0] for i in wm_output]
tout = [i[0] for i in tout]

matlab_engine.set_param('APS1','SimulationCommand','start','SimulationComma
nd','pause',nargout=0)

i = 0
la = 0.001
while (matlab_engine.get_param('APS1','SimulationStatus') != ('stopped' or
'terminating')):
    #la += 0.001
    #matlab_engine.eval("b = {}".format(la),nargout=0)
    #i+=1
    #print(i)

matlab_engine.set_param('APS1','SimulationCommand','continue','SimulationCo
mmand','pause', nargout=0)

matlab_engine.eval("wm_output = out.wm_output;",nargout=0)
matlab_engine.eval("tout = out.tout;",nargout=0)

wm_output = matlab_engine.workspace['wm_output']
tout = matlab_engine.workspace['tout']

wm_output = [i[0] for i in wm_output]
tout = [i[0] for i in tout]
```

APÊNDICE D – Esquemáticos das *ECUs*

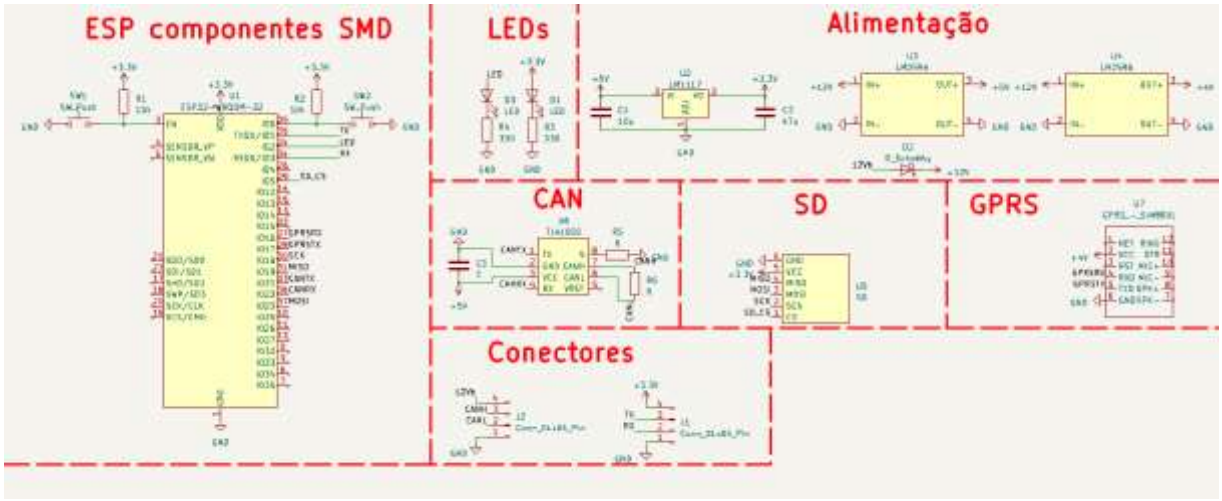
Esse apêndice contém os esquemáticos desenvolvidos para as centrais. Sendo os circuitos que definiram o desenvolvimento das placas das Figuras 16 e 17. A Figura 30 contém o esquemático das *ECUs* dianteira e traseira, e a Figura 31 contém o esquemático da *TCU*.

Figura 30 - Esquemático *ECUs* dianteira e traseira.



Fonte: O Autor, 2024.

Figura 31 - Esquemático *TCU*.



Fonte: O Autor, 2024.