AutLive: ferramenta de apoio ao monitoramento e testes de veículos autônomos em pequena escala

Samuel Barbosa Jatobá de Souza[1], **Abel Guilhermino da Silva Filho**[1]

[1]Centro de informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851 – 50732-970 – Recife – PE - Brazil

sbjs@cin.ufpe.br, agsf@cin.ufpe.br

Abstract. *The autonomous vehicle market grows every year and, with it, various technologies emerge, such as Platoon, in which a convoy of vehicles is formed and its members can move short distances between each other without human intervention. Therefore, to test these scenarios, teams around the world use scaled-down vehicles, simplifying associated costs. However, monitoring various real-time parameters of these vehicles, such as speed and distance between those involved, can be a complex task and a challenge that this work proposes to solve through the AutLive tool, developed to support testing with autonomous vehicles on a small scale.*

Resumo. *O mercado de veículos autônomos cresce a cada ano e, com ele, emergem diversas tecnologias como o* Platoon*, em que um comboio de veículos é formado e seus membros podem se locomover a curtas distâncias entre si sem intervenção humana. Portanto, para testar esses cenários, equipes ao redor do mundo utilizam veículos em escala reduzida, simplificando custos associados. Entretanto, monitorar diversos parâmetros em tempo real desses veículos, como velocidade e distância entre os envolvidos pode ser uma tarefa complexa e um desafio que esse trabalho propõe solucionar através da ferramenta* AutLive*, desenvolvida para apoiar testes com veículos autônomos em pequena escala.*

1. Introdução

O mercado de veículos autônomos tem crescido a cada ano e, só em 2022, estima-se que esse ecossistema tenha movimentado em torno de USD 121.78 bilhões de dólares. Além disso, para 2032 é esperado que o montante seja de aproximadamente USD 3 trilhões de dólares [Lee et al. 2017]. Portanto, para alcançar esse crescimento previsto, o número de testes e de desenvolvimento de novas soluções também deverá ser elevado e isso implica em lidar com uma grande quantidade de dados gerados, que atualmente pode chegar a 3 Terabytes por hora para cada veículo, capturados por sensores e câmeras [GRZYWACZEWSKI 2017]. Deste modo, reduzir a escala dos veículos implica, por consequência, na atenuação da quantidade de dados gerados dado a possibilidade de definir o escopo a ser testado, além de mitigar riscos de incidentes durante testes em ambientes controlados [WANG 2020].

Os veículos estão saindo das montadoras cada vez mais conectados e, com isso, o gerenciamento de informações de diversos sensores e sistemas demanda cada vez mais processamento. Por isso, as *Electronic Control Units (ECU)* passaram a ser adotadas com frequência e proveem mecanismos de comunicação de dados entre diversos barramentos e redes nos carros, sendo responsáveis até pela comunicação intra-veicular. Além disso, com o advento dos carros autônomos, essas estruturas passaram a lidar com cada vez mais responsabilidades, como os sistemas de assistência *Advanced Driver Assistance Systems (ADAS)* [Viegas 2022], que são responsáveis por aumentar a segurança nas estradas e confiabilidade dos motoristas.

Além do incremento da segurança nas estradas, uma outra vantagem pode ser provida pela adoção de carros autônomos: a possibilidade de deslocamento em pelotão ou *Platooning*, em que veículos andam a curtas distâncias espaçados a menos de 3 metros em um comboio, o que reduz o arrasto aerodinâmico entre os participantes e resulta em redução de uso do combustível, podendo alcançar taxas de 4%-8% de economia [YANG 2019].

Aplicar a formação de pelotão em veículos autônomos pode ser uma tarefa complexa em um ambiente com veículos de tamanho real, visto que diversos fatores de segurança devem ser considerados durante a sua execução [Janssen et al. 2015]. Portanto, neste cenário, a adoção de veículos em pequena escala pode reduzir riscos consideravelmente. Além disso, [Witaya et al. 2009] descreve algumas outras vantagens da adoção de veículos autônomos de pequena escala, listadas a seguir:

- **Redução de custos**: os veículos de pequena escala possuem menos componentes quando comparado com os de escala real e não são projetados para passageiros, o que implica em não exigir regulamentações e possuírem bem menos sistemas.
- **Simplicidade**: por não possuírem o mesmo número de sistemas internos, os testes se tornam mais simples, visto que é mais fácil contornar eventuais problemas em sistemas secundários. Além disso, um outro fator contribuinte é a redução de tamanho, de modo que veículos menores exigem menos espaço, se tornando mais fácil criar cenários que se adéquam ao objeto de estudo.
- **Mitigação de riscos**: teste de veículos de tamanho real exige uma preparação e regulamentação antes da serem realizados e, ainda assim, riscos estão envolvidos e incidentes podem ocorrer. Entretanto, no contexto de pequena escala, um

eventual atropelamento não ocorreria, visto que seu peso e velocidade são consideravelmente reduzidos.

Durante a execução de testes, é necessário entender se os componentes veiculares estão operando nominalmente, visto que falhas podem impor riscos para operação, além de impedir que os cenários em teste sejam validados [Kim et al. 2016]. Além disso, componentes como baterias podem fornecer estimativas de quando deixarão de fornecer a capacidade máxima de desempenho, sendo necessária a realização de uma manutenção preventiva, por exemplo. Portanto, é indispensável que a visão geral sobre o estado de cada veículo esteja disponível para monitoramento em tempo real, bem como informações de condições de funcionamento dos componentes internos.

Neste trabalho será explorado o desenvolvimento de uma ferramenta de apoio ao monitoramento e testes de veículos autônomos de pequena escala. Para isso, inicialmente os dados de sensores como movimento, ângulo de direção, entre outros, serão coletados das *ECUs* e enviados para uma aplicação front-end. Deste modo, a aplicação contará com uma estratégia de *Back-end for Front-end (BFF)*, em que o back-end emitirá dados para montar visualmente a solução e viabilizando uma maior reutilização da ferramenta. Além disso, o front-end deverá exibir um *dashboard* com informações relevantes sobre o funcionamento dos veículos em tempo real, dentre as quais destacam-se a seção de *Platoon*, em que as distâncias de cada veículo poderão ser visualizadas, qual dos carros é o líder do pelotão, quais veículos estão com freio acionado; A seção de gráficos, em que parâmetros de velocidade e direção serão exibidos, permitindo ao gestor o entendimento geral acerca do contexto dos veículos; A de comunicação, que contém os registros de mensagens trocadas entre cada *ECU*, bem como a possibilidade de visualizar os registros do pelotão; E, por fim, a de análise de falhas, responsável por exibir mensagens de erros dos componentes veiculares.

É objetivado construir uma ferramenta de apoio ao monitoramento e teste de veículos autônomos em pequena escala, com foco no acompanhamento de dados em tempo real, a partir de uma solução *front-end*, com disposição de informações de múltiplos sensores. Além disso, o trabalho avaliará o uso da solução em um ambiente laboratorial, no *Laboratório de Inovação para Cidades Inteligentes (LIVE)* na *Universidade Federal de Pernambuco (UFPE)*.

A presente pesquisa estende o trabalho desenvolvido por [Coelho 2023], que apresenta o desenvolvimento do *Small-Scale Autonomous Vehicle Infrastructure for Platoon (SAVIPS)*, uma infraestrutura para veículos autônomos de pequena escala para *Platoon*, na medida em que constrói uma solução *front-end* direcionada ao monitoramento dos veículos propostos pelo autor. Além disso, o veículo de pequena escala apresentado possui escala de 1/4, contém potenciômetros, sensores, câmeras e radar e é construído com os microcontroladores *Arduino Mega 2560* e *ESP32*.

2. Definições técnicas

Nesta etapa serão elucidados alguns conceitos pertinentes ao entendimento da pesquisa, como a definição de veículos autônomos de pequena escala, níveis de automação, sistemas de assistência ao motorista e técnica de *Platoon*. Na seção 2.1 é apresentada a visão geral sobre os veículos autônomos de pequena escala, na seção 2.2 é detalhada a definição de *ADAS* e os sistemas que a compõem, já na seção 2.3 é descrito o conceito de níveis de automação e sua importância no contexto veicular e, por fim, na seção 2.4 contém a formação em pelotão *Platoon*.

2.1. Veículos autônomos de pequena escala

Um veículo autônomo em pequena escala é uma solução encontrada por diversas equipes de pesquisa e desenvolvimento ao redor do mundo para redução de custos e teste de implementações no contexto de carros autônomos. Desta maneira, alguns trabalhos que os adotaram podem ser visualizados na figura 1 a seguir.

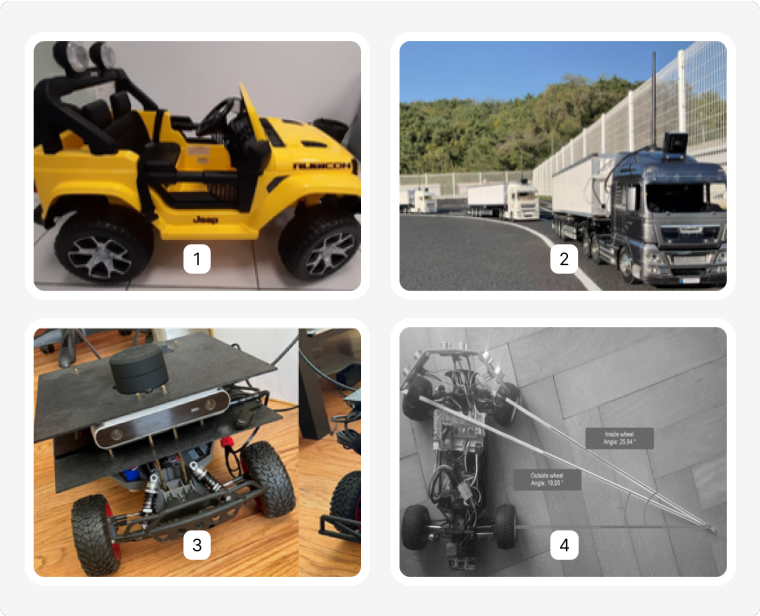

Figura 1. Recorte de veículos autônomos de pequena escala em diferentes tamanhos. Fontes: 1.[Coelho 2023], 2.[Lee et al. 2022], 3.[Yu et al. 2021] e 4.[Ciuciu et al. 2017].

Modelos em pequena escala podem prover os sistemas necessários para a execução de pesquisas por uma fração do custo quando comparados a um modelo de tamanho real e, como consequência, a operação se torna mais segura, dado que riscos de acidentes são mitigados [Marshall 2019]. Além disso, diferentes variações nas escalas são adotadas pelos pesquisadores, de acordo com os requisitos impostos para cada pesquisa. Desta maneira, os trabalhos que aderiram a diferentes escalas, representados na figura 1, serão brevemente descritos a seguir:

- **Escala 1/4**: Em *1*, [Coelho 2023] escolheu o *Bandeirantes Wrangler*, que possui uma razão de 1/4 com relação ao tamanho real, além de contar com tração 4x2 e ser baseado no veículo *Jeep Willys*, o autor justifica a escolha do modelo pelo aproveitamento do espaço interno e facilidade de montagem.

- **Escala 1/14**: Em *2*, o trabalho desenvolvido por [Lee et al. 2022] tem escala de 1/14 e é utilizado para estudo do *Platoon* em caminhões e contém sensores como *Lidar* e câmeras. Em sua conclusão, o autor compara as velocidades inferidas no modelo em escala para um caminhão em tamanho real.
- **Escala 1/20**: No ponto *3*, o trabalho de [Yu et al. 2021] adota a plataforma *Traxxas*, na escala de 1/20. Os autores modificaram o motor e adicionaram sensores como giroscópios, acelerômetros e também utilizaram um módulo *TX2 Jetson NVIDIA* para a unidade de processamento.
- **Escala 1/18**: Em *4*, [Ciuciu et al. 2017] adotaram um veículo de escala 1/18 em que as partes foram construídas utilizando impressão em *3D*. Além disso, sensores de angulação, motores servo, uma controladora *Arduino* foram utilizados para controle e monitoramento do veículo. Por fim, a pesquisa testou o modelo em uma pista de aproximadamente 11 metros de comprimento e 4 metros de largura, com rampas de 18º de inclinação.

Constata-se que há inúmeras variações de tamanhos e plataformas para o estudo de veículos autônomos em pequena escala. Entretanto, essa pesquisa desenvolverá uma solução de apoio ao monitoramento e testes do modelo apresentado por [Coelho 2023] e desenvolvida em conjunto com o *LIVE*, em uma escala de 1/4.

2.2. *ADAS*

O *ADAS* compõe um conjunto de sistemas de auxílio ao condutor, fornecendo vários níveis de automação, que vão desde alertas para que o motorista entenda o contexto do veículo, até execução de ações em sistemas com uso de algoritmos de *Machine Learning* (Aprendizagem de Máquina), como troca de faixas, frenagem semi-autônoma e piloto automático. Portanto, o objetivo principal desses sistemas é afirmar a segurança dos passageiros.

Como visto em [Nandavar et al. 2023], a adoção do *ADAS* reduz os incidentes nas estradas e melhora a segurança de condução dos veículos. Portanto, para corroborar com esse fato, serão descritos a seguir alguns dos sistemas que ajudam a melhorar a confiabilidade da direção através do *ADAS*:

- ***Adaptive Cruise Control (ACC)***: Ao conduzir um veículo a legislação obriga que uma distância seja respeitada em relação a outros carros na pista, visto que caso uma frenagem seja necessária, os envolvidos terão tempo para que a parada total em segurança ocorra. Para isso, o *ACC* ajuda a definir o espaçamento correto de acordo com a velocidade atual do condutor.
- ***Lane Keeping Assist (LKA)***: Em auto-estradas com fluxos de veículos em sentidos opostos ou com múltiplas faixas, é necessário que o condutor redobre a atenção para que um acidente não ocorra. Portanto, sistemas como o *LKA* são utilizados para impedir que acidentalmente o veículo não transgrida o sentido da via, colocando em risco a segurança dos passageiros. Dessa maneira, ao detectar um comportamento incomum, o sistema emite um alerta ao condutor, que deverá retomar o controle do veículo com urgência.
- ***Electronic Stability Control (ESC)***: É uma tecnologia que atua no monitoramento e controle da estabilidade do veículo e, caso detecte uma perda de controlabilidade, seja por uma curva realizada com uma velocidade muito alta ou falta de

aderência, pode atuar reduzindo a capacidade do motor e aplicando os freios com maior pressão e de forma assimétrica.

- ***Advanced Emergency Braking (AEB)***: É responsável por prover assistência de frenagem, monitorando a distância relativa a um obstáculo na frente do veículo. Além disso, o sistema pode emitir alertas sobre a necessidade de aplicar os freios e em caso de inatividade do condutor poderá acionar de forma automática os freios de emergência.

2.3. Níveis de automação

Nível de automação, ou interação entre o veículo e humanos, pode ser definido em até 6 escalas, de acordo com a *U.S. National Highway Traffic Safety Administration (NHTSA)*, que vai de zero a cinco, em ordem crescente de acordo com o nível de automação, em que no nível zero, não existe qualquer automação e, no nível 5, não há mais a necessidade de um condutor humano [NHTSA 2017].

No Nível 0, o sistema veicular pode apenas fornecer alguns alertas ou informações relevantes acerca da condução do motorista, através de um *ADAS*, por exemplo. Portanto, nessa modalidade, cabe ao motorista a responsabilidade de monitorar e dirigir o veículo.

Ao passar para o nível 1, o *ADAS* poderá emitir intervenções de freio e aceleração, em caso de uma mudança de faixa, por exemplo, utilizando-se de sistemas do *ADAS* como o *ACC* e *LKA*, anteriormente descritos. Portanto, no nível 1, o condutor ainda deve controlar e estar sempre atento.

Já no nível 2, além dos controles de frenagem e aceleração presentes no nível 1, o *ADAS* pode controlar o deslocamento lateral do veículo, em sistemas como o *Highway Pilot* em que a automação pode ajudar a manter o veículo na faixa. Entretanto, o condutor ainda deve monitorar e dirigir o veículo.

Ao alcançar o nível 3, o veículo consegue realizar a condução autonomamente. Entretanto, muitos cenários podem resultar na perda operacional do sistema, que passa a responsabilidade da operação para o motorista, que deve estar sempre atento.

Já no Nível 4, a ação do condutor passa a ser opcional. Ainda é possível assumir o controle caso queira, mas os sistemas do veículos são capazes de manter a segurança da operação sem necessidade de intervenções externas.

Por fim, no Nível 5, a diferença com relação ao nível anterior é que não existe mais a opção de um operador humano conduzir o veículo. Desse modo, todas as operações são realizadas com segurança por um sistema autônomo e auto-suficiente.

2.4. Platooning

A formação em pelotão ou *Platoon* é uma técnica de deslocamento amplamente explorada no estudo de veículos autônomos, em que uma sequência de carros é posicionado a curtas distâncias, a fim de reduzir o arrasto aerodinâmico gerado pelo atrito ou resistência com o ar ao locomover-se [Martínez-Díaz et al. 2021]. Desta maneira, os integrantes da formação devem se comunicar em tempo real, transacionando dados como a velocidade, posição e distância entre os membros. Além disso, para cada pelotão, existe um líder, que guiará os demais membros.

Existem alguns desafios para a utilização do *Platoon*, sendo um deles o *delay* de comunicação entre os integrantes, por exemplo, que deve ser baixo o suficiente para que as instruções sejam compartilhadas em tempo hábil e sem riscos à operação. Acrescenta-se também um outro fator relevante a ser considerado, a entrada e saída de participantes do pelotão, o que pode ocorrer a qualquer momento [Kamali et al. 2017]. Portanto, ao deslocar veículos em pelotão, é visível que a comunicação é preponderante para o seu sucesso. Desta forma, acompanhar o envio de mensagens entre os membros é necessário para a detecção de falhas e gerenciamento do comboio.

A visão geral da formação pode ser vista na figura 2 a seguir, que exibe os dois tipos de comunicação que ocorrem durante o processo: *Vehicle-to-Vehicle (V2V)* e *Vehicle-to-Infrastructure (V2I)*. Portanto, na comunicação *V2V* ocorre a troca de informações entre os membros do pelotão, ou seja, o líder é capaz de propagar sua velocidade e telemetria entre todos os veículos do comboio, sendo uma funcionalidade importante para o funcionamento esperado, dado que possíveis intervenções durante o percurso precisam ser compartilhadas para que redução da velocidade ou até uma parada sejam realizadas de maneira sincronizada. Outrossim, no domínio da comunicação *V2I*, o líder do pelotão é responsável por estabelecer uma troca de dados com a *Roadside Unit (RSU)* que por sua vez faz parte da infraestrutura e, tanto recebe quanto envia informações para os veículos e para dispositivos de controle de tráfego. Por sua vez, o controle de entrada e saída de um membro do *Platoon* é realizado pelo domínio de infraestrutura, que repassa para o líder do pelotão a entrada ou saída de um veículo do comboio.

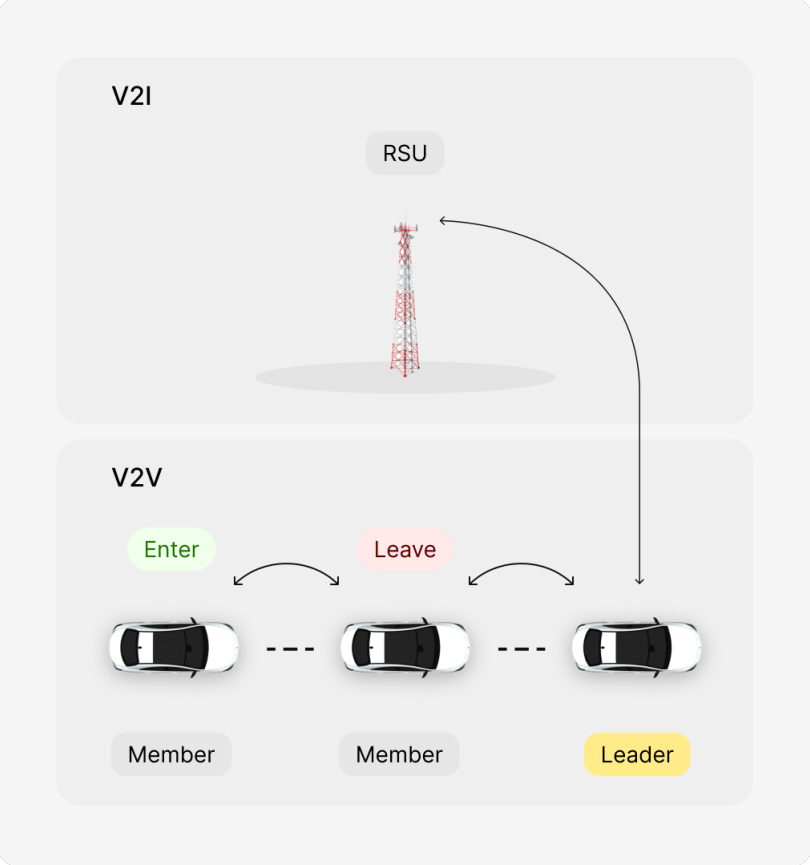

Figura 2. Representação dos domínios de comunicação na formação *Platoon*

3. Trabalhos Relacionados

Pesquisar e desenvolver soluções direcionadas a veículos autônomos exige a visualização de dados de diversos sensores, o que muitas vezes pode ser uma tarefa árdua a ser executada. Um exemplo disso é a realização de testes na formação de *Platoon*, em que as mensagens trocadas entre cada membro devem ser analisadas, pois falhas em seu recebimento podem resultar na perda de controle e, consequentemente, impor riscos a todos os envolvidos. Entretanto, o número elevado de dados trocados faz com que a detecção de falhas durante o processo se torne difícil. Em decorrência disso, este trabalho propõe uma ferramenta de apoio ao monitoramento e testes de veículos autônomos em pequena escala, objetivando reunir informações pertinentes aos veículos em um painel unificado e de fácil visualização. Desta maneira, parâmetros como velocidade individual, ângulo de direção, distância entre membros do *Platoon* podem ser rapidamente acessados, o que facilita o acesso aos dados e garante o acompanhamento dos veículos.

O trabalho desenvolvido em [Hamid et al. 2019] apresenta um sistema de monitoramento de parâmetros de motor veicular como o número de rotações e temperatura, utilizando a tecnologia de *Internet of Things (IoT)*, através de uma aplicação *Android* e *web*. Para isso, os dados são coletados por meio de um dispositivo *On-board Diagnostics II (OBD-II)* e enviados a um servidor em nuvem. Além disso, os autores afirmam que a latência alcançada para esse arranjo é de aproximadamente 1 segundo e que o trabalho foi capaz de categorizar motoristas de acordo com o perfil da velocidade na qual conduziram os veículos.

A pesquisa realizada por [BinMasoud and Cheng 2019] desenvolve uma solução web para monitoramento de dados obtidos através de diversos sensores, a partir de um dispositivo *OBD-II*, em que um algoritmo foi desenvolvido para previsão de falhas nos motores, bem como nos sistemas de refrigeração. Além disso, a quantidade de ar diluído ao combustível é medida através da variação do tempo médio consumido pelo motor para balancear a mistura. Por fim, os resultados obtidos foram validados com o veículo em movimento e em regime estacionário e a ferramenta *web* foi capaz de exibir corretamente o comportamento durante os testes.

Em [Krupitzer et al. 2019], os autores desenvolveram um *framework* baseado no simulador *PLEXE* [Segata et al. 2014], de código aberto e amplamente utilizado em pesquisas relacionadas a veículos autônomos e *Platoon*. Sobretudo, o *framework* simplifica configurações e fornece um ambiente de desenvolvimento com acesso à *API* do sistema de coordenadas *Platoon*. Além disso, os autores demonstraram uma ferramenta *web* para exibição de dados resultantes de simulações, baseado em *PHP* e *JavaScript*, que possui duas páginas: a primeira lista o conjunto de simulações previamente iteradas e que podem ser mutuamente comparadas; Na segunda, ao selecionar ao menos duas simulações, é plotado o mapa com o percurso percorrido por cada veículo e, parâmetros como a distância entre os veículos pode ser facilmente observado. Por conseguinte, devido ao elevado grau de desacoplamento da solução, o simulador utilizado pode ser substituído.

A pesquisa realizada por [Holla et al. 2023] apresenta uma alternativa simplificada de monitoramento em tempo real de diversos parâmetros veiculares. Para isso, os autores desenvolveram um sistema de aquisição de dados conectado ao barramento de saída da *Controller Area Network (CAN)* e enviaram os dados para uma placa *Raspberry Pi*. Deste modo, as informações coletadas são serializadas em formato temporal e envi-

adas para um *dashboard* na nuvem, onde são disponibilizadas para análise dos técnicos. Por fim, os resultados esperados da pesquisa são alcançados assim que os dados coletados pela placa são exibidos no *dashboard*.

No contexto de *IoT*, o trabalho realizado por [Ab Rahman et al. 2019] desenvolveu uma ferramenta de aquisição e monitoramento de dados veiculares em tempo real, utilizando-se da conexão *Wi-Fi* da *ECU*, responsável por controlar injeção de combustível e diversos parâmetros veiculares e, repassando as informações coletadas para um *System on Chip (SoC) ESP8266* de 32 bits através do protocolo de comunicação *Message Queuing Telemetry Transport (MQTT)*. Além disso, uma plataforma *front-end* foi desenvolvida, utilizando a tecnologia *Node-RED*, focada em *IoT*. Por fim, os autores proveram mais de uma forma de analisar os dados coletados pelo *SoC*: o *dashboard web* e em formato *CSV*, amplamente utilizado em ferramentas como o *Microsoft Excel*.

Simuladores como o *PLEXE* [Segata et al. 2014], que estende as funcionalidades do *SUMO* [Lopez et al. 2018], se relacionam com a corrente pesquisa na medida em que fornecem métodos de visualização em tempo real do deslocamento em *Platoon*. Nesse sentido, o trabalho implementou soluções para garantir a segurança na adoção de estratégias de direção cooperativa, como o deslocamento em pelotão, através de mecanismos de *fallback*, em que o controle do veículo volta a ser do condutor em casos de falha. Finalmente, algumas de suas funcionalidades chave são descritas a seguir:

- ***Traffic helpers***: são blocos responsáveis por inicializar o tráfego, permitindo até que uma formação *Platoon* pré-estabelecida possa ser rapidamente alocada. Além disso, também devem permitir que a perda de pacote de dados na comunicação entre a troca de pelotões seja simulada, facilitando o entendimento do cenário.
- ***Scenarios***: mapeiam comportamentos dos veículos, como o acionamento dos freios, até os padrões de velocidade de um líder do pelotão.
- ***Protocols***: responsáveis pela troca de *beacons*, que encapsulam os dados de controle. Por fim, nessa funcionalidade, há duas implementações com base no protocolo *IEEE 802.11p* [Segata et al. 2015].

Ao analisar os trabalhos listados anteriormente, é possível destacar algumas funcionalidades importantes, evidenciadas a seguir, e, que serão posteriormente comparadas com a ferramenta desenvolvida na presente pesquisa.

- ***Real-time sensor measurement***: possibilidade de ler dados de sensores do veículo em tempo real.
- ***Real-time sensor analysis***: capacidade de analisar os dados obtidos e disponibilizar relatórios a partir dos resultados obtidos.
- ***Desktop application***: disponibilização de aplicação para computadores.
- ***Web application***: disponibilização de aplicação *web*.
- ***Platooning visualization***: visualização da formação de pelotão ou *Platoon*.
- ***In-vehicle communication***: capacidade de monitorar comunicação interna do veículo, o envio, recebimento e falha de mensagens entre cada *ECU*.
- ***Platooning communication***: monitoramento de mensagens entre veículos membros da formação de pelotão.

A partir das funcionalidades listadas anteriormente, na *tabela 1* é realizada uma comparação entre os trabalhos relacionados e seus principais diferenciais, na

medida em que complementam o desenvolvimento da solução proposta. Por consequente, ao analisar os trabalhos de [Hamid et al. 2019], [BinMasoud and Cheng 2019] e [Ab Rahman et al. 2019] é constatada a capacidade de monitoramento e análise dos dados de sensores veiculares, além disso, as ferramentas desenvolvidas ofertam uma aplicação *web*. Já a solução desenvolvida por [Krupitzer et al. 2019] foca na exibição do *Platoon*, dispõe de aplicações *web* e *desktop* e fornece uma visualização da comunicação do pelotão. Ao mesmo tempo que, no trabalho de [Holla et al. 2023], o *dashboard web* exibe resultados dos sensores extraídos do barramento da *CAN* em tempo real, além da comunicação intra-veicular. Por fim, no simulador desenvolvido por [Segata et al. 2014] é possível analisar dados de visualização do pelotão, comunicação entre os veículos e a comunicação do pelotão, em uma aplicação *desktop*.

Feature	[1]	[2]	[3]	[4]	[5]	[6]	Proposal
Real-time sensor measurement	X	X		X	X		X
Real-time sensor analysis	X	X			X		X
Desktop application			X			X	X
Web application	X	X	X	X	X		X
Platooning visualization			X			X	X
In-vehicle communication				X		X	X
Platooning communication			X			X	X

Tabela 1. Comparativo das funcionalidades de cada solução relacionada com a proposta da pesquisa. [1] [Hamid et al. 2019], [2] [BinMasoud and Cheng 2019], [3] [Krupitzer et al. 2019], [4] [Holla et al. 2023], [5] [Ab Rahman et al. 2019], [6] [Segata et al. 2014].

A solução proposta nessa pesquisa reúne as principais funcionalidades previamente listadas em um *dashboard web* e *desktop* multiplataforma e dispõe de análise dos sensores, informando se apresentam falhas ou não, da visualização dos dados lidos em tempo real, da comunicação intra e inter-veicular com foco na visualização do *Platoon*. Dessa forma, a seguinte pesquisa integra essas funcionalidades através de uma ferramenta *front-end* e direcionada para veículos autônomos de pequenas escalas, o *AutLive*.

4. AutLive: ferramenta de apoio ao monitoramento e testes de veículos autônomos de pequena escala

Como descrito anteriormente, o trabalho objetiva construir uma ferramenta de apoio para testes e monitoramento de carros autônomos de pequena escala, em colaboração com o *LIVE*. Entretanto, elaborar uma ferramenta para o contexto veicular não é uma tarefa trivial e o contexto, bem como limitações arquiteturais podem ser fatores limitantes durante o processo e devem ser levadas em consideração na etapa de concepção da solução. Para isso, foi definida uma metodologia que atendesse ao objetivo final da pesquisa.

A metodologia adotada pode ser dividida em 5 etapas, como descrito na figura 3: a definição da arquitetura geral do sistema (1), na seção 4.1 e que apresenta uma visão da arquitetura de ponta-a-ponta da solução; A definição do escopo do problema (2), na seção 4.2 que descreve os desafios que a ferramenta se propõe a resolver; A prototipação (3), seção 4.3, em que são descritas as técnicas de design utilizadas para definir a experiência de uso; Definição da arquitetura de software (4), seção 4.4, em que a arquitetura da implementação *front-end* é definida, bem como as ferramentas adotadas durante o desenvolvimento; Implementação (5), definida na seção 4.5, em que as etapas do desenvolvimento são listadas.

Figura 3. Etapas da metodologia.

4.1. Arquitetura geral do sistema

É pertinente para o entendimento da pesquisa, que o contexto de monitoramento dos veículos autônomos abordados pelo projeto seja compreendido, desde a geração da informação até a aplicação *front-end*, tema deste trabalho. Para isso, na figura 4 é apresentada uma visão geral do monitoramento de veículos autônomos em pequena escala desenvolvido pelo *LIVE*, dividido em 3 camadas: *Platoon* (1), responsável pelo envio de dados através de uma placa *ESP32*; *RSU* (2), que recebe informação dos veículos e repassa para a camada de monitoramento; Camada de monitoramento de baixa latência (3), onde se encontra a aplicação desenvolvida na presente pesquisa.

Na camada de *Platoon*, dados como velocidade, ângulo de direção e mensagens das *ECU* são coletados e processados por um microcontrolador *ESP32* embarcado em cada veículo. Desta maneira, o envio é realizado através do protocolo *ESP-NOW* [Systems], direcionado para aplicações de tempo real, críticas e que tolerem pouca perda de pacotes, permitindo ainda o envio em *broadcast*.

Ao receber os dados enviados do *Platoon*, a partir de um receptor *ESP32*, o módulo da *RSU* envia informações para uma aplicação *back-end* em seu servidor. Neste ponto, os dados são processados e emitidos via protocolo *Hypertext Transfer Protocol (HTTP)* para um banco de dados local, que correspondem ao contrato *BFF* definido na seção 4.4.1. Além disso, vale salientar que esse módulo é resultado do trabalho desenvolvido por [Menge 2023], que propõe a criação de um ambiente de simulação da comuni-

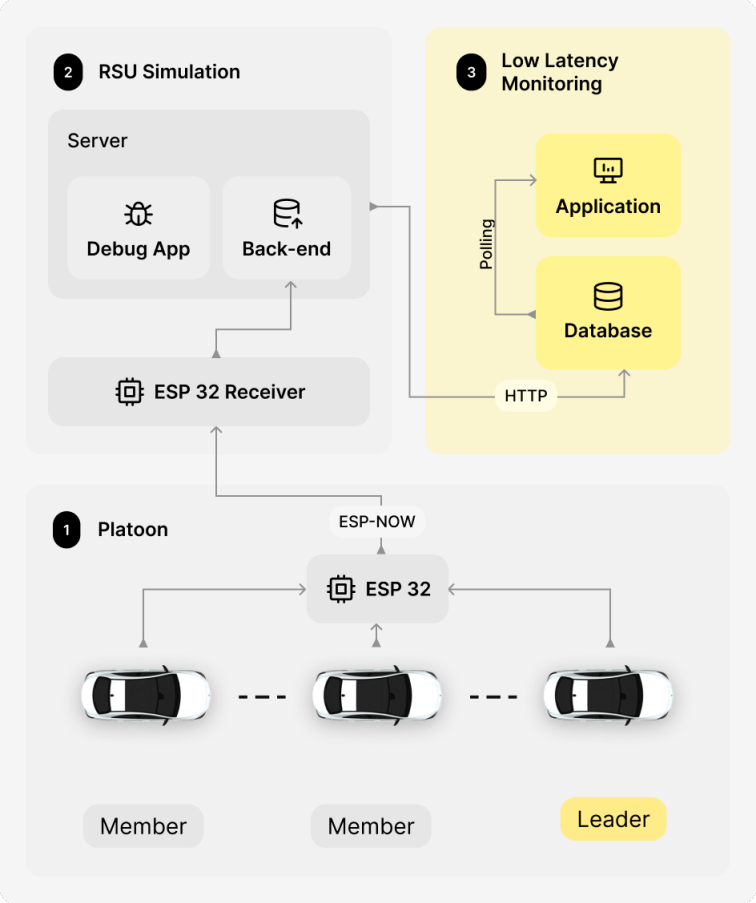

Figura 4. Arquitetura geral da comunicação de ponta a ponta.

cação *Vehicle-to-Everything (V2X)* para testes de ataques cibernéticos e vulnerabilidades em redes veiculares.

Por fim, na última camada, a de monitoramento de baixa latência, concentram-se a aplicação *front-end* desenvolvida na pesquisa e o banco de dados. Portanto, assim que o contrato definido for recebido, uma conexão *HTTP* estabelecida notificará mudanças no *front-end*, renderizando componentes e gráficos em tempo real.

4.2. Definição de escopo

O entendimento acerca do problema, suas limitações e principais desafios são conceitos necessários para a definição do escopo da pesquisa. Deste modo, algumas dificuldades foram levantadas inicialmente pelos pesquisadores do *LIVE* ao realizar testes com os veículos de pequena escala no laboratório, como a impossibilidade de monitorar parâmetros de comunicação em tempo real, devido à ausência de ferramentas apropriadas para esse fim. Além disso, durante a execução de testes, milhares de mensagens são enviadas a cada segundo entre os veículos e suas *ECU*. Portanto, localizar falhas de envio e recebimento não é trivial e demanda monitoramento contínuo em um grande volume de dados. Em vista disso, uma das principais funcionalidades que a solução precisa ofertar é o acompanhamento das informações trocadas.

A visualização de dados de diversos sensores também se mostrou importante, visto

que permite o acompanhamento de parâmetros relevantes no contexto, como velocidade, posição, direção e distância de cada veículo. Por conseguinte à expressiva quantidade de dados, definir uma abordagem genérica e escalável aparentou ser a melhor escolha nesse caso, sendo possível assim, condicionar a informação de acordo com a necessidade.

Um outro gargalo observado é a dificuldade de visualização simplificada de uma formação em pelotão, que é realizada em muitas ferramentas como o SUMO [Lopez et al. 2018], onde acontecem simulações em condições ideais. Porém, acompanhar dados como a distância entre veículos de um pelotão e ângulo de direção em condições laboratoriais requer uma infraestrutura e interfaces pensadas especificamente para esse propósito. Portanto, devido à sua relevância, essa dificuldade foi mencionada durante a prospecção.

Deste modo, foi estabelecido que a solução apresentada nesse trabalho deveria ser capaz de monitorar dados em tempo real de diversos sensores, ser escalável para que os dados possam ser facilmente alterados e definir um conjunto de ferramentas para análise da comunicação intra e inter-veiculares.

Por fim, foi necessário entender o ambiente em que os testes e pesquisas são conduzidos no *LIVE*, além da disponibilidade de equipamentos e facilidade de acesso. A partir disso, pôde-se compreender que acesso a uma aplicação *desktop* e via navegadores web, como o *Google Chrome* eram requisitos da solução. Por consequente, uma abordagem multiplataforma deveria ser adotada para permitir versatilidade de acesso à ferramenta.

4.3. Prototipação

A partir da definição do escopo do problema, a etapa subsequente é a prototipação, que é composta por algumas das técnicas de prototipação presentes na literatura [Wang et al. 2022], sendo adotadas 3 delas, especificamente e, brevemente detalhadas a seguir:

- **Jornada do usuário**: consiste no mapeamento de ações e sentimentos do usuário e é comumente utilizada para entender o passo a passo executado para realizar uma determinada ação.
- **Wireframe**: é uma técnica que consiste na elaboração da estrutura a nível de *User Experience (UX)*, em que não há uma preocupação com fontes e cores, apenas com a posição e informações a serem exibidas e com a experiência de uso.
- **Protótipo de alta fidelidade**: o desenvolvimento em alta fidelidade da solução entrega um modelo próximo ao resultado final a ser desenvolvido, a partir dos resultados aprendidos com a iteração das técnicas anteriores.

A jornada do usuário aplicada foi dividida em 4 seções: objetivos, necessidades, análise de sentimentos e barreiras. Dessa maneira, os objetivos dos pesquisadores ao utilizarem a ferramenta podem ser diversos, como o acompanhamento de parâmetros de sensores e a visualização de mensagens enviadas. Portanto, algumas necessidades relacionadas aos objetivos foram consideradas: a informação deve ser atualizada em tempo real, mas uma latência de poucos segundos pode ser tolerada e é ideal que a aplicação seja intuitiva. Além disso, uma projeção da perspectiva do usuário foi realizada para entender o comportamento ao realizar atividades na ferramenta. Ao final, barreiras como a falta de conexão com a internet foram consideradas.

Durante a etapa de *wireframe* foi realizada uma análise crítica sobre o conteúdo a ser exibido na ferramenta, dado que as informações serão atualizadas em tempo real. Assim sendo, reduzir o excesso de elementos repetitivos como grandes listas pode ajudar a carga cognitiva imposta ao usuário, por exemplo [Cabrera 2021]. Além disso, em simuladores de formação em pelotão anteriormente apresentados, como o *SUMO*, é comum a adoção de visualização dos veículos vistos do topo, em duas dimensões. Por essa razão, foi definida uma experiência similar, o que pode reduzir o tempo de adaptação.

No protótipo de alta fidelidade é construído o conjunto de componentes, *design system*, tipografia e espaçamentos que permitirão o desenvolvimento da ferramenta. Portanto, nessa etapa a interface é priorizada e passa a ser construída a partir das definições anteriores. Em síntese, o protótipo final foi validado em conjunto com os pesquisadores, além de atender aos requisitos previamente descritos.

4.4. Definição da arquitetura de software

Ao definir o escopo do problema, as informações a serem exibidas e tendo um protótipo que valida a solução proposta, é necessário definir o contrato da aplicação. Acrescentando-se ao que já foi mencionado anteriormente, a escalabilidade da ferramenta é relevante e deve ser considerada. Por essa razão, foi proposta uma abordagem baseada no padrão *BFF* [Calçado 2015], que fornece uma maneira de entregar elementos visuais por meio de serviços *back-end* especializados. Em suma, a escolha foi embasada pela personalização que a arquitetura oferece, sendo possível remover veículos, gráficos e trocar até cores, sem a necessidade de intervenção na aplicação *front-end*, que por sua vez reage às mudanças em tempo real.

Inicialmente, o *Firebase* foi adotado para a troca de dados entre a solução *front-end* e a unidade responsável pela leitura das informações dos sensores, que não será detalhada em profundidade neste trabalho, mas brevemente mencionada. Entretanto, o elevado número de consultas e de dados trafegados iria resultar em um custo elevado devido ao alto número de leituras e escritas, o que motivou pela adoção de um servidor *HTTP* local, que consiga lidar com um grande volume de dados.

Como kit de desenvolvimento de interfaces, o *Flutter* foi adotado, devido às capacidades multiplataforma e programação orientada a eventos. Além disso, o *framework* escolhido fornece diversas formas de lidar com fluxos de dados e definir *listeners*.

No decorrer da seção serão detalhadas as escolhas previamente levantadas de acordo com a seguinte distribuição: na subseção 4.4.1 será apresentado o contrato da solução *front-end*, na subseção 4.4.2 serão listadas as ferramentas e *frameworks* adotados durante o desenvolvimento e na subseção 4.4.3 a definição da arquitetura *front-end*.

4.4.1. Definição do contrato

Um contrato de aplicação corresponde ao conteúdo a ser exibido e/ou enviado através da solução *front-end* e por isso, diversos fatores podem impactar na estratégia adotada, como as funcionalidades, regras de negócio e grau de desacoplamento pretendido para a solução. Portanto, devido à adoção da estratégia *BFF*, foi necessário construir uma arquitetura que atendesse à dinamicidade dos dados e ao mesmo tempo fosse capaz de

modularizar a solução. Para isso, foi proposto um modelo que reduza validações, regras de negócio e o processamento da solução *front-end*.

O contrato da aplicação *front-end* é composto por uma lista de veículos, em que cada um pode conter 4 objetos, relativos à visão geral, *"overview"*, aos gráficos de sinais, *"charts"*, à comunicação veicular, *"communication"*, e por fim, às falhas gerais, *"failure"*.

No *Listing* 1 é definido o objeto de visão geral, ou *Platoon* e exibe dados de distância entre membros do pelotão, velocidade média e distância alvo. Dessa forma, no *front-end*, o número de membros do pelotão reflete visualmente a quantidade de elementos retornados pela estrutura de dados. Além disso, na linha *3* uma variável é designada para informar se o freio foi acionado, o que fará a aplicação exibir o ícone associado, da mesma forma que é executado para a linha *4*, que indica se o veículo atual é o líder do pelotão.

```
1            "overview": {
2                "name" : "Veiculo 1",
3                "isBraking" : true,
4                "isLeader" : true,
5                ...
6            },
```

Listing 1. Pseudocódigo da representação da visão geral.

No código 2, *"charts"* relaciona gráficos associados a um veículo ou pelotão, em que há uma correspondência direta entre o resultado total de itens enviados e as linhas renderizadas pela aplicação. Além disso, parâmetros como valor, nome do gráfico e tempo podem ser facilmente modificados. Portanto, a quantidade de objetos contidos na lista da linha *1* representa o número de parâmetros exibidos em um gráfico, que precisa de uma série de registros para que seja plotado, incluindo a data e o valor, nas linhas *5* e *6*, respectivamente. Por isso, como definido no contrato, torna-se possível a exibição de quantos gráficos forem necessários.

```
1            "charts": [
2                ...
3                "values": [
4                    {
5                        "datetime": "1452488445471",
6                        "value": 5.0
```

Listing 2. Pseudocódigo da representação dos gráficos.

A estrutura de comunicação definida no pseudocódigo 3 lista diversos parâmetros como o conteúdo da mensagem, nome do veículo ou *ECU* e se houve falhas no envio e recebimento das mensagens. Esses dados também podem ser alterados, assim como a cor de cada bloco visual. Portanto, como já relacionado, a informação a ser exibida não é representada explicitamente no contrato, generalizando a implementação. Acrescenta-se ainda a indicação da linha *2*, *"hasFailure"*, que indica se o bloco de mensagens sofreu alguma falha, sendo importante para que filtros sejam realizados pelo usuário. Já entre as linhas *5-7* são representados as cores e conteúdo de cada mensagem do bloco.

```
1            "communication": [
```

```
2                   "hasFailure": false,
3                   "series" : [
4                       {
5                           "iconName": "check",
6                           "content": "0x56 | 30-30",
7                           "textColor": "WHITE"
8                           ...
```

Listing 3. Pseudocódigo da representação da comunicação.

Por fim, o código 4 define o objeto de falhas, composto por uma lista, onde cada elemento corresponde a um cenário passível de erro. Portanto, serão enviados nessa lista, todas as possibilidades de erro durante os testes, como a posição das rodas em sentidos opostos ou conclusão da rotina de inicialização. Deste modo, a aplicação *front-end* irá renderizar o código e mensagem de erro, definidas nas linhas *3* e *4*, respectivamente. Além disso, na linha *5* encontra-se uma variável utilizada para trocar a cor do componente para vermelho, caso verdadeiro, simbolizando um erro.

```
1               "failure" : [
2                   {
3                    "errorCode": "0x00",
4                    "errorMessage": "Calibration Routine",
5                    "hasError": true
6                   }
7               ]
```

Listing 4. Pseudocódigo da representação de falhas.

Valores numéricos foram definidos como números de ponto flutuante, de acordo com o padrão *IEEE 754* [IEEE 2019]. Por outro lado, os demais parâmetros como cores e ícones foram estruturados como *String*, o que garante flexibilidade para adaptar componentes textuais posteriormente. Portanto, a adoção de cores e ícones como *String* foi baseada na capacidade da plataforma de reagir a mudanças de sensores e parâmetros monitorados. Dessa forma, apenas é necessário enviar na solução *back-end* os dados de acordo com o contrato adotado e será imediatamente renderizado no cliente *front-end*.

Em síntese, a adoção de uma abordagem generalista possibilita a redução de regras de validação no *front-end* na medida em que a informação coletada passa por pouco processamento até ser exibida. Além disso, é possível customizar um grande número de variações que se adéquem melhor às necessidades dos pesquisadores, dado que os parâmetros recebidos pela aplicação *front-end* são em sua maioria prontos para a exibição.

4.4.2. Escolha de ferramentas e frameworks

Um dos requisitos da solução é a atualização de dados em tempo real, ou com o mínimo de latência possível. Considerando isso, um servidor *HTTP* local foi criado para hospedar a estrutura de *BFF*, em que o *front-end* realiza chamadas regularmente ao servidor, que retorna os dados atualizados.

O elevado número de parâmetros e atualizações em tempo real é um desafio por si só. Por isso, é indispensável construir a aplicação em torno de um *framework* robusto, direcionado a eventos e *stream* de dados. Dessa forma, é possível garantir maior facilidade para lidar com volumes de dados e atualizar os componentes que deles dependem. Felizmente, existem algumas tecnologias de desenvolvimento front-end atualmente, dentre elas, *Flutter* foi escolhida por atender aos requisitos e ser multiplataforma.

Além dos requisitos necessários, o *Flutter* fornece funcionalidades e melhorias que vão além do desenvolvimento multiplataforma, quando comparado ao *React Native*, por exemplo [Wu 2018]. Entretanto, por não ser um *framework* nativo para a plataforma alvo, a performance é um fator relevante e que deve ser considerado, em função do número de componentes visuais apresentados na tela e constantes atualizações. Por isso, como descrito por [Wu 2018], *Flutter* tende a alcançar melhores resultados que o *React Native*, principalmente em cenários com grandes listas e elementos que demandem alto uso de recursos computacionais. Portanto, para o desenvolvimento da solução apresentada na pesquisa, o *Flutter* versão *3.13.0* foi utilizado e a *Integrated Development Environment (IDE)* escolhida foi o *Visual Studio Code*, da *Microsoft* na versão *1.81.1*. Além disso, foi necessário a adoção de algumas bibliotecas externas para construção de gráficos e ícones, por exemplo, o *SyncFusion*, na versão *22.1.39* e o *Material Design Icons*, com a versão *7.0.7296*.

Um outro recurso relevante advindo da escolha do *Flutter* foi a capacidade de gerar aplicações *Windows* e *MacOS*, além de *web*. Portanto, a versatilidade faz com que a alteração de sistema operacional não impacte no uso da ferramenta. Além disso, a visualização em tempo real dos componentes criados acelera o processo de desenvolvimento, visto que erros de *layout* podem ser rapidamente identificados e tratados.

4.4.3. Definição da arquitetura front-end

A definição da arquitetura é uma etapa importante no desenvolvimento de *softwares* [Oliveira et al. 2022]. Portanto, definir a organização do fluxo de dados bem como a responsabilidade de cada componente é necessário para desacoplar e facilitar a reutilização de código, além de melhorar a legibilidade. Deste modo, alguns padrões arquiteturais foram considerados: *Model View ViewModel (MVVM)*, *Model View Controller (MVC)* e *Business Logic Object Components (BLOC)*.

O padrão *MVVM* apresenta uma camada responsável por regras de negócio e que deve desconhecer o domínio de exibição da aplicação, a *ViewModel*, já o *MVC* em contrapartida, une a lógica com a camada de exibição, condição que pode dificultar a realização de testes unitários. Por sua vez, o *BLOC* é recomendado pela *Google*, mantenedora do *Flutter*, além de não expor métodos como o *MVVM*, uma vez que a interface de comunicação é realizada através de *listeners* e eventos [Cheboksarova et al. 2022].

Para elevar a compreensão acerca da arquitetura de software da aplicação, a figura 5 a seguir apresenta os 4 principais blocos esquematizados durante o desenvolvimento. No número *1*, está definida a camada de *provider*, que é responsável por suprir os dados definidos no contrato do *back-end*. Além disso, a implementação de chamadas *REST* e a comunicação com a base de dados é transparente a nível de interface, uma vez que a

camada de rede da aplicação assume o papel de comunicar sempre que uma alteração for detectada, em tempo real.

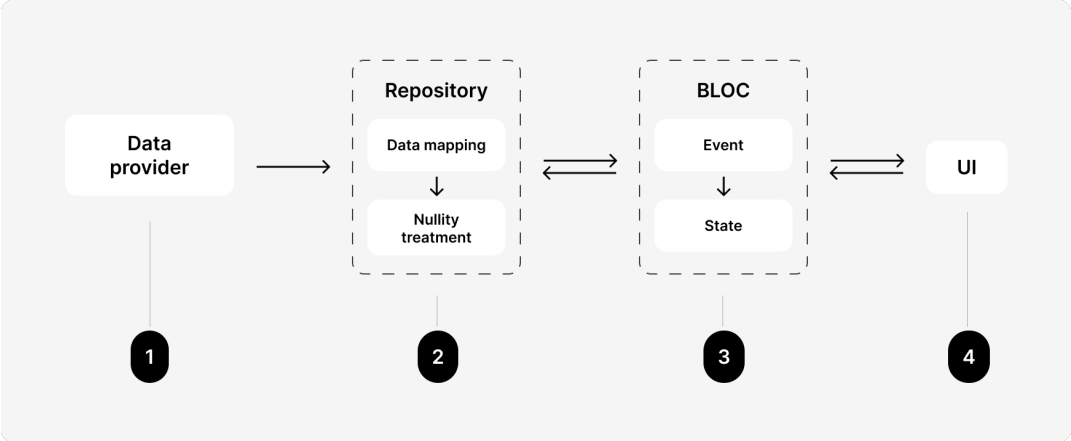

Figura 5. Diagrama esquemático da arquitetura de *software* adotada.

A camada de *Repository*, número *2*, recebe a comunicação do *Data provider* e mapeia para o domínio da aplicação, ou entidade de resposta. Portanto, no caso de um contrato que se espera receber listas, é conjecturado que existam laços de repetição para que cada elemento possa ser corretamente alocado. Além disso, uma outra atribuição do *Repository* é lidar com a ausência de dados, desde que dentro do acordo definido no *back-end*, sendo possível, por exemplo, que nenhum veículo tenha sido cadastrado.

Após os dados serem obtidos da rede e mapeados nas etapas *1* e *2*, a camada de *BLOC* é responsável por lidar com algumas regras de negócios, eventos e emitir estados. Assim, o recebimento de novas informações gera um evento que por sua vez resulta em um estado que deverá ser entendido pela camada de interface e utilizado para exibição dos componentes. Deste modo, alguns dos estados definidos são: *loading*, em que os dados estão sendo obtidos, *error*, quando um erro ocorreu e não existem dados a serem exibidos e por fim o *hasData*, em que a informação foi processada e pode ser exibida.

É possível que entre as fases *2* e *3*, bem como entre *3* e *4*, possam haver comunicações multilaterais, devido à interação existente na camada de exibição, que pode resultar em atualizações na camada de rede.

Por fim, a camada de *UI* ou de exibição em *4* é uma máquina de estados, que reflete a regra de negócio emitida pelo *BLOC*. Por consequente, ao enviar um estado contendo dados, a interface é renderizada de acordo com o conteúdo recebido. Além disso, todo o desenvolvimento de componentes visuais e interações é realizado nessa fase.

4.5. Implementação

A primeira etapa da implementação é a divisão do projeto na seguinte estrutura: diretório de *Blocs*, que contém o código de cada *BLOC* responsável por regra de negócios, *Models*, em que é definida a camada de domínio da aplicação, *Resources*, que contém imagens e ícones, *Repository*, encarregada de realizar o mapeamento do resultado obtido através de um servidor local para *UI*, onde estão localizados os *widgets*, blocos visuais de implementação do *Flutter*. Em síntese, a estrutura final pode ser visualizada a seguir na figura 6 e será detalhada no decorrer da seção.

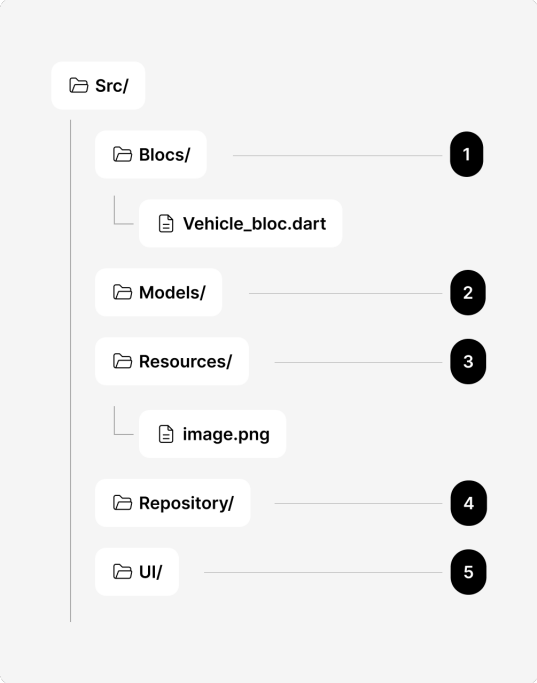

Figura 6. Distribuição de pastas e arquivos adotada na aplicação.

Em *Repository* (*4*), ficou definida a camada de rede da plataforma, contendo requisições e mapeamentos para a camada de domínio da aplicação, em que fica implementada a comunicação com o servidor, estabelecida através de um parâmetro definido no repositório, que determina o intervalo entre as chamadas *HTTP* em uma *thread* secundária. Deste modo, um *polling* é iniciado e a partir do intervalo definido e deverá consultar o servidor regularmente e, sempre que houver uma atualização no banco de dados, um *listener* irá notificar o método responsável pelos mapeamentos para entidades pré-definidas do contrato na camada de domínio. Diante disso, sempre que houver uma mudança nos dados obtidos através do *polling*, a camada de *UI* renderizará novamente os componentes contemplados pelas alterações.

O *Listing* 5 representa o método responsável por realizar o processo de chamada regular ao servidor *HTTP*. Deste modo, na linha *1*, o *Stream* corresponde a uma fonte de eventos assíncronos, em que um deles pode ser um erro. Além disso, o tipo retornado é um mapeamento de *String* e o escopo da função é assíncrono. Ainda na mesma linha, o caractere *'*'* implica que a função nunca termina a sua execução, sendo relevante para que novas requisições sejam sempre realizadas. Outrossim, na linha *2*, um laço de repetição sem condição de parada é definido, para que o *polling* continue sendo realizado. Além disso, já na linha *3*, é definido o intervalo entre as requisições, nesse caso de 5000 milissegundos, assim, a *thread* pode ser pausada pelo escalonador. Por fim, ao completar o período definido em *3*, o método que requisita um novo conjunto de dados via *HTTP* é executado e o parâmetro *yield* garante o retorno dos dados obtidos sem que o escopo da função seja finalizado.

```
1 Stream<Map<String, dynamic>?> getBFF() async* {
2     while (true) {
3       await Future.delayed(const Duration(milliseconds: 5000));
4       yield await executeBFF();
```

```
5       }
6   }
```

Listing 5. Código do *polling* da aplicação *front-end*.

De acordo com o contrato definido, alguns parâmetros podem não ser recebidos e a nulidade foi considerada para que não impactasse no funcionamento da ferramenta. Dessa forma, fica permitido que um veículo possa ser projetado na seção de gráficos e não seja descrito na visão geral, por exemplo, o que é útil para representar um pelotão, em que informações individuais não precisam ser exibidas.

Na camada de modelo, *Models(2)*, ficaram mapeadas todas as entidades definidas no contrato, porém em uma estrutura mais próxima da interface de usuário. Deste modo, tipos mapeados como *String* na camada de serviço e que representavam enumeradores foram estruturados na camada de domínio. Portanto, um exemplo dessa transformação é o conjunto de cores, que é simbolizado como *String* na camada de rede, mas ao passar para a entidade de domínio, passa a ser enumerado, trazendo diversas vantagens observadas durante o desenvolvimento, sendo uma delas a possibilidade de lidar com extensões de tipos abstratos ou estáticos, o que torna factível definir uma cor que o componente de *view* possa entender. Além disso, lidar com enumeradores pode ser menos suscetível a erros, já que é possível realizar a correspondência de um cenário não previsto como dado primitivo da camada de rede para um item esperado no enumerador.

No diretório *Blocs(1)*, foi implementado o *Bloc* de veículos, que é responsável por conter algumas regras de negócio e escutar o *listener* do repositório. Assim, quando uma alteração ocorre, o *Bloc* é notificado e realiza a emissão de um evento. Por sua vez, o evento gerado irá solicitar à camada de domínio o mapeamento dos dados oriundos da rede e ao concluir, emitirá um estado para o *Bloc*. Portanto, o estado emitido poderá ser uma das 3 opções: *loading*, enquanto os dados estiverem sendo obtidos; Erro, quando o contrato foi quebrado ou houve um problema de conexão; *Data*, assim que os dados são mapeados para a camada de domínio e podem ser direcionados para a aplicação.

Durante a implementação, foi previsto um cenário incomum, em que a listagem estivesse completamente vazia. Nesse caso, se a lista de veículos não apresentar elementos, um estado de erro será emitido pelo *Bloc*, o que evitará dar acesso à interface sem que componentes estejam devidamente renderizados. Além disso, também foi implementado um método de *dispose* que é responsável pela limpeza e remoção de *streams*, sempre que for necessário pela aplicação.

Em *Resources(3)* ficaram armazenadas todas as imagens utilizadas pela aplicação, como alguns ícones e também constantes que serão utilizadas na camada de *UI*. Uma dessas constantes define métricas de espaçamento entre elementos, *padding*, e tamanhos de componentes para que possam ser facilmente modificados sempre que necessário. Entretanto, muitos dos ícones utilizados pela ferramenta não estão em formato de imagem, devido à adoção de uma biblioteca externa para prover fontes de ícones, a *Material Design Icons*. Portanto, a partir dela, foi possível receber o nome do ícone desejado através da camada de rede e repassá-lo para a renderização na camada de exibição, o que permite alteração de recursos sem a necessidade de intervenção da plataforma.

Por fim, em *UI(5)*, fica definida toda a implementação de *widgets*, os blocos de componentes visuais do *Flutter*. Além disso, esse diretório foi dividido em duas partes,

as extensões e os *Widgets* propriamente ditos.

```
1 extension UIColor on ColorsModel {
2   Color toUIBackgroundColor() {
3     switch (this) {
4       case ColorsModel.white:
5         return const Color.fromARGB(255, 255, 255, 255);
6     ...
```

Listing 6. Trecho de código de uma extensão de cores.

Como previamente mencionado, as extensões têm grande relevância na implementação dessa solução, devido ao fato dos dados serem mapeados para uma camada de domínio e recebidos na camada de exibição como enumeradores ou estrutura de dados interpretáveis pelos componentes, como *Color*, por exemplo. Isso ocorre pois os *Widgets* não sabem lidar com *enums* do domínio e, por isso, extensões de cores e imagens foram criadas para converter descrições em tipos concretos interpretáveis. Portanto, ao enviar *"white"* como texto, o domínio da aplicação o mapeará para um enumerador *white* do tipo *ColorsModel*, como exemplificado no *Listing* 6 e que ao aplicar a extensão, a interface final tratará como *Color.fromARGB(255, 255, 255, 255)*, na linha *5*, o que é entendido pelo *Flutter*. Assim, o método definido na linha *2*, pode ser utilizado em todos os componentes, sendo totalmente escalável e independente da interface. Além disso, as extensões de imagens, por sua vez, foram utilizadas para mapear resultados da camada de domínio para ícones correspondentes na biblioteca, com tamanhos previamente definidos.

A tela principal da solução é composta por 4 macro-componentes ou *Widgets*, que correspondem às seções do protótipo: *Platoon*, *Signals*, *Communication* e *Failures*. Esses componentes apresentam mais de 1900 linhas de código, distribuídas em 35 arquivos. Deste modo, a primeira seção é a de visualização do pelotão, representada no ponto *1* da figura 7, e também é sub-dividida em diversos micro-componentes. Assim, para cada veículo registrado no contrato, um *Widget* contendo a imagem de um carro e parâmetros como a velocidade instantânea, ângulo de direção, se é o líder do pelotão e se o freio está sendo acionado é renderizado. Portanto, os ícones em amarelo e vermelho correspondem ao líder do pelotão e acionamento dos freios, respectivamente. Além disso, a decisão de exibir ou não cada uma das informações é da solução *back-end*, através das medições realizadas em cada veículo e, caso exista mais de um item na lista, há a possibilidade de projeção da distância entre pares do pelotão.

Na segunda seção, em *2*, informações gerais acerca de cada veículo são apresentadas na forma de gráficos. Portanto, a implementação adotada permite que veículos diferentes possam acompanhar dados separadamente. Assim, o *"Veículo 1"* pode exibir velocidade, e rotações do motor em determinada faixa de tempo, enquanto o *"Veículo 2"* pode monitorar taxa de frenagem e ângulo de rotação. Entretanto, como definido na prospecção, é importante que os mesmos gráficos sejam postos lado a lado para comparação, então uma linha de referência é traçada para cada um dos veículos. Além disso, um seletor foi desenvolvido para que a troca entre as informações exibidas de cada veículo pudessem ser facilmente intercaladas.

Na terceira seção, em *3*, é exibida a comunicação entre as *ECU*s de cada veículo, bem como entre os membros de uma formação de pelotão, por exemplo. Além disso, assim como definido no contrato, um carro poderia ser cadastrado sem possuir dados de

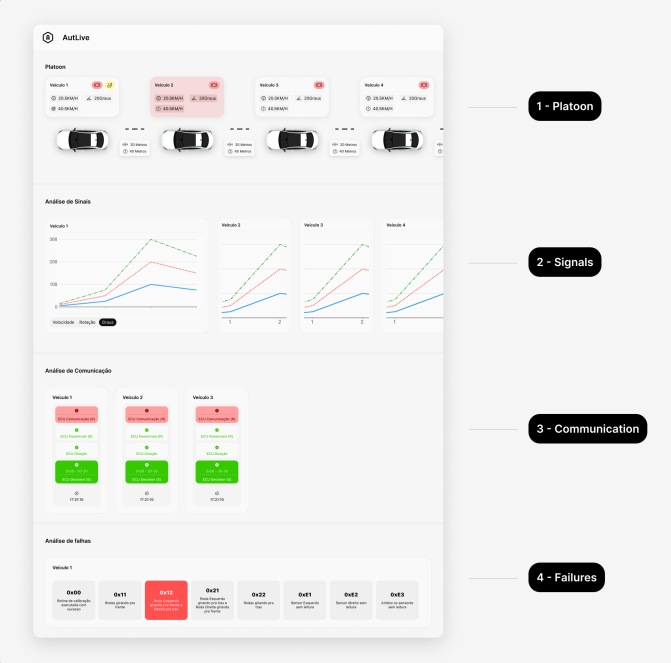

Figura 7. Visualização do *dashboard* do *AutLive*.

visão geral na seção de *Platoon* e, a razão para essa escolha é que isso permite que o *Platoon* seja analisado na seção de comunicação, sem que seja plotado na visão geral, o que não faria sentido. Deste modo, assim como nos outros blocos, a comunicação também é composta de diversos micro-componentes, entre eles, as células que representam uma mensagem enviada ou recebida. Dessa forma, o contrato recebido consegue definir até o padrão de cores de cada elemento, bem como o formato da mensagem, sendo assim possível interpretar com maior facilidade o contexto.

A implementação da seção de comunicação é modular e isso traz algumas vantagens, como a possibilidade de alteração de padrão de cores, ícones, conteúdos renderizados e também a ordenação e, com isso, a ordem dos itens é exibida de acordo com a forma que foram recebidos da camada de rede. Dessa forma, se os dados mais antigos passarem a ter prioridade, é possível enviá-los em ordem temporal reversa, sem intervenção da aplicação.

Por fim, na seção *4*, são listadas as falhas do ambiente veicular e podem incluir diversos mapeamentos, como o movimento das rodas em sentido contrário, ausência de dados de sensores e até execução da rotina de inicialização. Entretanto, a presença dos componentes não implica necessariamente em um erro e isso se deve ao fato da escolha em manter todos os cenários possíveis sempre na mesma posição, com o intuito de facilitar a observação dos erros quando ocorrerem. Portanto, ao ser marcado pela cor vermelha, internamente denotado por uma variável *booleana*, o componente associado indica para o gestor que aquele cenário encontra-se com erro no momento.

5. Resultados

Diversas condições e funcionalidades chave da ferramenta foram levantadas com base em trabalhos relacionados e dificuldades encontradas pelos pesquisadores do *LIVE*. Dentre as principais, encontram-se a baixa latência, boa usabilidade, monitoramento de sensores em tempo real, aplicação *desktop* e *web*, comunicação entre os veículos e a *ECU* e análise de falhas.

Objetivando analisar a solução desenvolvida, é necessário dividir o capítulo em 3 seções, em 5.1 é detalhado o ambiente de testes na qual a solução será submetida, na subseção 5.2 são exemplificados os cenários considerados para os testes, bem como os resultados obtidos durante sua execução e na seção 5.3, uma discussão acerca dos trabalhos relacionados no capítulo 3.

5.1. Ambiente de teste

Para condução dos testes, foi utilizado o espaço laboratorial do *LIVE-UFPE*, onde são desenvolvidas pesquisas relacionadas a cidades inteligentes e com foco na mobilidade urbana.

Os veículos autônomos de pequena escala adotados para o cenários a serem definidos posteriormente são os (*SAVIPS*), resultados do trabalho desenvolvido por [Coelho 2023] e que podem percorrer uma distância de até 183 centímetros em aproximadamente 8 segundos, de acordo com as medições realizadas pelo próprio autor.

Foi utilizado um computador *desktop Apple Mac Mini* executando o sistema operacional *MacOS 13.5* e processador *M2* de *3.5GHZ*. Além disso, a solução *AutLive* adotada é uma versão de testes em modo de *debug*, facilitando a interceptação de logs provenientes de requisições à camada de rede da aplicação, que é um servidor local, *localhost*, que recebe requisições *HTTP* na mesma rede em que o computador está conectado.

Para as funcionalidades de análise de sinais, na subseção 5.2.2, análise da comunicação, subseção 5.2.3 e análise de falhas, subseção 5.2.4, um servidor local desacoplado dos veículos foi utilizado com uma versão do contrato que atenda aos requisitos a serem testados. Dessa forma, os testes utilizando os veículos foram limitados ao *Platoon*.

5.2. Cenários de teste

Alguns cenários foram considerados para a realização dos testes, que incluem desde a configuração da quantidade de veículos, adoção do *Platoon* até a parametrização do intervalo entre requisições realizadas ao servidor local pela aplicação *front-end*.

O parâmetro de *polling* da aplicação *front-end* foi parametrizado em diversas configurações a serem definidas na seção 5.2, mas que passam por valores entre 5 segundos de intervalo entre as requisições até 0.5 segundo, correspondente a duas chamadas por segundo. Além disso, esse parâmetro é relevante pelo fato de controlar a taxa de atualização da ferramenta e por consequência a frequência com que os pesquisadores verão novos dados na plataforma. Deste modo, um valor muito baixo pode resultar na sobrecarga da aplicação e um valor muito elevado pode prejudicar o acompanhamento dos dados.

Os testes foram segmentados de acordo com as funcionalidades da ferramenta e estão listados nas seguintes subseções: a subseção 5.2.1 testa a visualização de *Platoon* a partir de diferentes arranjos de veículos, a subseção 5.2.2 apresenta a atualização dos

diversos gráficos de sinais dos automóveis, na subseção 5.2.3 é apresentado o resumo das *ECU* e se as mensagens foram recebidas corretamente e por fim, a subseção 5.2.4 exibe um conjunto de métricas do estado de cada veículo, indicando possíveis falhas.

5.2.1. Testes da seção *Platoon*

Inicialmente, a ferramenta *AutLive* foi testada com apenas um veículo *SAVIPS*. O objetivo pretendido para esse caso era garantir que a comunicação entre os veículos, *RSU* e *back-end* chegassem a até a aplicação final, o *front-end*.

Para isso, foi aplicada no veículo uma variação da posição das rodas e um deslocamento inicial, o que pôde ser acompanhado através da plataforma desenvolvida na presente pesquisa. Desta forma, o ponto *1* da figura 8 demonstra um recorte da seção de *Platoon* atualizada a partir dos dados emitidos pelo único veículo testado.

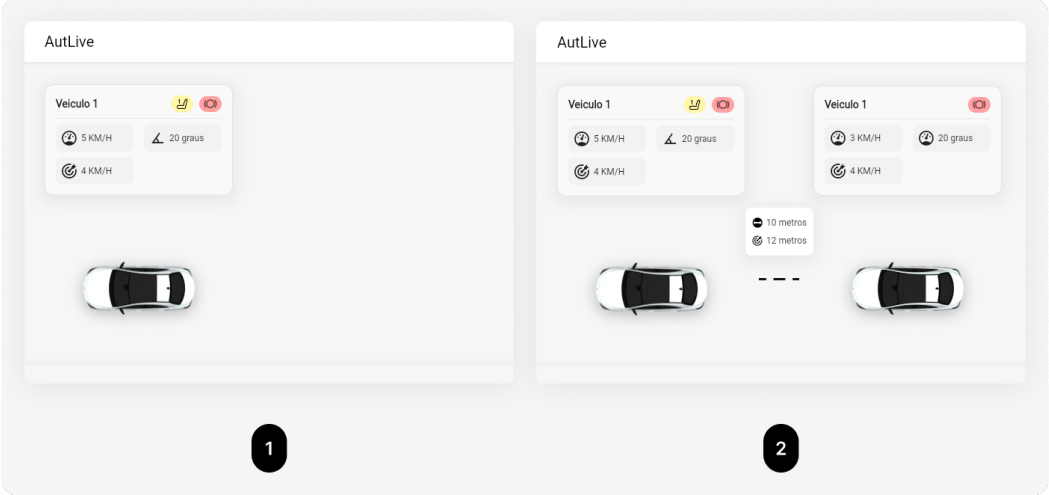

Figura 8. Corte da seção *Platoon* da ferramenta. Em *1* apenas um veículo foi incluído, em *2*, dois veículos são exibidos.

Ainda de acordo com a figura, é possível visualizar 7 informações relevantes: o nome do veículo membro do pelotão, indicador de líder do pelotão, representado pelo ícone amarelo, indicador de que os freios foram acionados, ícone vermelho, e as medições instantâneas de velocidade atual, ângulo de inclinação das rodas e a velocidade alvo, que é o objetivo que o veículo deve atingir no decorrer dos testes. Por fim, a cor branca ao redor das informações indica que o veículo está conectado e faz parte do pelotão.

Em seguida, um segundo veículo solicita ingresso ao pelotão e, com isso, passa a ser exibido no ponto *2* da figura 8. Dessa forma, assim que o veículo passa a integrar a seção, algumas informações pertinentes ao contexto são exibidas, como a distância relativa entre os dois membros e uma distância alvo previamente definida.

5.2.2. Testes da seção de sinais

A seção de sinais é responsável por exibir informações temporais acerca das medições realizadas em cada veículo e que compreendem a velocidade, os graus de direção das

rodas e o intervalo de distância. Portanto, para testar essa funcionalidade, foi definido um cenário em que um veículo emite os dados que serão plotados nos gráficos, a partir de um servidor local *HTTP* com dados pré-definidos. Dessa forma, foi possível comparar facilmente os dados exibidos com o conteúdo emitido pelo servidor, na qual os valores esperados na aplicação *front-end* foram alcançados.

Para esse teste, foram considerados 5 valores para o gráfico de velocidade, simulando o comportamento de freio de um veículo, em que há uma elevação gradual da velocidade seguido de uma redução abrupta, como é possível avaliar na figura 9 no ponto *1*. Além disso, também é visível que dois outros gráficos estão disponíveis para visualização, o de direção e espaçamento entre os veículos.

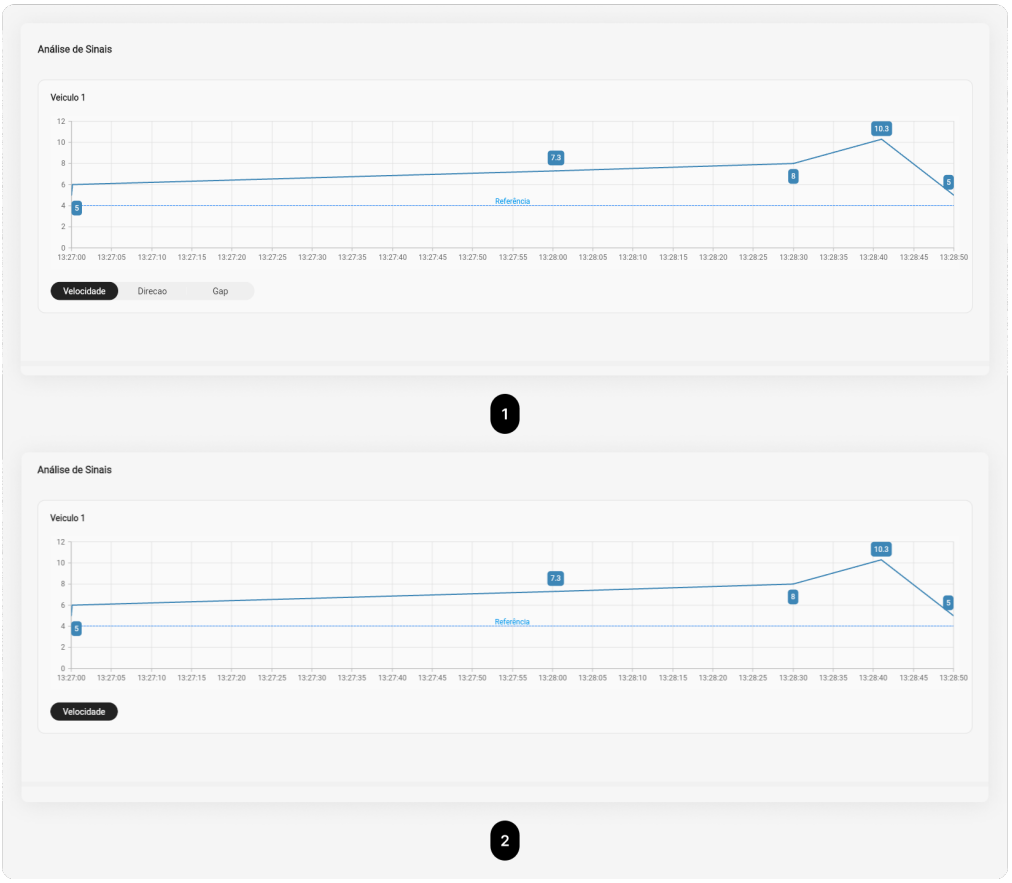

Figura 9. Recorte da seção de análise de sinais veiculares, antes e após a inserção dos gráficos de direção e *gap*.

Além do conteúdo estar condizente com o enviado pelo servidor, a ausência de uma das informações mapeadas também precisa ser considerada, dado que o contrato previamente estabelecido garante flexibilidade. Para isso, um novo conjunto de dados foi enviado pelo servidor local, contendo apenas o gráfico de velocidade, sendo esperado que a aplicação renderize exatamente o previsto. Por fim, o resultado pode ser visto no ponto *2* da figura 9, em que não há mais a presença de rotação e distância entre veículos.

5.2.3. Testes da seção de comunicação

A seção de comunicação tem como objetivo garantir que as mensagens trocadas no veículo entre as *ECU* sejam acompanhadas. Entretanto, em alguns casos é possível que uma *ECU* não receba a mensagem ou até não seja capaz de enviar. Por isso, serão simulados dois cenários, onde o primeiro vai indicar a falha de comunicação no *sender* e outro com erro no *receiver*.

Uma falha de comunicação no *sender* pode acontecer quando nenhuma das outras *ECU* forem capazes de interceptar a mensagem esperada. Dessa forma, para a formação de *Platoon* é importante que esses eventos sejam mapeados para que não possam prejudicar a formação. Dessa maneira, o primeiro teste compreende o envio de um contrato especificando que houve um problema ao enviar a mensagem no *sender*. Portanto, é esperado que a aplicação renderize um erro no bloco da *ECU sender* e que as demais permaneçam sem alteração, evidenciado no ponto *1* da figura 10. Por fim, no segundo teste, o *sender* envia a mensagem corretamente, no ponto *2* e a *ECU* de comunicação não recebe.

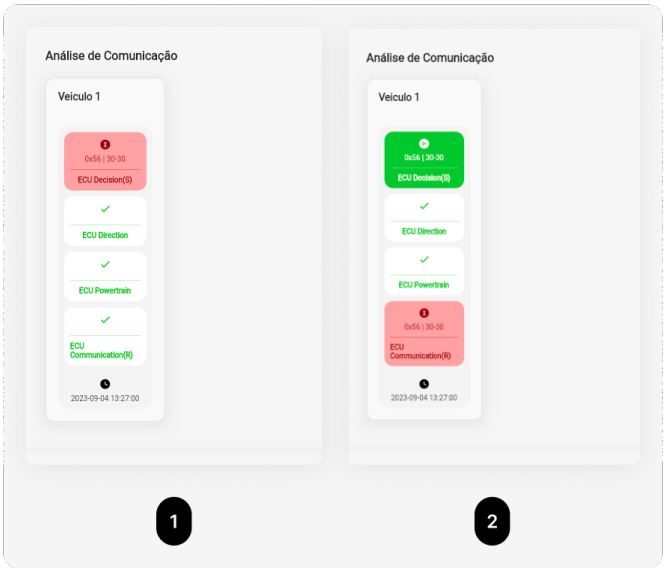

Figura 10. Recorte da seção de análise da comunicação veicular, retratando problemas de envio e recebimento da mensagem.

5.2.4. Testes da seção de falhas

A seção de falhas é responsável por exibir problemas nos sensores, nas rotinas de inicialização e posição das rodas em tempo real. Portanto, na figura 11 é evidenciada a interface da funcionalidade, em que é possível visualizar algumas das validações disponíveis, como a diferença de direção das rodas, representado pelo código *0x11*.

Para testar o funcionamento dessa seção foram propostos dois cenários de teste: a simulação de um cenário em que todo o sistema opera nominalmente, sem falhas, e um segundo cenário que considera falha na direção das rodas, representado pelo código *0x11*.

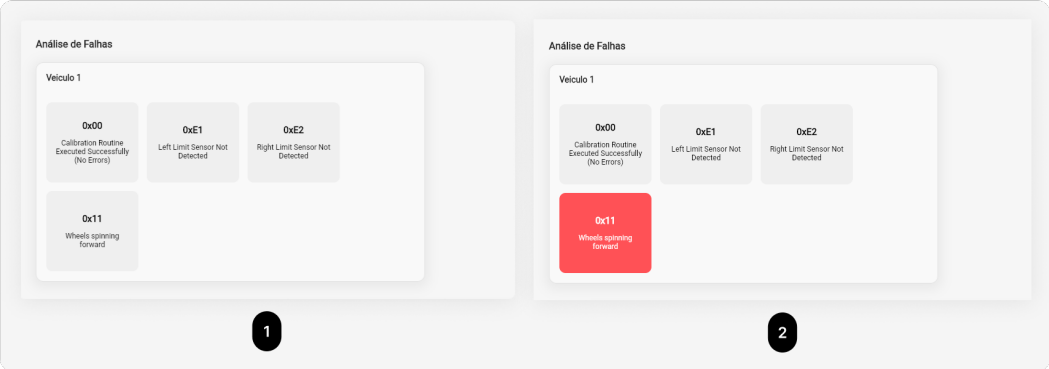

Figura 11. Recorte da seção de análise de falhas.

Em *1*, da figura 11, está representado o primeiro cenário de testes e pode ser concluído que a solução não emitiu em vermelho nenhum estado de erro. Entretanto, no ponto *2*, o mesmo comportamento não ocorreu, dado que a falha no código *0x11* implica na troca da cor do componente, marcado pela cor vermelha.

5.3. Discussão

Após testar todas as funcionalidades da ferramenta desenvolvida nessa pesquisa, é relevante comparar os resultados obtidos com trabalhos relacionados. Dessa forma, a pesquisa desenvolvida por [Hamid et al. 2019] apresentou um sistema de monitoramento de parâmetros de motor veicular como o número de rotações e temperatura, através de uma aplicação *Android* e *web*, é possível observar que a solução *Android* desenvolvida funciona como um controlador intermediário, responsável por enviar os dados para um servidor na nuvem. Entretanto, na interface *web*, a informação é recebida diferentemente do *AutLive*, em que gráficos dos parâmetros estão disponíveis para análise.

No trabalho realizado por [Ab Rahman et al. 2019], em que foi apresentada uma ferramenta de aquisição e monitoramento de dados veiculares em tempo real, o *dashboard web* construído com *Node-RED* se assemelha à solução proposta neste trabalho, na medida em que fornece a visualização de informações como o número de rotações do motor. Entretanto, seu foco é no monitoramento dos parâmetros em tempo real e não disponibiliza gráficos para acompanhamento da sua evolução. Dessa forma, o *AutLive* disponibiliza uma seção de gráficos para que os pesquisadores possam entender o contexto geral da operação.

Em simuladores como o *PLEXE* [Segata et al. 2014] e *SUMO* [Lopez et al. 2018], é possível visualizar uma representação gráfica dos veículos em pelotão e suas velocidades. Por isso, o *AutLive* é capaz de prover uma seção de *Platoon*, em que as distâncias de cada veículo estão representadas. Além disso, a velocidade e ângulo de inclinação das rodas também estão disponíveis.

6. Conclusão e trabalhos futuros

Com o aumento do número de veículos autônomos no mercado mundial e consequente elevação do investimento em pesquisa e desenvolvimento, além do advento dos veículos em pequena escala, torna-se necessário ampliar as ferramentas de apoio disponíveis. Pensando nisso, neste trabalho foi apresentada a solução *AutLive*, uma ferramenta de apoio ao

monitoramento e testes de veículos autônomos de pequena escala. A partir dela, o acompanhamento de parâmetros como a velocidade instantânea, distância relativa na formação de pelotão e falhas gerais foi simplificado.

A ferramenta desenvolvida trouxe algumas contribuições para o monitoramento de veículos autônomos em pequena escala, como a união de funcionalidades antes existentes a partir de programas separados. Além disso, o intervalo de atualização é parametrizável e permite flexibilizar a taxa de entrada de novos dados, o que pode facilitar no entendimento no processo de *debug*.

A solução apresenta algumas limitações que não foram exploradas e serão tema de trabalhos futuros, como a visualização em tempo real da posição relativa dos veículos na medida em que se locomovem, que atualmente é apresentada textualmente. Além disso, outros formatos de gráfico poderão ser implementados para oferecer uma nova perspectiva acerca dos dados extraídos. Por fim, a geração de arquivos *PDF* e *CSV* poderá ser gerada através de logs armazenados pelo banco de dados, estendendo as possibilidades para os pesquisadores, indo além da interface da plataforma.

Referências

Ab Rahman, A. F., Selamat, H., Alimin, A. J., Muslim, M. T., Msduki, M. M., and Khamis, N. (2019). Automotive real-time data acquisition using wi-fi connected embedded system. *ELEKTRIKA-Journal of Electrical Engineering*, 18(3-2):7–12.

BinMasoud, A. and Cheng, Q. (2019). Design of an iot-based vehicle state monitoring system using raspberry pi. In *2019 International Conference on Electrical Engineering Research & Practice (ICEERP)*, pages 1–6. IEEE.

Cabrera, M. (2021). The Psychology of Design: 15 Principles Every UI/UX Designer Should Know | Dribbble — dribbble.com. `https://dribbble.com/resources/psychology-of-design`. [Accessed 15-08-2023].

Calçado, P. (2015). The Back-end for Front-end Pattern (BFF) — philcalcado.com. `https://philcalcado.com/2015/09/18/the_back_end_for_front_end_pattern_bff.html`. [Accessed 15-08-2023].

Cheboksarova, N. I., Vakaliuk, T. A., and Iefremov, I. M. (2022). Development of crm system with a mobile application for a school. In *Ceur workshop proceedings*, volume 3077, pages 44–65.

Ciuciu, C., Bărbuţă, D., Săsăujan, S., and Şipoş, E. (2017). Autonomous scale model car with ultrasonic sensors and arduino board. *Acta Technica Napocensis*, 58(4):6–11.

Coelho, A. F. (2023). Savips - uma infraetrutura veicular autônoma em pequena escala para platoon.

GRZYWACZEWSKI, A. (2017). Training ai for self-driving vehicles: the challenge of scale. `https://developer.nvidia.com/blog/training-self-driving-vehicles-challenge-scale/`, Last accessed on 2023-08-06.

Hamid, A. H. F. A., Chang, K. W., Rashid, R. A., Mohd, A., Abdullah, M. S., Sarijari, M. A., and Abbas, M. (2019). Smart vehicle monitoring and analysis system with iot technology. *International Journal of Integrated Engineering*, 11(4).

Holla, R., Shanbhag, A., and Murugu, K. (2023). In-vehicle communication and data acquisition system.

IEEE (2019). IEEE Standards Association — standards.ieee.org. `https://standards.ieee.org/ieee/754/6210/`. [Accessed 16-08-2023].

Janssen, R., Zwijnenberg, H., Blankers, I., and de Kruijff, J. (2015). Truck platooning. *Driving the*.

Kamali, M., Dennis, L. A., McAree, O., Fisher, M., and Veres, S. M. (2017). Formal verification of autonomous vehicle platooning. *Science of computer programming*, 148:88–106.

Kim, B., Kashiba, Y., Dai, S., and Shiraishi, S. (2016). Testing autonomous vehicle software in the virtual prototyping environment. *IEEE Embedded Systems Letters*, 9(1):5–8.

Krupitzer, C., Lesch, V., Pfannemüller, M., Becker, C., and Segata, M. (2019). A modular simulation framework for analyzing platooning coordination. In *Proceedings of the*

1st ACM MobiHoc Workshop on Technologies, mOdels, and Protocols for Cooperative Connected Cars, pages 25–30.

Lee, D. H., Kang, D., Rzayeva, I., and Rho, J. J. (2017). Effects of energy diversification policy against crude oil price fluctuations. *Energy Sources, Part B: Economics, Planning, and Policy*, 12(2):166–171.

Lee, H., Park, J., Koo, C., Kim, J.-C., and Eun, Y. (2022). Cyclops: Open platform for scale truck platooning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8971–8977. IEEE.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE.

Marshall, A. (2019). These small cars can help drive the autonomous future.

Martínez-Díaz, M., Al-Haddad, C., Soriguera, F., and Antoniou, C. (2021). Platooning of connected automated vehicles on freeways: A bird's eye view. *Transportation Research Procedia*, 58:479–486.

Menge, G. (2023). An environment for testing cyberattacks and countermeasures in v2x communication.

Nandavar, S., Kaye, S.-A., Senserrick, T., and Oviedo-Trespalacios, O. (2023). Exploring the factors influencing acquisition and learning experiences of cars fitted with advanced driver assistance systems (adas). *Transportation research part F: traffic psychology and behaviour*, 94:341–352.

NHTSA (2017). The evolution of automated safety technologies. `https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety`, Last accessed on 2023-08-03.

Oliveira, B. R., Garcés, L., Lyra, K. T., Santos, D. S., Isotani, S., and Nakagawa, E. Y. (2022). An overview of software architecture education. In *Anais do XXV Congresso Ibero-Americano em Engenharia de Software*, pages 76–90. SBC.

Segata, M., Bloessl, B., Joerer, S., Sommer, C., Gerla, M., Cigno, R. L., and Dressler, F. (2015). Toward communication strategies for platooning: Simulative and experimental evaluation. *IEEE Transactions on Vehicular Technology*, 64(12):5411–5423.

Segata, M., Joerer, S., Bloessl, B., Sommer, C., Dressler, F., and Cigno, R. L. (2014). Plexe: A platooning extension for veins. In *2014 IEEE Vehicular Networking Conference (VNC)*, pages 53–60. IEEE.

Systems, E. ESP-NOW Wireless Communication Protocol | Espressif Systems — espressif.com. `https://www.espressif.com/en/solutions/low-power-solutions/esp-now`. [Accessed 22-08-2023].

Viegas, M. M. G. P. F. (2022). *The impact of ADAS in the insurance world*. PhD thesis.

Wang, J., Ranscombe, C., and Eisenbart, B. (2022). Prototyping in smart product design: Investigating prototyping tools to support communication of interactive and environmental qualities. *Proceedings of the Design Society*, 2:2243–2252.

WANG, J. e. a. (2020). Safety of autonomous vehicles. *Journal of Advanced Transportation*, 2020:1–13.

Witaya, W., Parinya, W., and Krissada, C. (2009). Scaled vehicle for interactive dynamic simulation (sis). In *2008 IEEE International Conference on Robotics and Biomimetics*, pages 554–559. IEEE.

Wu, W. (2018). React native vs flutter, cross-platforms mobile application frameworks.

YANG, Z.-F. e. a. (2019). Simulation study on energy saving of passenger car platoons based on drivaer model. *Energy Sources Part A Recovery Utilization and Environmental Effects*, 41(24):3076–3084.

Yu, T., Lu, W., Luo, Y., Niu, C., and Wu, W. (2021). Design and implementation of a small-scale autonomous vehicle for autonomous parking. In *2021 6th International Conference on Automation, Control and Robotics Engineering (CACRE)*, pages 398–402. IEEE.