UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

THIAGO NOGIRI IGARASHI

DESENVOLVIMENTO DE SISTEMA DE AQUISIÇÃO E CALIBRAÇÃO PARA ECUS

Joinville
2022

THIAGO NOGIRI IGARASHI

DESENVOLVIMENTO DE SISTEMA DE AQUISIÇÃO E CALIBRAÇÃO PARA ECUS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica , da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Dr. Anderson Wedderhoff Spengler

Joinville
2022

Dedico este trabalho à minha família.

AGRADECIMENTOS

Agradeço à minha família o incondicional apoio que me deram, por sempre acreditarem no meu potencial e por serem uma base forte na qual pude me sustentar, e à Caroline por sempre cobrar o melhor de mim e me incentivar a perseverar, vocês são fundamentais.

Sou muito grato aos professores Anderson Wedderhoff Spengler e Giovani Gracioli pelo suporte e orientação durante a pesquisa, e aos membros do Laboratório de Integração Software/Hardware que contribuíram para o desenvolvimento da pesquisa, em especial os mestrandos Sergio Arribas Garcia e João Paulo Bedretchuck. Também agradeço aos colegas Maria Eduarda Rosa e Miguel Suchodolak, por desenvolverem o programa gerador de experimento e o aplicativo de celular, respectivamente.

Agradeço à Renault do Brasil pelo fornecimento do carro utilizado e pelo suporte oferecido, em especial ao Erlon Murilo Fogaça pelo esclarecimento de todas as dúvidas, por participar de nossas reuniões e nos receber em visitar feitas à Renault.

Por fim, agradeço à Fundação de Desenvolvimento da Pesquisa – Fundep, Rota 2030 - Linha V pelo financiamento da pesquisa.

RESUMO

A medida que são adicionadas novas tecnologias aos veículos, sistemas eletrônicos de controle se tornam mais comuns e vitais para seu ideal funcionamento. Tais sistemas permitem que parâmetros do carro sejam calibrados durante seu desenvolvimento e adaptados conforme as condições do veículo. Para realizar a calibração e validação dos controladores, sistemas de aquisição e calibração são utilizadas. Tais sistemas representam uma grande despesa no processo de desenvolvimento dos veículos e poderiam ser substituídas por um sistema mais simples e de menor custo em estágios avançados do processo, nos quais deseja-se realizar a validação dos parâmetros calibrados. Este trabalho apresenta um sistema embarcado capaz de usar o protocolo de comunicação Controller Area Network (CAN) para coletar dados de variáveis da controladora do motor, além de permitir simples calibrações e o envio dos dados para um servidor em tempo real. O hardware foi desenvolvido tendo como principais componentes a placa de desenvolvimento FZ3 e os módulos EC25 e ESP32. O software foi projetado utilizando máquinas de estados para atingir o paralelismo. Além disso, foram realizados testes em diferentes condições de carga do sistema, a fim de comprovar a capacidade do sistema de processar os dados nas condições suportadas.

Palavras-chave: Sistema embarcado. CCP. ECU. Aquisição de dados.

ABSTRACT

As new technologies are added to vehicles, electronic control systems become more common and vital to their optimal operation. Such systems allow car parameters to be calibrated during development and adapted to vehicle conditions. To perform the controllers' calibration and validation, acquisition and calibration systems are used. These systems represent a large expense in the vehicle's development process and could be replaced by a simpler and lower cost system at advanced stages of the process, in which it is desired to validate the calibrated parameters. This work presents an embedded system capable of using the Controller Area Network (CAN) communication protocol to collect data from the engine controller variables, besides allowing simple calibrations and sending the data to a server in real time. The hardware was developed having as main components the FZ3 development board and the EC25 and ESP32 modules. The software was designed using state machines to achieve parallelism. In addition, tests were performed under different system load conditions in order to verify the system's ability to process data under the supported conditions.

Keywords: Embedded system. CCP. ECU. Data acquisition.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BLE Bluetooth Low Energy

CAN Controller Area Network

CCP CAN Calibration Protocol

CRM Command Return Message

CRO Command Receive Object

DAQ Data Acquisition

DTO Data Transmission Object

ECU Electronic Control Unit

JSON JavaScript Object Notation

ODT Object Descriptor Table

PID Packet Identifier

XCP Universal Measurement and Calibration Protocol

SUMÁRIO

1 INTRODUÇÃO

Múltiplas unidades de controle eletrônico (ECUs) estão presentes em carros modernos e são responsáveis por diversas funcionalidades como controle do motor, freios do tipo Anti-lock Braking System (ABS), airbags, janelas elétricas, entre outros. Em alguns casos, mais de uma ECU é usada para garantir uma característica do veículo, como o conforto e segurança do motorista (HERPEL et al., 2009).

Cada ECU se conecta a sensores e atuadores relacionados a sua área de funcionamento, além de se comunicarem entre si. Devido a isso, foram criados diversos protocolos de comunicação e transporte que facilitam e padronizam a troca de informação entre diferentes módulos, como o protocolo Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST) e Flexray. Dentre esses, CAN é o mais utilizado na indústria automotiva (HERPEL et al., 2009; SHARMA; MÖLLER, 2018).

Devido à competitividade do mercado automotivo, reduzir os custos de desenvolvimento de produtos é importante, ao mesmo tempo em que há demanda para produtos mais potentes e eficientes (YU; WILAMOWSKI, 2011). Porém, soluções comerciais capazes de se comunicar com ECUs de um veículo possuem hardwares com alto custo inicial e estão muitas vezes atrelados a softwares que requerem assinaturas (GRACIOLI; FRöHLICH, 2020).

Esse investimento é justificado em etapas iniciais de desenvolvimento, onde a programação e calibração da ECU é realizada em laboratório e deve assegurar que não há possibilidade de erros. Entretanto, em etapas finais, nas quais são realizados testes de verificação nas ruas, a coleta de dados não necessita de sistemas complexos e custosos, pois os dados obtidos são analisados posteriormente e, normalmente, não há alguém habilitado para verificar se houve algum problema durante o teste (GRACIOLI; FRöHLICH, 2020).

O objetivo deste trabalho é, portanto, projetar um sistema embarcado, de menor custo, capaz de se comunicar com uma ECU, extrair dados de variáveis e calibrar parâmetros. Além disso, os dados obtidos devem ser enviados pela internet em tempo real, permitindo a análise durante o experimento.

O sistema embarcado projetado é composto da placa de desenvolvimento FZ3, da MYIR Tech, que suporta uma distribuição Linux de nome PetaLinux e possui pinos dedicados para comunicação CAN. O sistema embarcado é programado em C++. Além disso, a placa é conectada a um microcontrolador ESP32-WROOM-32D, responsável por se comunicar com um celular utilizando o sistema operacional Android, usado para controlar o início e término do teste, bem como calibrar parâmetros e visualizar o estado

do sistema. Por fim, também é conectado o EC25-AU, um módulo 4G utilizado para medir a posição do veículo através de Global Positioning System (GPS) e transferir as informações adquiridas para um servidor.

1.1 OBJETIVO

Para resolver a problemática dos custos altos relacionados a testes de verificação, propõe-se neste trabalho os seguintes objetivos.

1.1.1 Objetivo Geral

Desenvolver um sistema embarcado capaz de medir variáveis e calibrar parâmetros de uma ECU, além de processar os dados e os enviar para um servidor.

1.1.2 Objetivos Específicos

- Identificar as características de sistemas de medição e calibração de ECUs;
- Caracterizar o sistema embarcado proposto;
- Comparar o sistema desenvolvido com alternativas comerciais.

2 TRABALHOS RELACIONADOS

Neste capítulo serão apresentadas pesquisas realizadas e produtos comerciais relacionados a medição e calibração de ECUs.

2.1 PESQUISAS RELACIONADAS

Ji-Hui, Hui e Ying (2008) desenvolveram uma plataforma que pode ser usada como base para sistemas que se comunicam com ECUs utilizando o barramento CAN e desejam enviar os dados pela internet. Para isso foi elaborado um hardware contendo um receptor CAN, que monitorava as mensagens transmitidas e salvava as desejadas em um buffer. Além disso, a plataforma era capaz de receber comandos da rede, a fim de calibrar parâmetros da ECU em tempo real.

Os dados salvos eram transmitidos pela internet usando um módulo Serviço de Rádio de Pacote Geral (GPRS) até um servidor, no qual os dados foram armazenados e posteriormente analisados. O microprocessador utilizado foi o XC164, de 16 bit, capaz de se comunicar com o módulo GPRS usando comandos AT para estabelecer uma conexão ponto a ponto com a rede (JI-HUI; HUI; YING, 2008).

Após onze meses de coleta de dados, os pesquisadores concluíram que o sistema era eficaz, estável e cofiável. Entre os resultados notáveis indicaram que a transmissão era melhor em momentos do dia em que há menos tráfego na rede, e pior em momentos em que a rede estava congestionada. Também concluíram que seria necessário reduzir o atraso na comunicação para ser possível utilizar o sistema em tempo real (JI-HUI; HUI; YING, 2008).

Wang et al. (2005) projetaram um sistema composto de um mestre e um escravo que se comunicam usando o barramento CAN e CAN Calibration Protocol (CCP). Ambos, mestre e escravo, foram projetados utilizando interfaces CAN e CCP e programados usando linguagem assembly. O sistema proposto mantinha os valores de parâmetros de calibração no mestre em uma região de memória chamada de memória espelho, de forma a reduzir a quantidade de leituras e escritas realizadas durante o funcionamento. Dessa forma, o mestre detinha uma cópia das informações de calibração e a memória espelho poderia ser usada para calibrar o escravo, mesmo sem um computador conectado ao mestre.

Wang et al. (2005) realizaram testes de simulação, bancada e nas ruas, em um motor movido a diesel e um motor elétrico, além disso, projetaram interfaces de calibração no computador, compatíveis com os motores a diesel e elétrico testados. Após os testes foi concluído que o sistema projetado era capaz de se comunicar rapidamente, gerenciar os dados de forma confiável e ajustar parâmetros durante

execução.

Xie et al. (2017) desenvolveram um sistema de acquisição de dados que utiliza a interface On Board Diagnostics (OBD2). Essa interface por sua vez utiliza o barramento CAN para se comunicar com a ECU. O sistema utilizou um microcontrolador STM32 para processamento das mensagens e um módulo CAN para receber e gerar tais mensagens. Além disso, a pesquisa ajuda a entender melhor o conteúdo transmitido no barramento CAN e como processar essa informação, apresentando a leitura de velocidade do carro durante um experimento junto com a mensagem recebida.

Baghli, Benmansour e Djemai (2014) apresentaram um sistema de coleta de dados que utiliza um celular para medir a posição usando GPS e enviar os dados para um servidor. O sistema também coleta dados através da interface OBD2 usando o microcontrolador ELM327, e os envia para o celular usando Bluetooth. Como resultado, apresentaram uma análise dos dados coletados e mapas com o trajeto realizado e informações do carro durante o trajeto.

As pesquisas encontradas auxiliaram a nortear o desenvolvimento do sistema e indicaram a possibilidade de coleta dos dados enquanto são enviados em tempo real. Também permitiram insights importantes para o desenvolvimento do firmware, como a forma escolhida de envio de dados e como lidar com o fluxo de dados recebidos. Por fim, foram importantes como referência para o desenvolvimento e escolha dos componentes do hardware.

2.2 SOLUÇÕES COMERCIAIS RELACIONADAS

Nesta seção serão apresentadas as principais soluções comerciais existentes utilizadas por fabricantes de automóveis.

2.2.1 ETAS GmbH

ETAS GmbH é uma subsidiária da Robert Bosch GmbH que produz sistemas eletrônicos e software capazes de medir, calibrar e diagnosticar ECUs, além de fazer testes e validações, usados em todas as etapas de desenvolvimento do software embarcado da ECU. Entre os produtos oferecidos pela empresa, se destacam a família de sistemas embarcados ETK, a família de módulos ES e o software INCA (ETAS GmbH, 2022a).

Entre os módulos ES, há a interface ES581, empregada para conectar um computador a uma ECU que conta com um conector serial e suporta comunicação CAN, FlexRay ou LIN (ETAS GmbH, 2022b). Tais características estão presentes em ECUs de desenvolvimento, mas as encontradas em automóveis vendidos não possuem tal conexão.

Diferente da interface ES581, a família ETK se conecta diretamente à ECU através de um barramento de dados paralelo ou uma interface de depuração, como a porta Joint Test Action Group (JTAG), e é instalada em cima ou até mesmo dentro da carcaça (ETAS GmbH, 2022c). Além disso, ETK se conecta a outros dispositivos utilizando o protocolo Ethernet, possibilitando transmissões de 100 Mbps, velocidade superior aos protocolos suportados pelo módulo ES581.

Os módulos oferecidos pela ETAS são controlados por um computador executando o software INCA, capaz de medir, calibrar e programar ECUs. INCA é compatível com os padrões da Association for the Standardization of Automation and Measuring Systems (ASAM), disponibiliza ferramentas de análise dos dados medidos e uma interface de usuário para visualizar os dados. Além disso, comercializa pacotes que adicionam suporte a outros protocolos e integram outras ferramentas, como Matrix Laboratory (MATLAB) (ETAS GmbH, 2022d).

2.2.2 Vector Informatik GmbH

Assim como a ETAS, Vector Informatik GmbH oferece soluções de hardware e software para desenvolver ECUs. A linha de produtos VN é composta de interfaces capazes de converter mensagens dos padrões CAN, LIN, J1708, FlexRay, 802.11p ou MOST em padrões comumente utilizados em computadores, como Universal Serial Bus (USB), Ethernet e PCI-Express (PCIe) (VECTOR INFORMATIK GmbH, 2022b). Além disso, Vector também dispõe de uma grande variedade de softwares, destacando-se entre eles o CANape, programa que permite calibrar a ECU conectada em tempo real, além de medir suas variáveis e as armazenar para análise (VECTOR INFORMATIK GmbH, 2022a).

A família VX1000 da Vector funciona de forma similar ao ETK da ETAS, um módulo chamado de Plug-on Device (POD) é conectado diretamente à ECU através de uma interface de depuração e instalado em seu encapsulamento. Um módulo base é então conectado ao POD e é responsável por controlá-lo e enviar os dados ao computador conectado. Diferentes módulos base suportam diferentes velocidades de transmissão e protocolos de comunicação, além de possibilitar o controle de mais de uma ECU simultaneamente (VECTOR, 2021).

2.2.3 Accurate Technologies Inc.

Accurate Technologies Inc. (ATI) desenvolve as interfaces A9, A8 e A7 que se conectam diretamente à ECU usando uma interface de depuração Device Access Port (DAP), On Chip Debug Support (OCDS) ou Nexus. Essas interfaces se conectam ao computador através do protocolo USB ou Ethernet (ACCURATE TECHNOLOGIES INC., 2022a).

ATI também oferece as interfaces CANary e Kvaser que convertem as mensagens do barramento CAN em USB. Todos os módulos da ATI utilizam o software VISION, capaz de medir variáveis e calibrar parâmetros (ACCURATE TECHNOLOGIES INC., 2022b). Embora o uso do VISION seja recomendado pela ATI, seus módulos também são compatíveis com software de outras fabricantes de sistemas de medição e calibração.

3 CONCEITOS RELACIONADOS AO TRABALHO

Neste capítulo serão apresentados conceitos importantes para o trabalho, como os sistemas de controle e protocolo usados em veículos.

3.1 ELECTRONIC CONTROL UNIT (ECU)

Uma ECU é um sistema embarcado usado para controlar sistemas mecânicos e eletrônicos em automóveis. Comumente, é composta de um microcontrolador, uma memória somente de leitura programável apagável eletricamente (EEPROM), uma memória de acesso aleatório estática (SRAM) e um meio de comunicação com outras ECUs e dispositivos (ALAM, 2018). Alternativamente, o termo ECU pode se referir a Engine Control Unit, uma ECU específica que é responsável pelo controle do motor do veículo.

Cada ECU é responsável por uma função ou um conjunto de funções específicos, e conectado a sensores e atuadores relacionados a essa função (ALAM, 2018). A ECU responsável pelo motor se destaca devido à importância de seu correto funcionamento para melhorar a execução e eficiência do veículo. Essa ECU se conecta a diversos sensores como o sensor de temperatura do líquido de arrefecimento do motor, de velocidade do virabrequim, de pressão do coletor de admissão, entre outros (FAIZAN; PILLAI, 2019).

Essa ECU também controla atuadores como o injetor de combustível e o controlador do acelerador. Além disso, a ECU do motor se comunica com outras ECUs, como as que controlam a velocidade, freio e transmissão, de forma a dividir o controle e reduzir a carga de processamento da ECU principal. Comumente, a ECU do motor apresenta tarefas ativadas em períodos fixos de tempo e tarefas ativadas pelo ângulo do virabrequim, que têm frequência de ativação proporcional à velocidade de rotação do motor (FAIZAN; PILLAI, 2019).

3.2 CONTROLLER AREA NETWORK (CAN)

CAN é um padrão definido pela Norma 11898, da International Organization for Standardization (ISO), criado para a indústria automotiva a fim de facilitar a ligação de diferentes nós usando dois fios (CHEN; TIAN, 2009). CAN utiliza uma arquitetura mestre-escravo, na qual múltiplos mestres podem estar conectados no mesmo barramento, sendo as mensagens transmitidas por broadcast, ou seja, a todos os nós conectados ao barramento (SILVA; SPOHN; PADILHA, 2017).

O padrão é compatível com o modelo Open System Interconnection (OSI), definindo as camadas física e de enlace, usado por protocolos de camadas superiores, como CANopen, CCP, XCP, entre outros. Versões atuais do protocolo suportam velocidades de transmissão de até um Mbps e identificadores de onze bits (para CAN 2.0A) ou 29 bits (para CAN 2.0B), portanto a versão mais recente permite até 537 milhões de identificadores em um mesmo barramento (PAN et al., 2017).

O protocolo CAN se destaca pela transmissão de pequenas mensagens para todos os nós, de até oito bytes de dados, sendo adequado para troca de pequenas informações como valores de sensores e comandos a atuadores. Além disso, é robusto pois oferece cinco métodos para checagem de erros nos quais qualquer falha resulta em uma mensagem de erro, indicando a necessidade de retransmitir a mensagem enviada (CHEN; TIAN, 2009).

3.3 CAN CALIBRATION PROTOCOL (CCP)

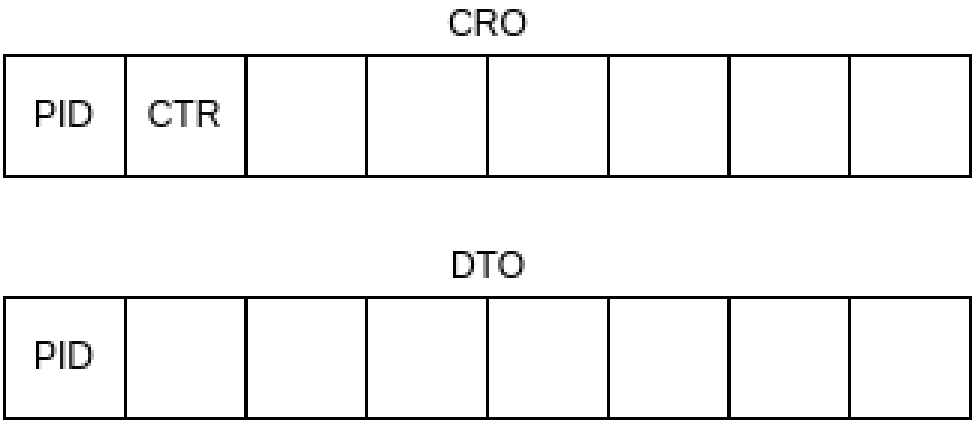
CCP é um protocolo de comunicação criado pela Association for the Standardization of Automation and Measuring Systems (ASAM), antiga Standardization of Application/Calibration Systems task force (ASAP), que utiliza o protocolo CAN e padroniza a comunicação entre um mestre e múltiplos escravos, sendo comumente usado para transferir comandos entre um computador e uma ECU (NASERIAN; KRUEGER, 2009). O protocolo suporta a leitura de variáveis internas e calibração dos parâmetros da ECU em tempo real, além de permitir sua programação.

Devido ao fato de usar CAN como protocolo de transporte, o CCP está limitado a mensagens de até oito bytes e todas as informações devem ser transmitidas dentro desse limite. Caso a ação desejada necessite, múltiplos comandos podem ser usados em sequência para realizar tal ação. Todo quadro do protocolo CCP deve ser identificado usando seu primeiro byte, no campo chamado de Packet Identifier (PID), que determina qual comando foi enviado e, portanto, como interpretar os conteúdos do quadro (ASAP, 1999).

O CCP define dois tipos de mensagens, Command Receive Object (CRO) e Data Transmission Object (DTO). Uma CRO contém um comando do mestre para o escravo e sempre requer uma resposta, chamada de Command Return Message (CRM). Uma DTO é uma mensagem do escravo para o mestre que pode ser uma CRM, uma mensagem de evento ou um quadro referente a transferências de dados assíncronas. A Figura 1 ilustra os dois quadros definido pelo protocolo, na qual cada quadrado representa um byte.

Além de identificar o comando usando o primeiro byte da mensagem, todo CRO precisa enviar o segundo byte como um contador de mensagens, que incrementa a cada envio. O conteúdo dos seis bytes restantes são definidos pelo comando escolhido,

Figura 1 – Formato dos quadros CCP



Fonte: Adaptado de ASAP (1999)

podendo ser usados ou não. Se o comando não necessitar de todos os seis bytes, os bytes não usados devem ser enviados com valores arbitrários. O Quadro 1 apresenta os principais comandos definidos pelo protocolo e suas funções.

O conteúdo das DTOs recebidas pelo mestre são interpretadas dependendo do seu PID. Se a DTO for uma CRM, o PID é 0xFF, o segundo byte é o valor do código de erro, e o terceiro é o número do contador do comando correspondente à resposta. Os outros bytes dependem do comando enviado. Se o PID for 0xFE, o quadro recebido é referente a um evento e o segundo byte é o código de erro que gerou o evento. Para PIDs entre 0x00 e 0xFD, o quadro é referente a uma transferência assíncrona de dados.

As seções a seguir explicam como as variáveis são medidas e os parâmetros calibrados usando CCP.

3.3.1 Transferência assíncrona de dados

O protocolo CCP suporta transferência de dados periodicamente e por evento, divididos em listas de aquisição de dados, chamadas de DAQ Lists ou rasters. Cada DAQ List é referente a um evento ou período diferente e contém uma quantidade de Object Descriptor Tables (ODTs) pré-definido pela programação realizada na ECU.

Uma ODT é uma estrutura capaz de conter endereços de memória referentes a até sete bytes de memória. Toda ODT deve estar relacionada a uma DAQ List e, quando a lista é acionada, um quadro contendo o PID da ODT e os valores contidos nos endereços de memória são enviados. A Figura 2 ilustra uma DAQ list com três ODTs.

No momento em que um evento é acionado ou um período é completo, todas as ODTs relacionadas à DAQ List são enviadas ao mesmo tempo. Portanto, a fim de garantir um consumo máximo de largura de banda, o número de ODTs para cada uma das listas é escolhido na programação da ECU e excedê-lo resulta em uma mensagem

Quadro 1 – Comandos importantes do protocolo CCP

Comando	PID	Função
CONNECT	0x01	Estabelece uma conexão com o escravo
GET_SEED	0x12	Retorna a seed do recurso ou função pedido
UNLOCK	0x13	Envia a chave referente a seed recebida, desbloqueia a funcionalidade desejada caso a chave esteja correta
SET_MTA	0x02	Inicializa um ponteiro no endereço de memória passado para transferências de memória realizados na sequência
DNLOAD	0x03	Copia os bytes de dados passados para o endereço de memória que começa no ponteiro criado pelo comando SET_MTA
UPLOAD	0x04	Solicita os bytes do endereço de memória do ponteiro criado pelo comando SET_MTA
SELECT_CAL_PAGE	0x11	Define a página de calibração apontada por SET_MTA como ativa
GET_DAQ_SIZE	0x14	Solicita o tamanho de uma lista e seu primeiro PID, além de limpar as ODTs configuradas a ela
SET_DAQ_PTR	0x15	Seleciona uma lista e ODT para subsequente configuração
WRITE_DAQ	0x16	Configura uma variável na lista e ODT apontado por SET_DAQ_PTR
START_STOP	0x06	Inicia ou termina a transmissão de dados referente a lista passada
DISCONNECT	0x07	Desconecta o escravo
GET_ACTIVE_CAL_PAGE	0x09	Solicita o endereço de memória do início da página de calibração ativa
START_STOP_ALL	0x08	Inicia ou termina a transmissão de dados de todas as listas configuradas

Fonte: Adaptado de ASAP (1999)

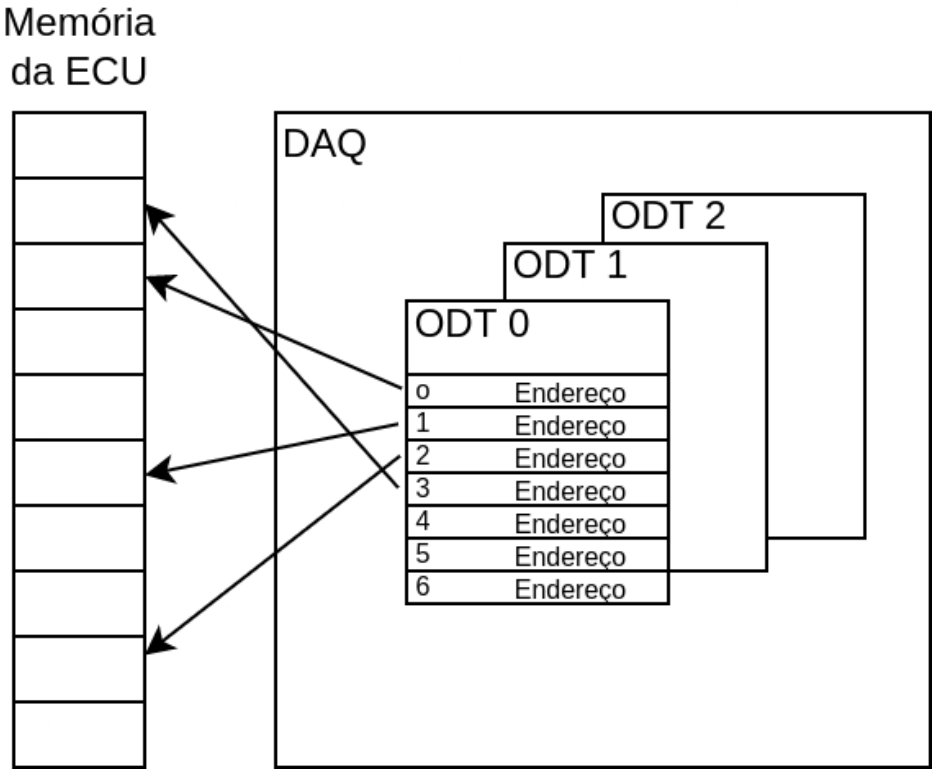
de erro.

Para configurar uma variável a ser aquisitada, primeiro deve-se escolher qual DAQ list e ODT vai armazenar seu endereço, usando o comando SET_DAQ_PTR. Então, o endereço e tamanho da variável deve ser informada usando o comando WRITE_DAQ. Esse processo se repete até que todas as variáveis tenham sido configuradas. Por fim, sucessivos comandos START_STOP são usados para iniciar a aquisição das ODTs.

3.3.2 Calibração

O CCP suporta calibração de variáveis utilizando o comando UPLOAD. Para isso, o comando SET_MTA deve ser usado para inicializar um ponteiro no endereço de

Figura 2 – Uma DAQ list com três ODTs



Fonte: Adaptado de ASAP (1999)

memória referente ao parâmetro que se deseja calibrar, e então o comando UPLOAD é enviado, contendo o novo valor do parâmetro.

O procolo também suporta múltiplas páginas de calibração, contendo os mesmos parâmetros com valores possivelmente diferentes. A ECU comumente conta com duas páginas de calibração, uma armazenada na EEPROM, chamada de Reference Page, e outra em RAM, chamada de Working Page. Ao ligar a ECU, os valores da Reference Page são copiados para a Working Page, e a calibração deve ser realizada nessa página.

Para alterar a página de calibração selecionada, os comandos SET_MTA seguido de SELECT_CAL_PAGE devem ser utilizados. A página deve ser selecionada antes das calibrações, de forma a garantir que os parâmetros sejam calibrados na página desejada. Adicionalmente, a página ativa pode ser descoberta usando o comando GET_ACTIVE_CAL_PAGE.

3.4 UNIVERSAL MEASUREMENT AND CALIBRATION PROTOCOL (XCP)

O protocolo XCP é o sucessor do CCP, oferecendo um protocolo melhorado e mais genérico. A principal melhoria em relação ao CCP se deve à generalização do protocolo de transporte, permitindo suporte aos protocolos CAN, FlexRay, Ethernet,

SPI, SCI e USB. O protocolo XCP não é compatível com CCP, mesmo se usado o protocolo de transporte CAN (ASAM, 2003a).

A utilização de alguns protocolos de transporte como Ethernet implicam em maior largura de banda e taxa de transferência, permitindo que mais variáveis possam ser aquisitadas por experimento, contanto que os dispositivos conectados suportem a velocidade (ASAM, 2003a).

Assim como seu antecessor, XCP suporta transferência assíncrona de dados com o mesmo formato de ODT, DAQ List e rasters, páginas de calibração e calibração durante o experimento. Adicionalmente, a especificação define outros recursos como o início de experimentos junto com a energização do escravo, programação da memória flash da ECU, transferência de comandos em bloco e estimulação (STIM), um recurso que utiliza DAQ Lists para realizar calibração ao invés de coleta de dados (ASAM, 2003a).

Os comandos utilizados para estabelecer conexão, configurar as listas de aquisição e calibrar são os mesmos usados pelo protocolo CCP, porém com diferentes PID e campos ligeiramente diferentes (ASAM, 2003b). Portanto, a mesma sequência de comandos necessária para se utilizar ECUs que suportam CCP se aplica para ECUs XCP.

Diferente do CCP, que registra nas mensagens um contador para identificar qual o comando referente à resposta recebida, o XCP não possui tal contador. Devido a isso, é especificado que um novo comando somente deve ser enviado após o recebimento de uma resposta. O protocolo define também que os dispositivos conectados possam suportar outros dois tipos de transferência de mensagens, em bloco e intercalada, incompatíveis entre si (ASAM, 2003a).

Envios em blocos somente recebem uma mensagem de resposta independente do número de comandos enviados. Esse modo é útil para agilizar grandes transferências de dados, por exemplo quando é realizada a programação da memória da ECU. Já o envio intercalado permite que novos comandos sejam enviados antes da resposta do comando anterior ser recebida, similar ao funcionamento do CCP (ASAM, 2003a).

Mensagens XCP são estruturadas em três partes, o cabeçalho, o pacote e a cauda. O cabeçalho e a cauda dependem do protocolo de transporte utilizado e basicamente consistem de um campo de controle. Já o pacote contém a parte genérica da mensagem, com os dados que se desejam transmitir. O pacote dispõe de um campo opcional na qual pode ser enviado o timestamp da mensagem, além dos campos obrigatórios de identificação e dos dados (ASAM, 2003a).

3.4.1 XCP on CAN

Devido aos limites do protocolo CAN, na qual a mensagem deve ser composta de no máximo 8 bytes, a especificação do XCP que utiliza CAN, chamada de XCP on

CAN, define que a mensagem não deve possuir cabeçalho e a cauda deve ser usada somente para completar os bytes não utilizados (ASAM, 2003c).

Além disso, o campo timestamp também não é utilizado. Portanto, as mensagens enviadas são muito similares aos enviados no CCP, com pequenas variações nos comandos, como comentado na seção anterior.

3.5 ASAM MCD-2 MC

ASAM MCD-2 MC é um padrão mantido pela ASAM que especifica o conteúdo de arquivos *a2l*, responsável por descrever as variáveis e características da ECU. Esse arquivo contém informações sobre o meio de comunicação usado pela ECU, além de informações sobre todos os rasters, variáveis e parâmetros (ASSOCIATION FOR STANDARDISATION OF AUTOMATION AND MEASURING SYSTEMS, 2022).

Um dos campos presentes no arquivo a2l é chamado de IF_DATA e descreve as características de funcionamento do protocolo de comunicação utilizado. Se a ECU suportar comunicação CCP, esse campo lista todos os rasters configurados e suas características, como o primeiro PID do raster e seu número máximo de ODTs. Além disso, também dispõe de informações sobre as páginas de calibração disponíveis, a velocidade de comunicação da ECU e os comandos opcionais suportados (ASAM, 2010).

Todos os parâmetros de calibração estão listados dentro do campo CHARACTERISTIC e todas as variáveis que se pode medir são listadas no campo MEASUREMENT. Os campos de informação contidos em cada parâmetro e variável são, em sua maioria, iguais. Dentre essas informações, as mais importantes são o nome, endereço de memória, tamanho, código de conversão, máscara de bits e os limites superior e inferior do valor que a variável pode tomar.

3.5.1 Conversão dos valores

Os valores enviados e recebidos pela ECU, chamados de valores internos, são compostos de um número definido de bytes e podem ser interpretados como inteiros com e sem sinal, e também como números racionais no formato float. Conversões são então utilizadas para transformar os valores internos em valores físicos, de fácil interpretação pelo usuário do sistema de medição e calibração.

As conversões estão listadas no campo COMPU_METHOD e possuem diferentes tipos. Se o tipo da conversão for IDENTICAL, o valor não precisa ser convertido. Caso seja LINEAR, a conversão tem a forma apresentada na Equação 1.

$$f(x) = ax + b \tag{1}$$

Em que *x* é o valor interno da ECU, *f(x)* é o valor físico e os coeficientes *a* e *b* são encontrados em COMPU_METHOD. Se a conversão for do tipo RAT_FUNC, a conversão utiliza uma função racional mostrada na Equação 2.

$$f(x) = \frac{ax^2 + bx + c}{dx^2 + ex + f} \tag{2}$$

Em que *f(x)* é o valor interno e *x* é o valor físico. Portanto, devido ao fato de que essa função transforma valores físicos em internos, os coeficientes devem ser definidos de forma que a função seja inversível. Se o tipo for FORM, a fórmula de conversão é especificada no campo FORMULA do método de conversão (ASAM, 2010).

Além disso, a conversão pode ser realizada usando tabelas. Se o tipo da conversão for TAB_INTP ou TAB_NO_INT, a tabela converte os valores para números usando interpolação dos pontos ou não, respectivamente. Caso o tipo seja TAB_VERB, os números são convertidos em texto, e são usadas para casos em que os números indicam um estado, como verdadeiro e falso, ou ligado e desligado.

4 MATERIAIS E MÉTODOS

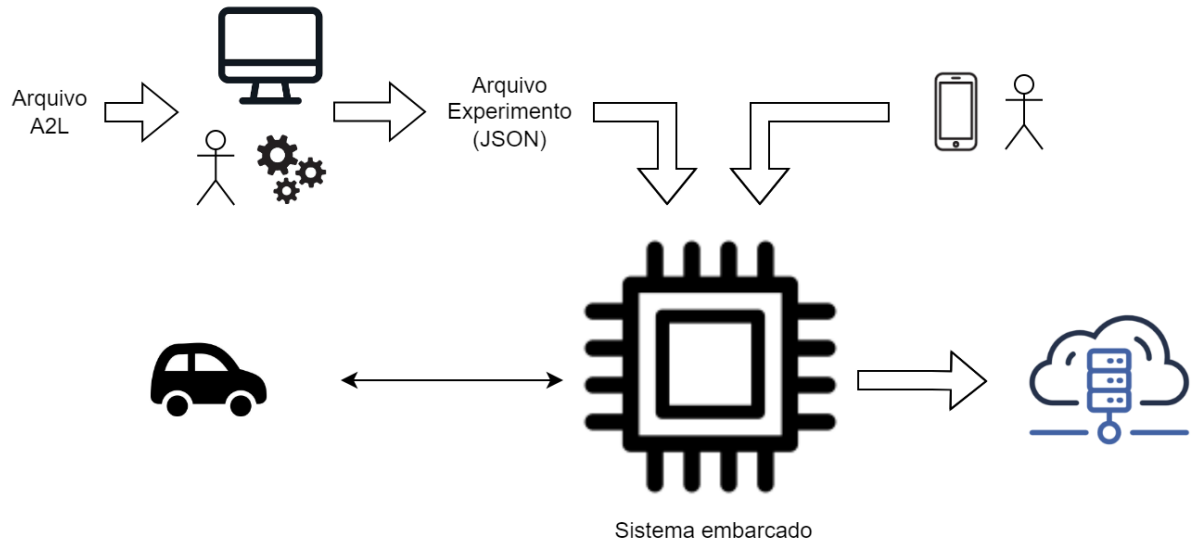
Neste capítulo serão apresentados os requisitos especificados para o sistema, o hardware escolhido e desenvolvido, bem como o servidor usado.

4.1 REQUISITOS

O fluxo de trabalho completo é composto de outros elementos além do sistema embarcado. Primeiramente, o arquivo A2L de configuração da ECU deve ser importado por um programa executando no qual um usuário pode configurar o experimento. Esse programa deve produz um arquivo que deve ser carregado no hardware embarcado, que deve configurar o experimento, coletar os dados e os enviar para o servidor.

Adicionalmente, um usuário deve poder iniciar e parar o experimento, e realizar calibração utilizando um aplicativo de celular. A Figura 3 mostra a ideia geral do sistema completo.

Figura 3 – Representação da ideia geral do sistema completo



Fonte: Autor (2022)

De forma a nortear o desenvolvimento do sistema, foram estabelecidos requisitos funcionais e não funcionais que devem ser cumpridos visando o correto funcionamento do sistema, apresentados a seguir.

4.1.1 Requisitos funcionais

- RF1: O sistema deve ser capaz de adquirir dados e calibrar usando o protocolo CCP ou XCP on CAN;

- RF2: O processamento deve ser rápido o suficiente para realizar a aquisição e processamento dos dados em tempo real, de forma que não acumule dados oriundos da ECU;
- RF3: O sistema deve possuir armazenamento suficiente para o equivalente a um dia de coleta de dados;
- RF4: O sistema deve ser capaz de enviar dados coletados para um servidor em tempo real;
- RF5: Deve haver uma interface na qual é possível visualizar o estado do sistema e realizar calibração;
- RF6: O sistema deve ser capaz de medir sua localização usando o sistema de posicionamento global (GPS);
- RF7: O experimento deve ser configurado utilizando um arquivo JSON;
- RF8: Os dados aquisitados devem ser armazenados e enviados no formato SmartData (LABORATóRIO DE INTEGRAçãO SOFTWARE/HARDWARE, 2022).

4.1.2 Requisitos não funcionais

- RNF1: O hardware deve possuir uma interface para cartão SD (memória flash);
- RNF2: Um cartão SD de no mínimo 8 GB deve ser usado;
- RNF3: O sistema deve ser alimentado através da bateria do carro;
- RNF4: O sistema deve caber no porta-luvas do carro;
- RNF5: A transmissão dos dados para o servidor deve ser realizada através de um módulo 4G;
- RNF6: A interface deve ser um aplicativo de celular para o sistema operacional Android;
- RNF7: A comunicação com a interface deve utilizar o protocolo Bluetooth Low Energy (BLE);
- RNF8: Um módulo ESP32 deve ser usado para a comunicação BLE;
- RNF9: O software deve ser escrito em C++, visando desempenho;
- RNF10: Quando não houver conexão 4G, os dados devem ser armazenados localmente;
- RNF11: Um módulo EC25 deve ser usado para comunicação 4G e GPS;
- RNF12: O tempo médio entre leituras consecutivas deve ser no máximo 100 microssegundos a mais ou a menos do que o período do raster para todas as variáveis;
- RNF13: O consumo do processador e memória deve estar abaixo de 80% na média;

4.2 FZ3

A placa de desenvolvimento FZ3 produzida pela MYIR Tech Limited é baseado no sistema em um chip (SoC) Zynq UltraScale+ ZU3EG, da Xilinx (MYIR TECH LIMITED, 2022). A placa possui um processador ARM Cortex-A53 de quatro núcleos, um processador de tempo real ARM Cortex-R5 de 2 núcleos, uma placa de vídeo integrada Mali400 e um arranjo de porta programável em campo (FPGA). Entre outras características, destaca-se a presença de um leitor de cartão SD, 4 GB de memória a uma frequência de 2400 MHz e suporte às interfaces CAN, General Purpose Input/Output (GPIO), Universal Asynchronous Receiver/Transmitter (UART) e Universal Serial Bus (USB). FZ3 suporta o sistema operacional PetaLinux 2020.1 e seu hardware pode ser configurado usando a ferramenta Vitis, também da Xilinx (MYIR TECH LIMITED, 2022).

A placa FZ3 foi escolhida devido ao suporte às interfaces CAN, UART e USB, usadas pela ECU, módulo BLE e 4G, respectivamente. Além disso, os múltiplos núcleos de processamento permitem que o software desenvolvido possa realizar múltiplas tarefas em simultâneo, característica essencial devido a grande quantidade de dados recebidos e a necessidade de processá-los em tempo real, facilitando o cumprimento do requisito RF2. Por possuir um leitor de cartão SD, a placa de desenvolvimento cumpre o requisito RNF1, permitindo que o armazenamento seja expandido. Por fim, a presença de uma interface CAN possibilita o cumprimento do requisito RF1.

4.3 ESP32-WROOM-32D

O módulo ESP32-WROOM-32D conta com um microcontrolador de dois núcleos capaz de utilizar as tecnologias sem fio Wi-Fi, Bluetooth e BLE, suporta frequência de operação variável na faixa de 80 MHz a 240 MHz e é capaz de realizar tarefas de alta demanda do processador, como codificação e decodificação de áudio. Integrado ao módulo há uma antena, utilizado para comunicação sem fio. A fonte de alimentação deve fornecer entre 3,0V e 3,6V, sendo o consumo de corrente máximo de 500 mA (ESPRESSIF SYSTEMS, 2022).

A escolha do módulo ESP32 se deve a diversos fatores. Em primeiro lugar, o protocolo de comunicação serial UART, suportado pelo módulo, permite que a comunicação entre o módulo e a placa de desenvolvimento seja estabelecida. Em segundo lugar, o protocolo BLE permite que dados sejam enviados e recebidos de um celular no qual podem ser configurados os experimentos e realizado calibração, cumprindo o requisito RNF7. Por fim, o protocolo Wi-Fi pode ser utilizado como uma opção secundária para o envio de dados, caso algum problema ocorra com o envio através do módulo 4G.

4.4 EC25-AU

O módulo EC25-AU é o modelo da família EC25 da fabricante Quectel destinado ao mercado da América Latina, que utiliza a tecnologia Long Term Evolution (LTE), conhecida no Brasil como 4G. Além dessa tecnologia, o EC25 também oferece suporte a diversas constelações do sistema de navegação por satélite (GNSS), permitindo que seja possível determinar sua posição atual, velocidade, horário e data. O módulo oferece um conector Mini PCIe, usado para se conectar ao leitor de chip de celular, ao dispositivo que utilizará a conexão de internet, a dispositivos que utilizam protocolos seriais de comunicação, entre outros (QUECTEL WIRELESS SOLUTIONS CO., LTD, 2022b).

Além do conector PCIe, o módulo também oferece 3 conexões de antena, uma antena principal, uma antena GNSS e uma antena para diversidade de antenas. A fabricante indica a utilização de antenas GNSS que operam na faixa de frequência entre 1559 MHz e 1609 MHz, e indica que a antena 4G deve ser adequada para o país no qual ela vai ser empregada (QUECTEL WIRELESS SOLUTIONS CO., LTD, 2022a). Utilizando as informações sobre as frequências de operação das antenas das torres de internet 4G no município de Joinville (ANATEL, 2022), foi gerada a Tabela 1.

Tabela 1 – Antenas de telefonia por banda de frequência no município de Joinville

Empresa de telefonia	B28	B3	B1	B7
CLARO S.A.	23	339	0	131
OI MÓVEL S.A.	0	755	0	495
TELEFONICA BRASIL S.A.	115	68	0	149
TIM S.A.	171	439	114	456
Não identificado	6	81	6	87
Total	315	1682	120	1318

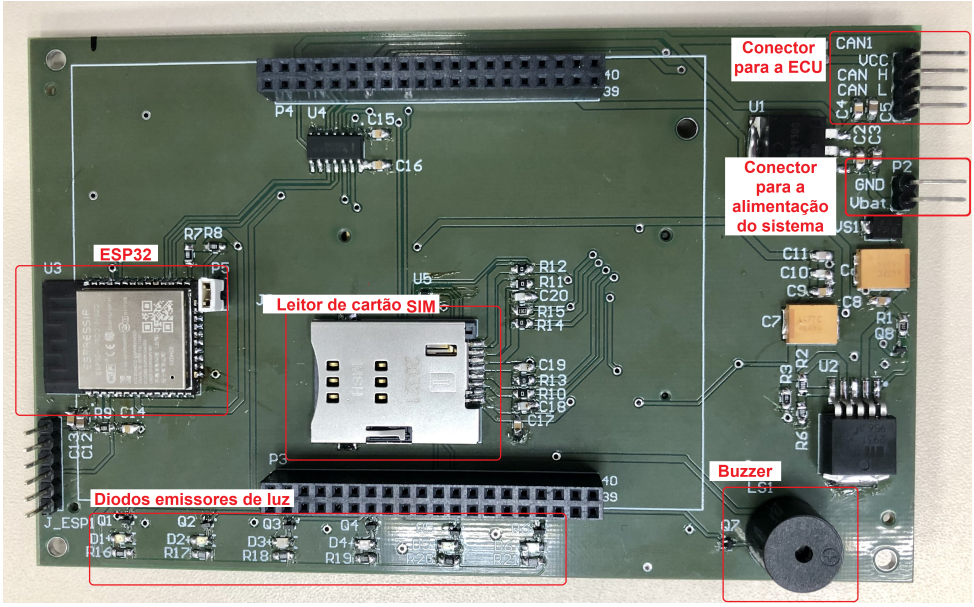
Fonte: Anatel (2022).

A partir desses dados, foi escolhida uma antena que opera na banda de frequência B3, a mais comum no município, de forma a obter sinal de telefonia na maior área possível. A utilização do módulo EC25 cumpre os requisitos RF4, RF6, RNF5 e RNF11.

4.5 HARDWARE DESENVOLVIDO

Com o objetivo de conectar os diferentes módulos à plataforma de desenvolvimento FZ3, foi desenvolvida uma placa que oferecesse as conexões necessárias, além de possuir conectores para a ECU do carro e fonte de energia. A vista superior da placa desenvolvida pode ser vista na Figura 4.

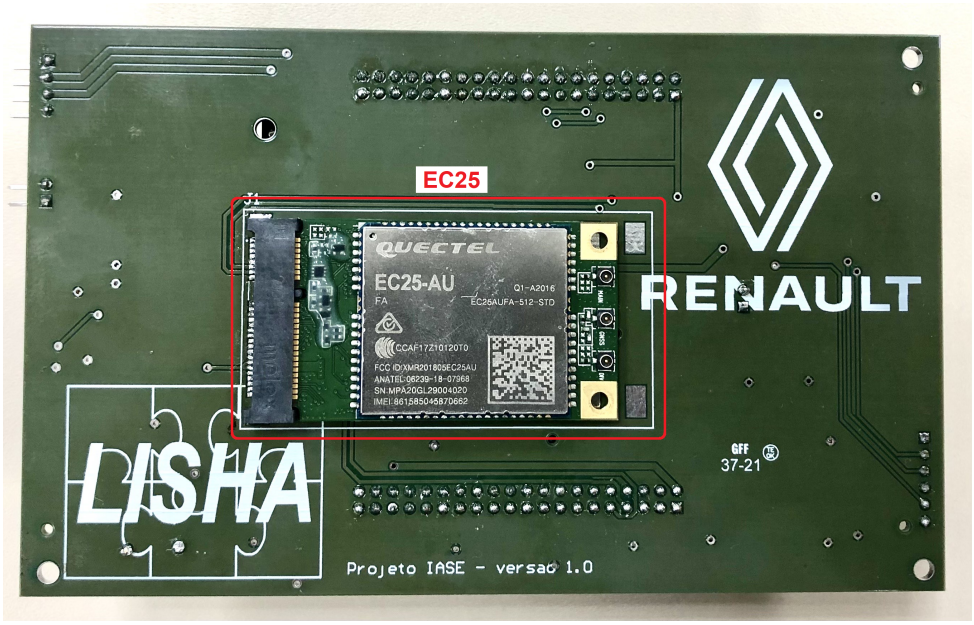
Figura 4 – Vista superior da placa desenvolvida



Fonte: Autor (2022)

Adicionalmente, foram colocados um leitor de cartão SIM, reguladores de tensão para a alimentação dos módulos periféricos, seis diodos emissores de luz (LED) e um Buzzer. A Figura 5 mostra a parte inferior da placa, onde é possível observar o conector mPCIe para o módulo EC25.

Figura 5 – Vista inferior da placa desenvolvida



Fonte: Autor (2022)

As barras de pinos fêmeas vistos na visão superior da placa são usadas para encaixar os pinos da FZ3, de forma que o hardware final possui o formato visto na

Figura 6.

Figura 6 – Hardware montado

Fonte: Autor (2022)

O hardware também foi encapsulado por uma caixa de acrílico, com furos de forma a permitir o fluxo de ar. Essa caixa de acrílico possui 95 milímetros de largura, 50 milímetros de altura e 150 milímetros de profundidade, portanto é pequena o suficiente para ser comportada no porta luvas do carro, cumprindo o requisito RNF4.

4.6 SERVIDOR E SMARTDATA

Os dados coletados foram armazenados em um servidor fornecido pelo Laboratório de Integração Hardware/Software (LISHA). Esse servidor recebe e responde pedidos usando o protocolo de transferência de hipertexto (HTTP), sendo o corpo do pedido no formato JSON e aceita dados no formato SmartData, proprietário do LISHA. Os comandos POST e GET do protocolo HTTP são suportados e podem ser usados para inserir ou consultar informações do banco de dados, respectivamente (LISHA, 2022b).

Para armazenar dados no servidor, primeiramente deve ser criado uma série de SmartData informando qual a unidade no sistema internacional (SI) referente aos dados, a versão do formato SmartData, a coordenada no sistema de coordenadas geocêntricas (em centímetros), os tempos de início e fim da série, o tipo e um identificador chamado de dev, responsável por distinguir dados com a mesma unidade e coordenadas no espaço-tempo. Além disso, devem ser informados o período (caso seja um dado aquisitado periodicamente) e o workflow, indicando por qual rotina de processamento os dados devem passar antes de serem armazenados no banco de dados (LISHA, 2022b).

A versão do formato indica se os dados coletados mudam de posição com o tempo. Portanto, para sensores e sistemas estacionários, a versão 1.1 deve ser usada, e para sensores e sistemas móveis, a versão 1.2 deve ser usada. Todas as marcas temporais devem ser passadas no formato UNIX e medidas em microssegundos. Se a coleta das variáveis for realizada periodicamente (acionada por tempo), o tipo da série deve ser time-triggered high-frequency (TTH), e se for acionada por evento, o tipo não deve ser especificado (LISHA, 2022b). A Figura 7 mostra o formato do JSON que deve ser enviado para criar uma série.

Figura 7 – Formato do JSON para criação de série

```
"Series" : Object  {
    "version" : 1.1
    "unit" : unsigned long
    "x" : long
    "y" : long
    "z" : long
    "r" : unsigned long
    "t0" : unsigned long long
    "tf" : unsigned long long
    "period" : unsigned long
    "uncertainty" : unsigned long
    "workflow" : unsigned long
}
```

Fonte: LISHA (2022b)

Devido à grande quantidade de dados a serem transmitidos em alta frequência, foi usado uma estrutura de mensagem que permite o envio de valores em lote, de tamanho configurável, chamado de MultiValueSmartData. No campo datapoints, destinado aos valores, é enviado um vetor com os valores e offset. A Figura 8 mostra o formato do corpo da mensagem quando se deseja enviar MultiValueSmartData.

Caso os dados sejam obtidos por eventos e, portanto, não houver uma periodicidade entre os dados, o campo offset deve ser passado, indicando o tempo decorrido após a última aquisição. No caso de dados periódicos, não há a necessidade de especificar o deslocamento de tempo, sendo ele calculado usando o período da variável.

Figura 8 – Formato do JSON para envio de dados em lote

```
"MultiValueSmartData" : Object {
    "version" : unsigned char
    "unit" : unsigned long
    "x" : long
    "y" : long
    "z" : long
    "t0" : unsigned long long
    "dev" : unsigned long
    "type" : char(3),                                   // 'TTH', 'TTL',  'ED'. 'OLD' assumed if not present
    "period" : unsigned long,
    "uncertainty" : unsigned long            // OPTIONAL: if given, then ommit it in data points
    "period" : unsigned long                 // OPTIONAL: if given, then ommit offset in data points
    "datapoints":  Array [
        {
            "offset : unsigned long          // OPTIONAL, not used if period is informed in the header
            "value" : double
            "uncertainty" : unsigned long    // OPTIONAL, not used if informed in the header
        }
    ]
}
```

Fonte: LISHA (2022b)

5 DESENVOLVIMENTO

Nesse capítulo serão apresentadas as configurações do hardware e a estrutura do software executado no sistema.

5.1 CONFIGURAÇÃO DO HARDWARE E SISTEMA OPERACIONAL

O SoC Zynq Ultrascale+ utilizado possui uma região do chip dedicado a ser uma FPGA, nomeado lógica programável (PL). Nessa região, os transistores podem ser programados para realizar diferentes funções a nível de hardware no software Vivado, da Xilinx. No software, UART1 foi habilitada e conectada aos pinos MIO8 e MIO9 da FZ3. Ainda, foram configurados nove pinos GPIO de saída e um pino de entrada. Após realizar a configuração no Vivado, um arquivo Xilinx Support Archive (XSA) é gerado e deve ser utilizado para criar a imagem do sistema operacional PetaLinux (LISHA, 2022c).

Uma imagem do PetaLinux pode ser criada seguindo os passos do guia de instalação (XILINX, 2020). Além de utilizar o arquivo XSA gerado, os seguintes drivers devem ser habilitados para permitir que o módulo EC25 seja reconhecido pelo sistema operacional.

- USB support
- USB modem support
- USB driver for GSM and CDMA modems
- PPP support

Além disso, para facilitar o desenvolvimento e adicionar novas funcionalidades, os seguintes pacotes devem ser instalados.

- bash
- tar
- curl
- minicom
- vim
- libusb1
- libsocketcan
- ppp

Adicionalmente, para utilizar o protocolo ponto-a-ponto (PPP) com o módulo 4G, arquivos de configuração na pasta */etc/ppp* devem ser criados ou modificados de acordo com o módulo usado e operadora de telefonia escolhido.

Para inicializar a conexão com o barramento CAN, os seguintes comandos devem ser executados toda vez que o sistema é ligado.

```
$ ip link set can0 type can bitrate 1000000
$ ifconfig can0 up
$ ip link set dev can0 txqueuelen 10000
```

Os dois primeiros comandos configuram a conexão com baud rate de 1 Mbps e a iniciam. O terceiro comando configura o tamanho da fila de envio do sistema para 10 mil.

5.2 MÁQUINAS DE ESTADOS

Buscando facilitar a programação do sistema e a divisão das tarefas entre os núcleos do processador, foram utilizadas máquinas de estados independentes entre si. Os estados da máquina podem ser alterados através de eventos e diferentes funções podem ser chamadas na entrada, durante e saída de um estado. Caso seja necessário transferir informações entre máquinas de estado diferentes, estruturas como filas, listas e pilhas podem ser utilizadas. As máquinas de estado foram nomeadas Main, DataController e ECUReader. A Figura 9 mostra o diagrama contendo somente as classes que controlam as máquinas de estado.

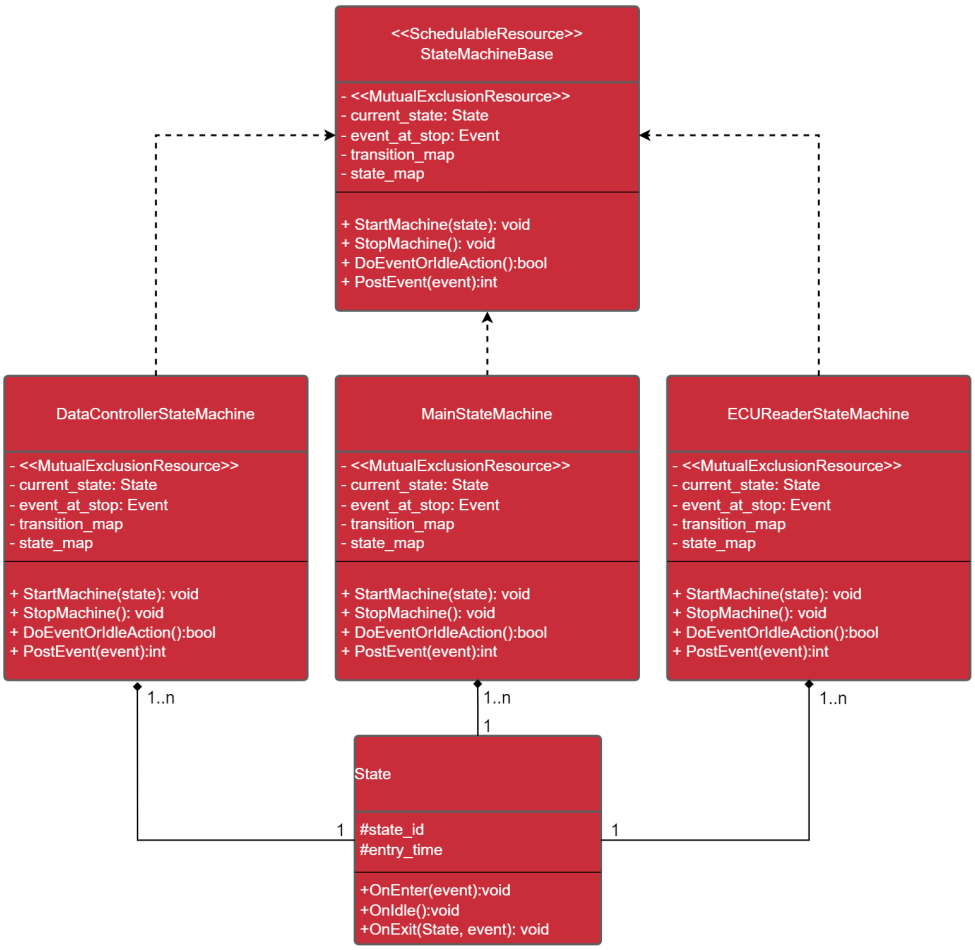
Todas as classes que controlam as máquinas de estado herdam da classe StateMachineBase, que declara os principais métodos virtuais usados pelas máquinas. Adicionalmente, os estados das máquinas herdam da classe State e implementam os métodos OnEnter, OnIdle e OnExit, executados quando a máquina entra no estado, enquanto está no estado e quando sai do estado, respectivamente.

5.2.1 Máquina de estados Main

A máquina de estados Main é a primeira a ser criada e é a responsável por criar os objetos das outras máquinas de estados. Além disso, também controla as interfaces com o módulo ESP32 e GPIO. Por fim, também controla o status do sistema, informando o usuário de eventuais problemas através dos LEDs e do aplicativo. A Figura 10 mostra o diagrama de classes com todas as classes relacionadas à máquina de estados Main.

Durante a inicialização da classe Main, um objeto da classe CircularBuffer é criado. Ele é um buffer cíclico com exclusão mútua, compartilhado pelas classes ECUReader e DataController e responsável por armazenar os dados coletados

Figura 9 – Diagrama de classes com as classes que controlam a máquina de estados



Fonte: Autor (2022)

enquanto não são processados e separados. Esse recurso foi escolhido devido a sua capacidade fixa, que permite o controle da quantidade de memória consumida, e velocidade no acesso dos dados, uma vez que os elementos contidos no buffer não precisam ser movidos a cada leitura ou escrita.

Adicionalmente, a classe possui um objeto HardwareController, que controla um objeto da classe ESP32 e um da classe GPIO. A classe ESP32 abre uma conexão UART para o módulo e se comunica usando um protocolo de comunicação próprio, no qual mensagens de 22 bytes são enviadas, sendo o primeiro byte usado para identificar o conteúdo da mensagem. A classe GPIO é responsável por configurar as saídas GPIO do Linux e controlar os LEDs e Buzzer, além de possibilitar a reinicialização do módulo 4G ao desligar sua alimentação. Os LEDs foram usados para indicar o estado atual do sistema e podendo alertar o usuário de possíveis erros. Já o Buzzer foi usado para indicar uma troca de estados do sistema e avisar potenciais erros.

Essa máquina possui os seguintes estados:

1. Init: Primeiro estado da máquina, somente troca o estado para Prep_ECUReader;
2. Prep_DataController: Ativa a máquina de estados DataController e altera o estado

Figura 10 – Diagrama das classes relacionadas à máquina de estados Main

Fonte: Autor (2022)

para Prep_ECUReader;

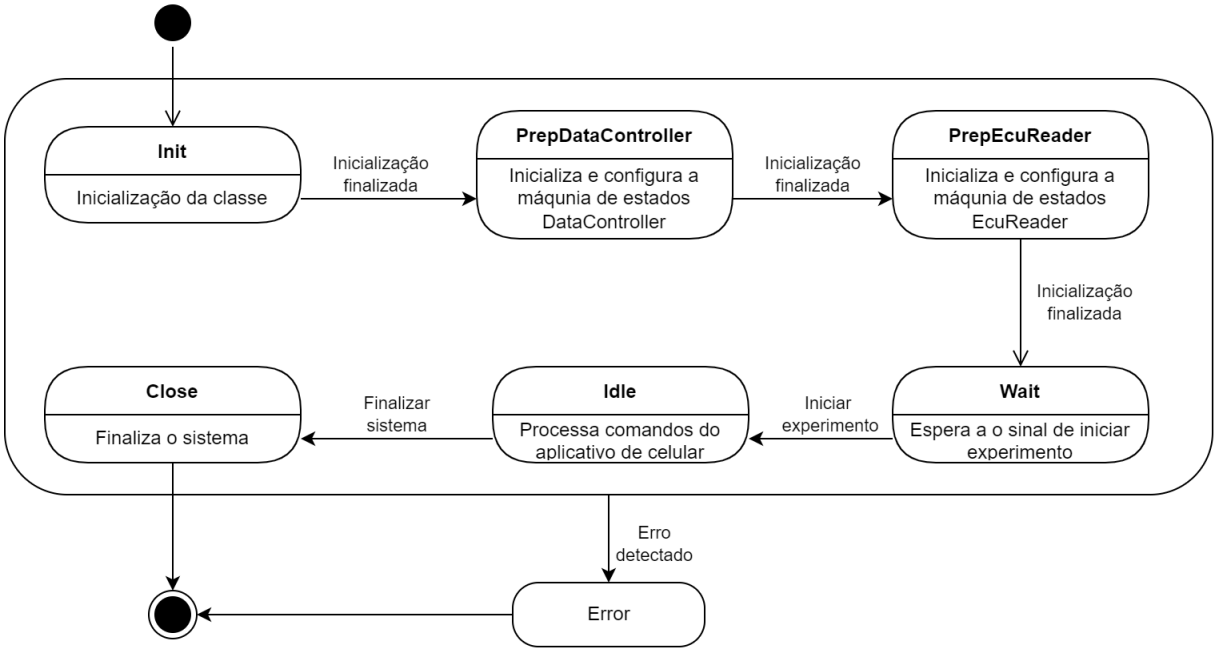
3. Prep_ECUReader: Ativa a máquina de estados ECUReader e espera receber um sinal de iniciar experimento do aplicativo para mudar para o estado Idle;
4. Wait: Espera receber um sinal de iniciar do aplicativo para mudar para o estado Idle;
5. Idle: Estado principal do sistema, troca mensagens com o módulo ESP32

e, consequentemente, com o aplicativo de celular. Recebe comandos, envia respostas e status para o usuário;

6. Error: Estado que indica a ocorrência de um erro;
7. Close: Envia sinais para parar as outras máquinas e espera até que elas sejam finalizadas.

A Figura 11 mostra o diagrama de estados dessa máquina de estados.

Figura 11 – Diagrama de estados da máquina de estados Main



Fonte: Autor (2022)

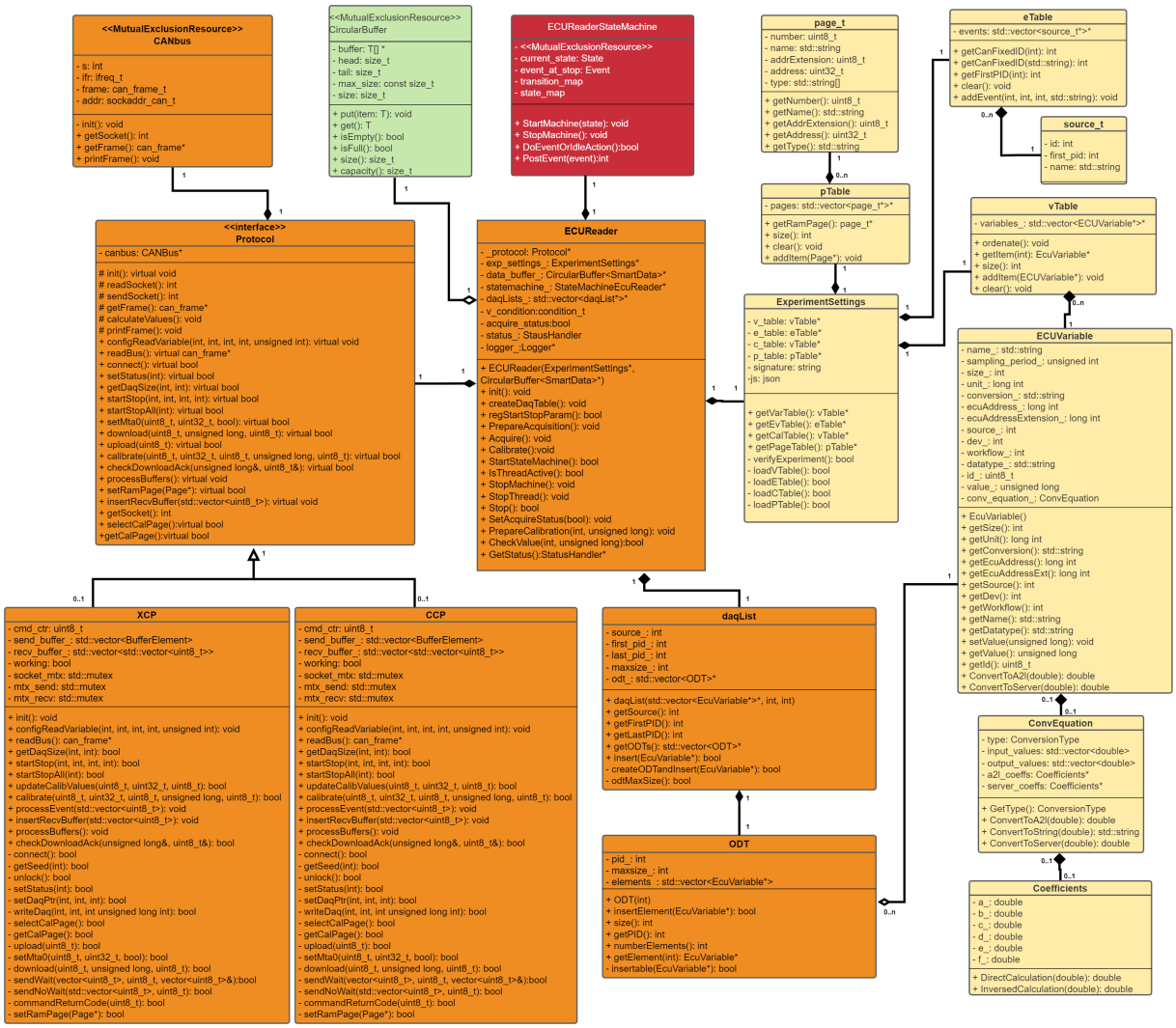
A máquina de estados Main permanece no estado Idle enquanto o experimento está executando, ou seja, durante a maior parte do tempo em que o sistema está ligado. As transições dessa máquina de estado dependem, em sua maioria, dos comandos recebidos do aplicativo do celular, e causam mudanças em outras máquinas.

5.2.2 Máquina de estados ECUReader

A principal função da máquina de estados ECUReader é realizar a comunicação com a ECU. Ela é responsável por ler o arquivo de configuração, configurar o experimento, receber os dados enviados pela ECU e a calibrar. A Figura 12 mostra o diagrama de classes relacionado a essa máquina.

No processo de inicialização da máquina, o arquivo JSON que contém as informações do experimento é lido e quatro tabelas são criadas e preenchidas. São elas as tabelas de páginas de calibração, eventos da ECU, variáveis de medição e

Figura 12 – Diagrama das classes relacionadas à máquina de estados ECUReader



Fonte: Autor (2022)

variáveis de calibração. Cada uma das tabelas possui um vetor contendo os elementos da tabela. Adicionalmente, é realizada a validação das configurações carregadas, a fim de evitar inconsistências.

Todas as variáveis de medição e parâmetros de calibração são armazenadas nas tabelas como objetos ECUVariable, contendo informações como seu nome, endereço em memória, tamanho ocupado e como converter os dados entre valores internos da ECU e valores com unidades reais. Para variáveis de medição, também são carregados informações sobre a frequência de medição e identificadores que devem ser enviados para o servidor. Além disso, parâmetros de calibração são passados com um número identificador, usado na comunicação com o aplicativo de celular.

Os rasters são carregados como objetos do tipo source_t e informam seu nome, um número identificador e qual o primeiro PID correspondente a ele. Adicionalmente, também é informado o comprimento do raster, ou seja, o número de ODTs que

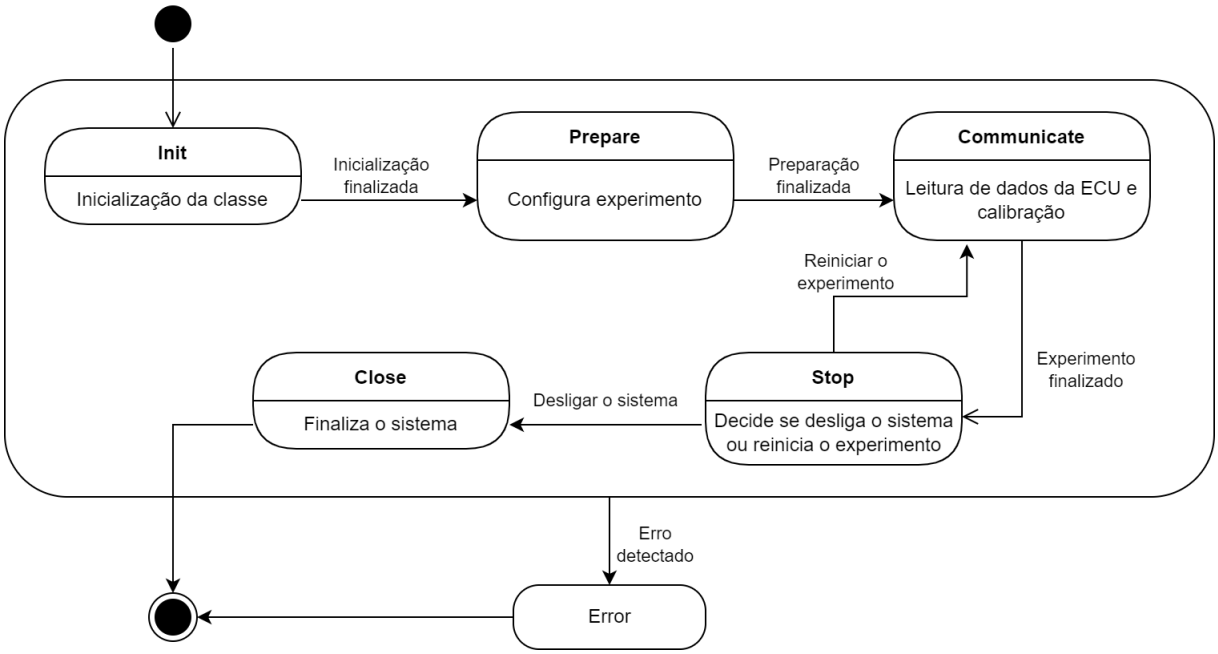
ele comporta, porém essa informação somente é usada para validação durante o carregamento do arquivo de configuração e portanto não é salvo. As páginas de calibração disponíveis também são carregadas, informando seu nome, tipo e endereço de memória, e são armazenadas em uma tabela.

Essa máquina possui os seguintes estados:

1. Init: Primeiro estado da máquina, somente altera o estado para Prepare;
2. Prepare: Configura todas as variáveis em suas ODTs e DAQ Lists, e inicia o envio de dados pela ECU, também troca para o estado Communicate;
3. Communicate: Cria duas threads para executar aquisição e calibração em paralelo, e fica esperando as threads terminarem, trocando o estado para Stop;
4. Stop: Pode reiniciar o experimento ou pará-lo, dependendo da escolha do usuário;
5. Error: Para a execução da máquina caso algum erro ocorra;
6. Close: Para a execução da máquina.

A Figura 13 mostra o diagrama de estados dessa máquina de estados.

Figura 13 – Diagrama de estados da máquina de estados EcuReader



Fonte: Autor (2022)

Durante a maior parte do tempo, a máquina está no estado Communicate, em que uma das threads está constantemente lendo mensagens recebidas da ECU e a outra está esperando comandos do aplicativo de celular para enviar mensagens para a ECU.

5.2.2.1 Configuração da aquisição

Para estabelecer comunicação com a ECU, primeiramente deve ser enviado um comando CONNECT, seguido de um comando GET_SEED informando o recurso que se deseja desbloquear. A resposta recebida deve ser usada pra calcular uma chave, que deve ser enviada com um comando UNLOCK. O processo deve ser repetido até que todos os recursos necessários sejam desbloqueados.

Após carregar as informações do arquivo de configuração, as variáveis devem ser alocadas em suas devidas ODTs na ECU. Para isso, primeiramente são enviados comandos GET_DAQ_SIZE para todos os rasters a fim de limpar sua configuração anterior. Em seguida, os comandos SET_DAQ_PTR e WRITE_DAQ devem ser enviados para cada uma das variáveis, respeitando a configuração estabelecida nos objetos das classes daqList e ODT.

Para iniciar o envio de dados por parte da ECU, comandos START_STOP devem ser enviados para cada uma das DAQ Lists. Adicionalmente, o mesmo comando pode ser enviado para somente selecionar a lista e, posteriormente, o comando START_STOP_ALL pode ser usado para iniciar todas as listas simultaneamente.

5.2.2.2 Aquisição

A partir do momento em que o experimento é iniciado, a thread responsável por ler as mensagens recebidas da ECU monitora constantemente o barramento CAN e recebe todas as mensagens recebidas. Primeiramente, é identificado se a mensagem recebida é correspondente a uma resposta de um comando enviado ou a um dado transmitido. Caso seja uma resposta, a mensagem é salva em um vetor interno, para posterior processamento.

No caso de um dado, as variáveis correspondentes ao conteúdo da mensagem são identificadas usando o PID. Os dados são então convertidos para unidades compatíveis com o servidor, o horário e posição são registrados e são criados SmartDatas para cada um. As SmartDatas criadas são então armazenadas no CircularBuffer.

5.2.2.3 Calibração

Antes de realizar calibrações, a página de calibração deve ser escolhida, de forma a não alterar os parâmetros definidos pela fabricante do carro. Dessa forma, as alterações realizadas são sobrescritas pela configuração de fábrica toda vez que o veículo é ligado, tornando o experimento mais seguro. Para trocar a página, o comando SET_MTA deve ser usado passando o endereço de memória da página desejada seguido de SELECT_CAL_PAGE.

Com a página correta selecionada, SET_MTA seguido de DNLOAD pode ser usado para calibrar um parâmetro da ECU. Como os sucessivos comandos dependem da resposta da ECU e a calibração pode ser realizada enquanto a ECU está enviando dados constantemente, uma lista de comandos enviados é mantida de forma a identificar qual comando corresponde à resposta recebida.

5.2.3 Máquina de estados DataController

A máquina de estados DataController é responsável por agrupar os dados recebidos pela ECUReader por variável, gerar os pedidos HTTP, fazer o envio para o servidor, medir a posição e armazenar os dados localmente. A Figura 14 mostra o diagrama de classes relacionado à máquina de estados DataController.

A classe EC25 herda das classes GPS e ServerConnection, interfaces usadas para utilizar funcionalidades de localização e de envio de mensagens ao servidor, respectivamente. Caso sejam usados módulos separados, classes diferentes podem ser implementadas para cada uma das funcionalidades.

A conexão com a internet foi realizada utilizando comandos AT, um protocolo de comunicação de texto para modems, ou utilizando a biblioteca cpr (SAUTER; TRAUB, 2022), que implementa uma interface de programação de aplicativos (API) baseado na biblioteca libcurl, capaz de enviar mensagens e arquivos utilizando uma variedade de protocolos de internet.

Utilizar comandos AT possui a vantagem de permitir maior controle sobre o processo de envio dos pedidos HTTP, porém existem dois modos diferentes quando utilizando tais comandos, o modo de dados e o modo de comandos, e alternar entre eles consome em torno de 300 milissegundos. Esse custo de tempo adicional faz com que não seja viável utilizar os comandos AT para enviar pedidos com alta frequência.

Para utilizar a biblioteca cpr, o protocolo PPP deve ser configurado no sistema operacional, de forma que o modem 4G é reconhecido como uma conexão de internet padrão, assim como uma conexão por cabo. Após isso, a API pode ser usada para criar os pedidos HTTP e os enviar. Essa é a forma principal de se enviar os dados coletados, devido ao seu desempenho e facilidade de uso.

Para obter as coordenadas atuais do sistema, o comando AT+QGPSLOC é enviado para o módulo EC25, retornando a latitude, longitude e altitude. Devido ao fato de que o servidor suporta somente o sistema de coordenadas geocêntrica, as coordenadas recebidas devem ser convertidas para esse sistema, com a unidade de distância em centímetros. A função de conversão foi adaptada do código apresentado em Rose (2014).

Durante sua execução, a principal função da máquina é ler as SmartDatas

Figura 14 – Diagrama de classes relacionado à máquina de estados DataController

Fonte: Autor (2022)

armazenadas no CircularBuffer e as organizar em suas respectivas variáveis em um mapa. Os dados se acumulam até que uma certa quantidade é atingida ou uma quantia de tempo se passa, e os dados são agrupados em uma MultiValueSmartData e enviados para o servidor. Os limites para acionar o envio podem ser dimensionados de acordo com a frequência de aquisição.

Através de métodos empíricos, foi observado que ao estipular o limite de 25 mil amostras, não havia o acúmulo de MultiValueSmartDatas a serem enviados, mesmo em situações nas quais a conexão à internet era fraca. Aumentar esse limite implica

em um maior acúmulo de dados em memória, com o benefício de eliminar o possível gargalo no envio para o servidor. Um limite de 50 mil permitiu que não houvesse esse acúmulo de MultiValueSmartData esperando o envio sem consumir uma quantidade considerável de memória.

Para variáveis de menor frequência de aquisição, atingir o limite estipulado requer muito tempo, o que atrapalharia a visualização dos dados no servidor simultaneamente. Por exemplo, considerando um limite de 25 mil amostras, uma variável amostrada a cada 100 milissegundos precisaria de mais de 40 minutos para atingir o limite. Por esse motivo, também foi estipulado um tempo limite para acumular as amostras, de cinco minutos.

Outra função dessa máquina de estado é comprimir e salvar em disco todas as MultiValueSmartData, separando as que foram enviadas para o servidor e as que não foram. A separação é necessário quando a conexão a internet é perdida ou o módulo 4G apresenta mal funcionamento, permitindo ao sistema enviar os dados posteriormente. Dessa forma, os arquivos comprimidos também podem servir como um backup caso seja necessário.

Toda vez que o sistema inicia, é verificado se existem dados que não foram enviados anteriormente e então é realizada a descompressão de tais dados e seu envio é tentado novamente. Para realizar a compressão e descompressão dos arquivos, foi utilizado a biblioteca zlib (GAILLY; ADLER, 2022), que comprime as mensagens no formato JSON em arquivos binários, de tamanho máximo controlado pelo sistema.

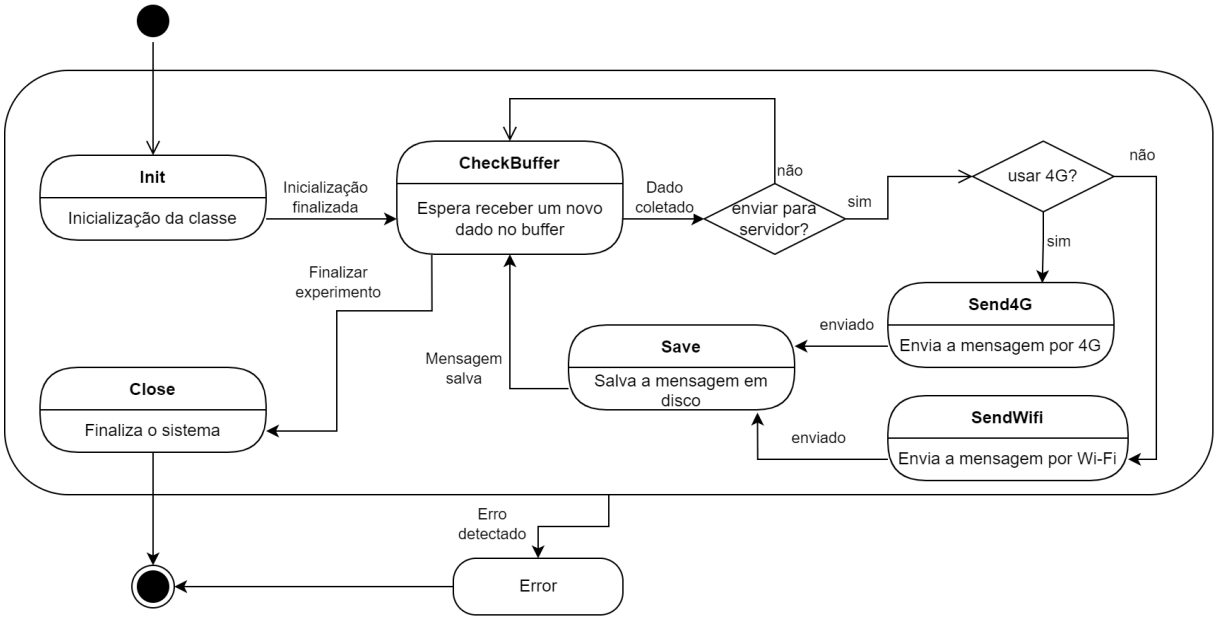
Essa máquina possui os seguintes estados:

1. Init: Primeiro estado da máquina, somente altera o estado para Check_buffer;
2. Check_buffer: Retira um elemento do CircularBuffer e o coloca no mapa de variáveis, caso houver um MultiValueSmartData para enviar, troca o estado para Check_connection;
3. Check_connection: Decide se o envio será realizado por Wi-Fi, 4G ou salvo em disco;
4. Send_wifi: Envia os dados por Wi-Fi e troca o estado para Save;
5. Send_4g: Envia os dados por 4G e troca o estado para Save;
6. Save: Salva a mensagem em disco e troca o estado para Check_buffer;
7. Close: Salva todos os dados ainda não enviados e para a máquina;
8. Error: Para a máquina devido a um erro.

A Figura 15 mostra o diagrama de estados dessa máquina de estados.

Na maior parte do tempo, a máquina alterna entra os estados Check_buffer, Check_connection e Send_4g, sendo 4G a principal forma de envio do sistema.

Figura 15 – Diagrama de estados da máquina de estados DataController



Fonte: Autor (2022)

5.3 PROGRAMA CONFIGURADOR DE EXPERIMENTO

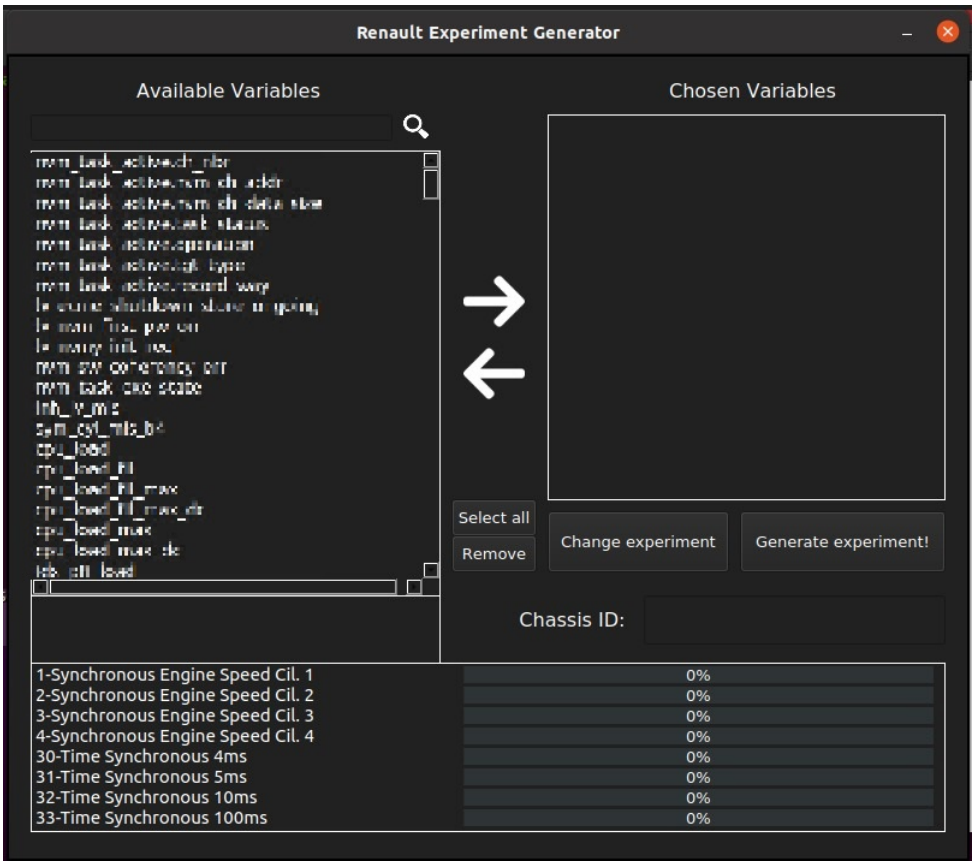
De forma a permitir que o usuário do sistema consiga facilmente criar um novo experimento, configurando quais parâmetros podem ser calibrados, quais variáveis devem ser aquisitadas e em quais frequências, foi desenvolvido pela estudante Maria Eduarda Rosa da Silva (SILVA, 2022) uma interface gráfica para dispor as possibilidades de configuração e assegurar que não haverá nenhum erro no experimento.

O software foi programado em Python e possui versões para os sistemas operacionais Windows e Linux. Primeiramente, o arquivo A2L que descreve as características da ECU deve ser convertido para o formato JSON, de mais fácil leitura. Esse arquivo pode então ser usado como entrada para o programa, e seu conteúdo é inserido nos elementos da interface. A Figura 16 apresenta a interface do software gerador de experimentos, com o nome das variáveis borrado por questões de confidencialidade.

Na aba measurements, uma lista com todas a variáveis de medição é apresentada na coluna da esquerda da interface, enquanto a coluna da direita contém as variáveis selecionadas. Na parte inferior, são apresentados os rasters e suas respectivas porcentagens de preenchimento, calculadas em tempo real. Ao trocar para a aba characteristics, podem ser selecionados os parâmetros de calibração (SILVA, 2022).

Por fim, a configuração escolhida pode ser exportada para um arquivo JSON

Figura 16 – Software gerador de experimentos



Fonte: Autor (2022)

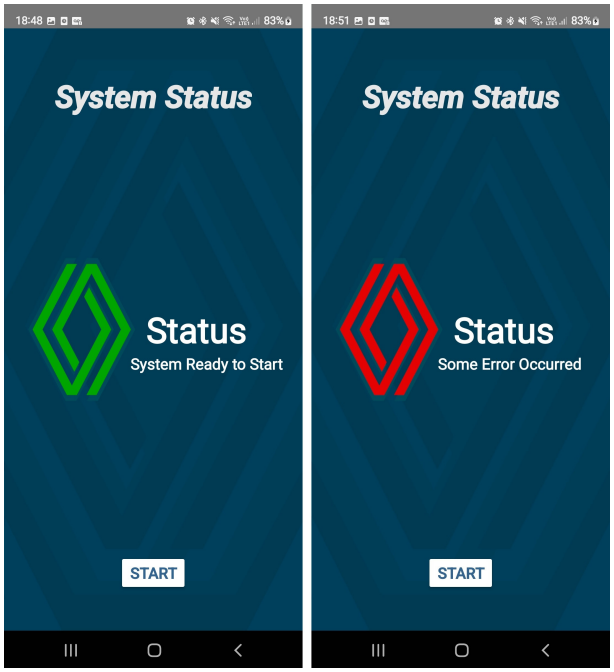
contendo todas as informações necessárias para o experimento. Esse arquivo deve ser carregado no sistema embarcado e no celular.

5.4 APLICATIVO DE CELULAR

Com o fim de monitorar e controlar a execução do sistema, um aplicativo de celular foi desenvolvido pelo estudante Miguel Suchodolak. No aplicativo, o usuário pode se identificar como um motorista ou um engenheiro. Caso se identifique como um motorista, somente é concedido acesso à tela de estado do sistema, na qual também é possível iniciar e finalizar o experimento. A Figura 17 mostra duas telas com o estado do sistema.

O estado é exibido utilizando as cores do símbolo da Renault. A cor verde indica o correto funcionamento do sistema e é nesse estado que se requer interação do usuário para iniciar ou finalizar o experimento. A cor amarela indica que o sistema está processando ou configurando algo, e que o usuário deve aguardar seu fim. Já a cor vermelha indica um erro fatal para o sistema, no qual a reinicialização é necessária.

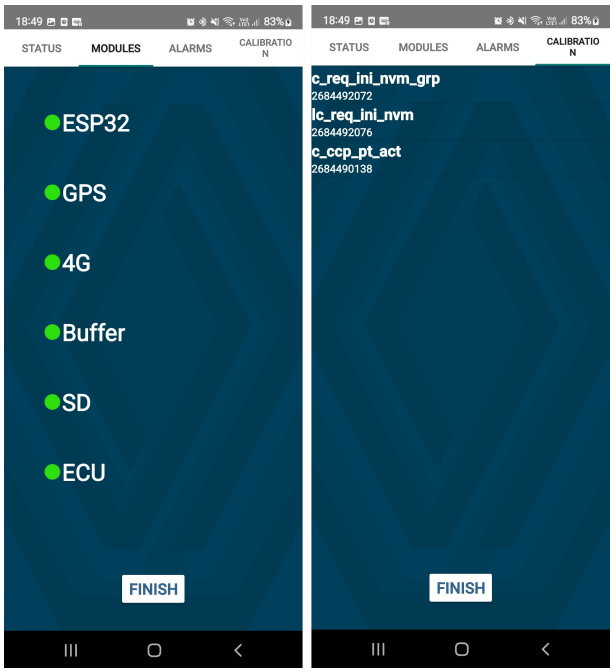
Figura 17 – Telas do aplicativo mostrando o estado do sistema



Fonte: LISHA (2022a)

Caso o usuário se identifique como um engenheiro, deverá se autenticar e recebe acesso a informações por módulo ou funcionalidade e a calibração. A Figura 18 mostra as telas de status de cada módulo e de calibração.

Figura 18 – Telas do aplicativo mostrando o status dos componentes do sistema e parâmetros de calibração



Fonte: LISHA (2022a)

Além de identificar qual funcionalidade do sistema está apresentando problemas, essa tela permite ao usuário visualizar o código do erro e sua descrição, possibilitando melhor depuração da causa do problema. Os parâmetros podem ser calibrados dentro da faixa especificada pelo arquivo de configuração, e a interação do usuário varia conforme o tipo do parâmetro.

5.5 CONFIGURAÇÃO E EXPERIMENTOS

A ECU presente no carro fornecido possuia rasters com período de quatro, cinco, dez e 100 milissegundos, além de quatro raster ativados pela posição do virabrequim, ou seja, por evento. Os rasters de evento, e os de quatro e cinco milissegundos comportam sete ODTs cada. Já os rasters de dez e 100 milissegundos comportam 15 ODTs cada.

Os parâmetros de calibração mais utilizados nos testes são relacionados à falha de ignição. O primeiro parâmetro controla a cada quantos ciclos do motor uma falha de ignição forçada vai ocorrer. Já o segundo parâmetro habilita ou desabilita as falhas forçadas. Adicionalmente, a ECU dispõe de uma variável que conta a quantidade de falhas ocorridas desde que o carro foi ligado, permitindo que seja visualizada o efeito da calibração.

6 RESULTADOS

Neste capítulo serão apresentados os resultados de testes realizados com o intuito de medir o desempenho do sistema e comprovar seu correto funcionamento.

6.1 DESEMPENHO EM DIFERENTES CONDIÇÕES

Uma importante característica de um sistema de coleta de dados é a capacidade de processar as informações recebidas o mais rápido possível, especialmente quando se utiliza protocolos de comunicação que não registram o tempo do envio no corpo da mensagem, como é o caso dos procolos CCP e XCP on CAN.

Portanto, para testar a capacidade do sistema de ler os dados assim que são recebidos, ou seja, sem que acumulem, foram criados cinco experimentos com quantidades de variáveis diferentes, simulando diferentes condições de carga do sistema e do barramento CAN. A Tabela 2 mostra os cinco experimentos criados e o número de variáveis aquisitados por cada um dos rasters.

Tabela 2 – Número de variáveis por raster de cada experimento

Total de variáveis	4ms	5ms	10ms	100ms
1	1	0	0	0
50	24	25	1	0
100	25	28	47	0
146	35	30	57	24
187	42	34	48	63

Fonte: Autor (2022).

Sendo que as variáveis são enviadas em ODTs, seu número é mais relevante para mensurar a comunicação entre a ECU e o sistema desenvolvido do que o número de variáveis. A Tabela 3 apresenta o número de ODTs por raster dos experimentos realizados.

Tabela 3 – Número de ODTs por raster de cada experimento

Total de variáveis	Total de ODTs	4ms	5ms	10ms	100ms
1	1	1	0	0	0
50	15	7	7	1	0
100	27	7	7	13	0
146	35	7	6	14	8
187	39	7	7	13	12

Fonte: Autor (2022).

Cada experimento foi iniciado e o tempo decorrido entre leituras consecutivas do barramento CAN foi registrado por cerca de 10 minutos. A Tabela 4 apresenta a média dos valores, o valor mínimo e máximo de cada um dos experimentos.

Tabela 4 – Tempo decorrido em microssegundos entre leituras consecutivas do barramento CAN

Total de variáveis	Média	Mínimo	Máximo
1	3986,83	2881	4413
50	290,90	6	3452
100	212,38	7	3415
146	215,95	5	3426
187	206,38	7	3430

Fonte: Autor (2022).

O experimento de uma variável possui somente uma ODT, enviada a cada quatro milissegundos. Portanto, a média de tempo decorrido entre leituras consecutivas deve ser um valor em torno desse valor. A pequena diferença entre a média e o valor esperado pode ser devido a falta de precisão da ECU, causando envios consistentemente mais frequentes. As divergências apresentadas nos valores máximo e mínimo podem ser justificadas pelo uso de exclusão mútua na escrita dos dados no CircularBuffer, eventuais atrasos no processamento da ECU ou congestionamento do barramento CAN.

Para os outros experimentos, a média de tempo teórica foi calculada com o inverso da frequência de ODTs recebidas em um segundo, na qual a frequência foi calculada ao somar o resultado da multiplicação do número de ODTs no raster pela sua frequência de aquisição, para cada um dos rasters. A Equação 3 apresenta o cálculo descrito.

$$\Delta T = \frac{1}{\sum \frac{NO}{T}} \tag{3}$$

em que:

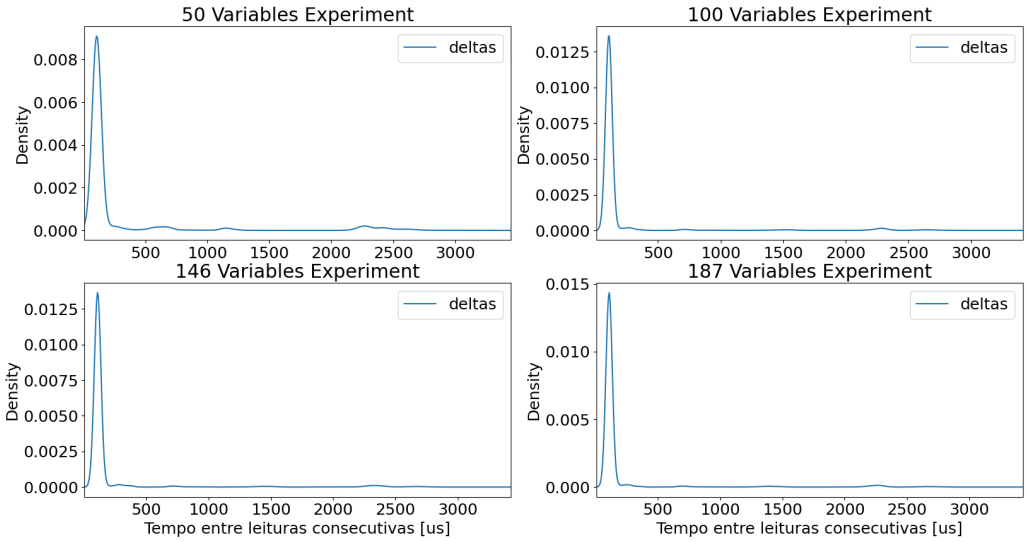
ΔT = tempo médio decorrido entre leituras consecutivas

NO = número de ODTs no raster

T = período do raster

Utilizando a Equação 3, foram calculadas as médias teóricas para os experimentos de 50, 100, 146 e 187 variáveis, e foram obtidos 307, 224, 225 e 218 microssegundos, respectivamente. A distribuição dos tempos para cada um dos experimentos pode ser visto na Figura 19. No gráfico pode ser observada a concentração em torno de 100 microssegundos, indicando que as ODTs recebidas são lidas e processadas rapidamente.

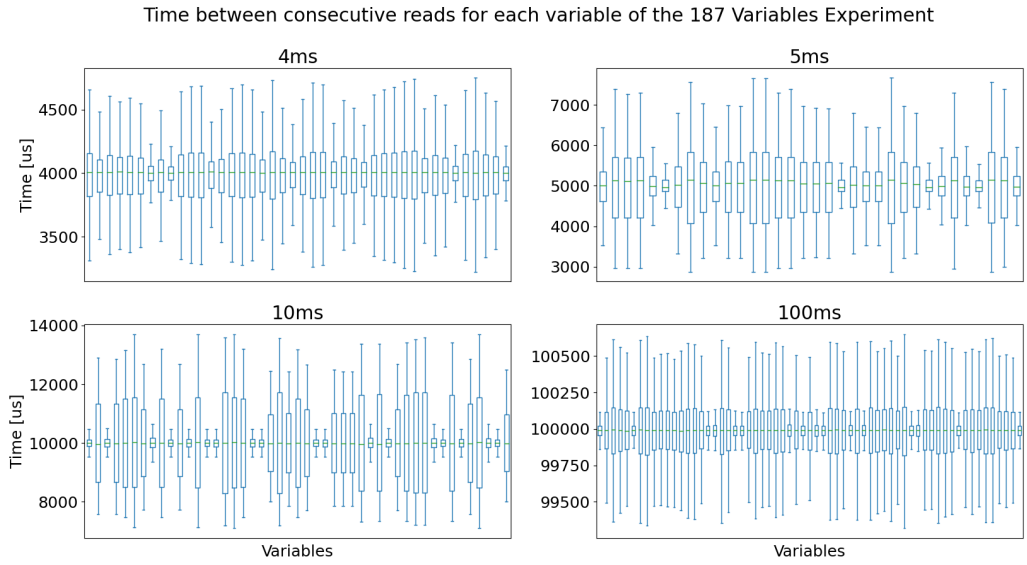
Figura 19 – Gráficos de densidade dos experimentos



Fonte: Autor (2022)

Analisando o tempo entre consecutivas leituras para cada uma das variáveis do experimento de 187 variáveis, foi gerado o gráfico boxplot apresentado na Figura 20.

Figura 20 – Boxplot do tempo entre leituras consecutivas para o experimento de 187 variáveis



Fonte: Autor (2022)

Nesse gráfico pode-se observar que os tempos estão aglomerados em torno dos valores esperados para cada raster, indicando que as leituras estão sendo realizadas no período correto para cada raster. Essa análise permite comprovar que os

requisitos RF2 e RNF12 foram cumpridos.

6.2 GRÁFICOS DE VARIÁVEIS LIDAS

Para verificar a conversão realizada e o correto armazenamento no servidor, a aplicação Grafana implementada no servidor foi usada. A variável escolhida para essa análise foi a velocidade de rotação do motor, devido ao fato de que o painel do carro oferece um ponteiro com essa informação. A Figura 21 mostra a interface com essa variável.

Figura 21 – Interface da Grafana mostrando a velocidade de rotação do motor

Fonte: Autor (2022)

Durante o experimento, o motor do carro foi ligado, foi colocado em ponto morto e o pedal do acelerador não foi pisado. Nessas condições, o ponteiro do painel referente à rotação do carro mostrava valores entre 700 e 800 rotações por minuto. Devido ao fato de que o servidor somente recebe unidades de medida do SI, os valores foram convertidos para hertz antes de enviados para o servidor e a faixa de velocidade mencionado é equivalente a aproximadamente 11,5 e 13,5 hertz.

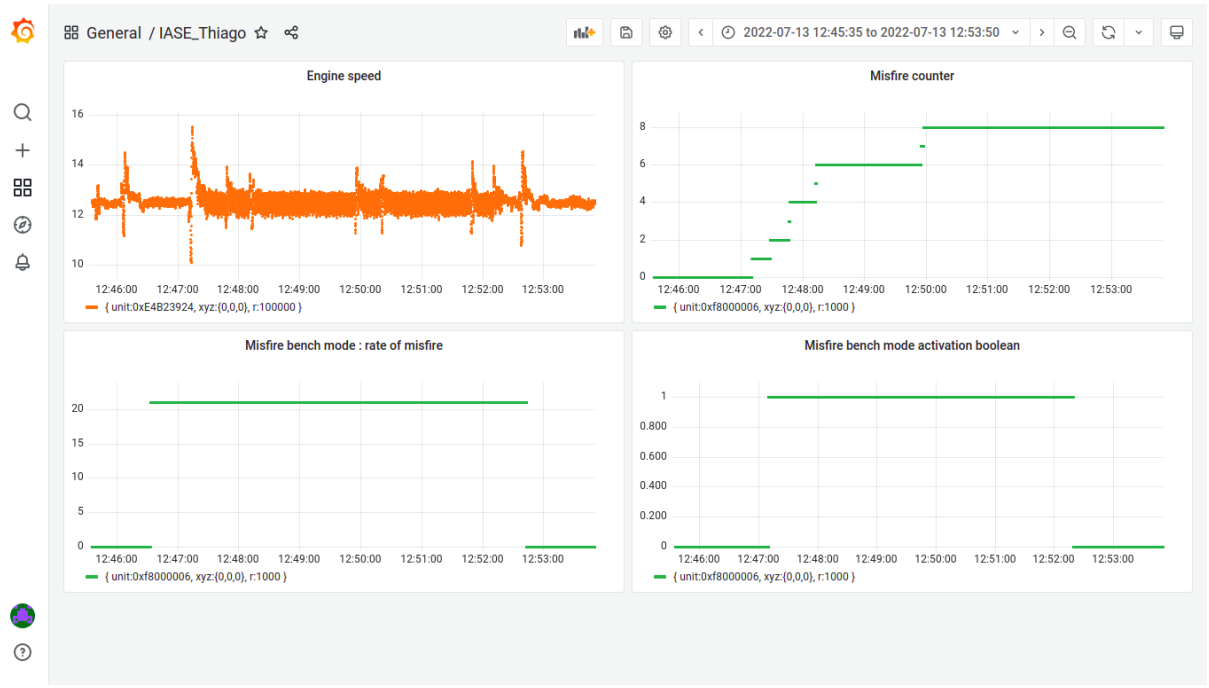
6.3 VALIDAÇÃO DA CALIBRAÇÃO

Buscando validar a calibração desenvolvida, foram escolhidas variáveis relacionadas à falha de ignição, ausência de combustão no cilindro causada pela falta de faísca que acarreta em reduzida potência do motor (SILVA, 2022). A falha

de ignição pode ser notada através do cheiro de enxofre do combustível que não foi queimado adequadamente ou sentindo o "engasgo"do motor quando a falha ocorre. Além disso, a ECU do carro também consegue detectar a ocorrência da falha, caso alguns sensores meçam características fora do normal.

A ECU utilizada possui dois parâmetros de calibração que podem ser usados para acionar essa falha. Um deles estipula a cada quantos ciclos do motor haverá uma falha de ignição programada e o outro ativa e desativa tais falhas. Ambos foram adicionados a um experimento, como parâmetros de calibração e variáveis de medição, juntamente com outra variável, que realiza a contagem de quantas falhas ocorreram desde que o veículo foi ligado. A Figura 22 mostra a interface Grafana registrada durante o experimento realizado.

Figura 22 – Interface da Grafana mostrando quatro variáveis coletadas



Fonte: Autor (2022)

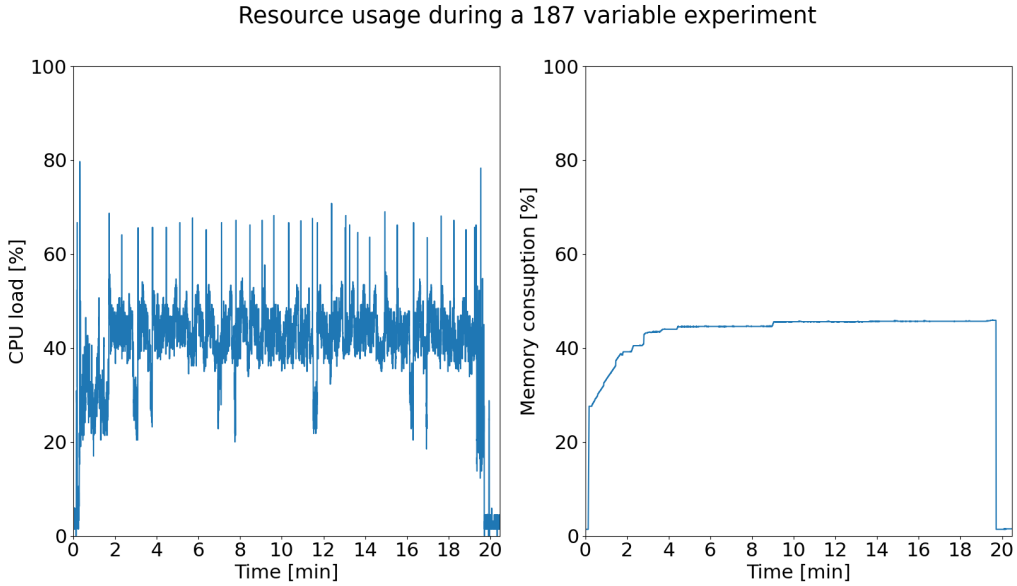
Durante o experimento, o período de ocorrência da falha foi calibrado para 21 ciclos do motor (apresentado no gráfico inferior esquerdo) e, logo em sequência, as falhas programadas foram ativadas (apresentado no gráfico inferior direito). Pode-se notar que no gráfico superior direito o valor do contador de falhas começa a aumentar quando são ativadas as falhas, e cessa após as falhas serem desligadas.

Adicionalmente, durante o experimento, foi possível sentir dentro do carro as vibrações causadas pela falha, corroborando com os gráficos indicando que as falhas estavam ocorrendo e que a calibração foi realizada corretamente.

6.4 USO DE PROCESSADOR E CONSUMO DE MEMÓRIA

Com o intuito de medir o desempenho do sistema, o experimento de 187 variáveis foi iniciado e foram registrados o consumo de memória e uso do processador durante um experimento de aproximadamente 20 minutos. A utilização dos recursos começou a ser registrado antes do início do experimento e parou somente após o encerramento do experimento. A Figura 23 mostra os valores medidos.

Figura 23 – Utilização dos recursos do sistema durante um experimento



Fonte: Autor (2022)

O uso do processador se manteve entre 40% e 50% de uso durante quase todo o experimento, indicando que os quatro núcleos de processamento são suficientes para lidar com todo o processamento necessário para executar o sistema conectado a uma ECU utilizando o protocolo CCP.

No gráfico de consumo de memória, pode-se observar que no início do software o consumo salta para quase 30%, indicando que essa é a quantidade de memória necessária para todos os objetos e estruturas criadas. Logo em sequência, o experimento é inciado e pode-se ver o consumo de memória aumentando, enquanto os dados recebidos da ECU são acumulados até que seja atingidos os limites que ativam o envio para o servidor.

A partir de certo tempo de experimento, o consumo se estabiliza em torno de 45% e permanece nesse patamar até o fim. Isso indica, portanto, que a memória do sistema é capaz de lidar com a capacidade máxima de transmissão do protocolo CCP, respeitando o requisito RNF13. Além disso, a estabilidade mostrada no gráfico é um indicativo de que não há vazamentos de memória ou acúmulo indesejado de dados no

sistema.

6.5 DISCUSSÃO DE CUSTOS

Os componentes do hardware foram adquiridos em 2021. O custo de entrada do sistema, ou seja, o preço de compra dos componentes de hardware é a soma da placa de desenvolvimento FZ3, os módulos EC25 e ESP32, a placa de circuito impresso (PCB) e outros componentes como o buzzer, reguladores de tensão e LEDs. O custo total de entrada pode ser visto na Tabela 5. Adicionalmente, foi considerado o custo anual dos dados celulares.

Tabela 5 – Custo estimado do sistema desenvolvido em dólares

Componente	Custo
FZ3	300
EC25	55
ESP32	5
PCB	30
Outros	40
Total	430
Dados celulares (ano)	240

Fonte: Autor (2022).

O sistema comercial disponível para comparação era da fabricante ETAS GmbH e foi fornecido pela Renault. É composto de um conector ES581 responsável por conectar a ECU em um computador, sendo esses dois componentes correspondentes ao custo de entrada. Além disso, a licença do software INCA também deve ser levado em consideração. Esses custos podem ser vistos na Tabela 6.

Tabela 6 – Custo estimado de uma solução comercial em dólares

Componente	Custo
ES581	1000
Computador	2900
Total	3900
Licença INCA (ano)	1600

Fonte: Autor (2022).

Logo, somente o hardware desenvolvido custou menos da metade do preço do conector ES581, com a vantagem de não ser necessário um computador, pois o processamento é realizado pelo hardware e enviado para um servidor. O sistema desenvolvido também representa um custo de manutenção anual menor, já que a despesa com dados celulares de uma operadora são estimados em torno de 240 dólares ao ano, em contraste aos 1600 dólares referentes à anuidade do software INCA.

6.6 COMPARAÇÃO DE FUNCIONALIDADES

Soluções comerciais foram utilizadas como base para se planejar as características desejadas para o sistema desenvolvido, portanto ele possui funcionalidades em comum e diferentes com tais soluções. O Quadro 2 mostra uma comparação entre as funcionalidades do sistema desenvolvido e de soluções comerciais da fabricante ETAS.

Quadro 2 – Comparação das funcionalidades do sistema com uma solução comercial

Funcionalidade	Solução ETAS	Sistema desenvolvido
CCP	X	X
XCP	X	X
Coleta de dados síncrona	X	X
Coleta de dados assíncrona	X	
Calibração	X	X
Armazenamento em disco	X	X
Localização por GPS		X
Envio de dados em tempo real		X
Visualização de dados online		X
Visualização de dados offline	X	
Suporte a múltiplos protocolos de transporte	X	
Acesso à porta JTAG	X	
Programação da ECU	X	

Fonte: Autor (2022).

O sistema desenvolvido não possui algumas funcionalidades presentes na solução comercial, como suporte a programação da ECU e múltiplos protocolos de transporte. Porém, dispõe de outras funcionalidades desejadas em testes realizados fora de laboratórios, devido à facilidade de se acompanhar os testes durante sua execução, uma vez que são enviados para um servidor.

7 CONCLUSÕES

Reduzir custos sem afetar a qualidade do produto é importante para se manter competitivo no mercado e aumentar lucros. No processo de desenvolvimento de um veículo, extensivos testes precisam ser realizados para garantir sua segurança e eficiência. Tais testes normalmente requerem caros sistemas de instrumentação e coleta de dados, que não possuem todas as funcionalidades necessárias e possuem outras desnecessárias.

Portanto, o trabalho desenvolvido buscou criar um sistema embarcado que pudesse ser usado para coletar dados de veículos durante testes realizados em ruas e estradas, além de permitir o envio dos dados coletados durante a execução do experimento a um servidor, onde pode ser processado.

Com base nos experimentos realizados, foi possível medir que o sistema é capaz de coletar e enviar dados sem atraso e que no experimento de maior carga realizado, o uso do processador e consumo de memória se mantiveram em torno de 45%, indicando que o hardware não está subdimensionado e que não há um problema com o software.

Foi verificado que as variáveis coletadas possuem valores corretos e os parâmetros podem ser calibrados. Além disso, foi observado o correto funcionamento da visualização dos dados no servidor e do estado do sistema no aplicativo de celular. Portanto, todos os requisitos funcionais e não funcionais do sistema foram cumpridos.

O sistema foi bem sucedido ao apresentar uma alternativa que representa menor custo de aquisição e manutenção anual se comparado a opções comerciais. Também foi sucedido por implementar funcionalidades que não estão disponíveis em outros produtos, como envio de dados em tempo real e localização por GPS.

7.1 TRABALHOS FUTUROS

Buscando melhorar o sistema desenvolvido, alguns pontos podem ser propostos. Primeiramente, o hardware pode ser melhor dimensionado a fim de reduzir o custo do total. A placa de desenvolvimento pode ser substituída por um hardware customizado para as necessidades do sistema, uma vez que o FPGA é mais caro do que um circuito integrado de aplicação específica e não há necessidade de ter uma GPU integrada. Adicionalmente, os conectores suportados pelo hardware podem ser reduzidos a somente os utilizados pelo protocolo XCP ou pela indústria automotiva.

Em segundo lugar, a máquina de estados ECUReader pode ser generalizada para trabalhar com outros protocolos de comunicação usados no mercado automotivo, como Ethernet, Flexray, LIN e outros, suportados por produtos oferecidos por empresas

já estabelecidas no mercado. Similarmente, a máquina de estados DataController poderia ser generalizada para enviar qualquer tipo de mensagem HTTP, visando facilitar a implementação da comunicação com qualquer servidor.

Em terceiro lugar, poderia ser adicionado o suporte completo a todas as funcionalidades do protocolo XCP, como a programação da memória da ECU, início automático de experimentos, configuração em bloco, entre outros. Isso permitiria que o sistema seja utilizado em diferentes estágios do desenvolvimento de um veículo, bem como em outras indústrias, caso desejado.

Por fim, poderiam ser adicionadas novas funcionalidades ao servidor, de forma a melhor processar os dados recebidos. Modelos treinados por inteligência artificial poderiam ser usados para detectar anomalias nos testes, indicar possíveis falhas, apontar possíveis melhorias no veículo ou em sua calibração, entre outros. Também poderiam ser definidas condições nas quais alertas são disparados quando algum problema for detectado, aumentando a segurança do motorista e reduzindo o desperdício de tempo.

REFERÊNCIAS

ACCURATE TECHNOLOGIES INC. **ECU Interfaces**. Novi, 2022. Disponível em: https://www.accuratetechnologies.com/ECUInterfaces. Acesso em: 25 jun. 2022.

ACCURATE TECHNOLOGIES INC. **VISION Software**. Novi, 2022. Disponível em: https://www.accuratetechnologies.com/Products/VISIONSoftware. Acesso em: 25 jun. 2022.

ALAM, M. S. U. **Securing vehicle Electronic Control Unit (ECU) communications and stored data**. Tese (Doutorado) — Queen's University, Canada, 2018.

ANATEL. **Emissões**. Brasília, 2022. Disponível em: https://sistemas.anatel.gov.br/se/public/view/b/licenciamento.php. Acesso em: 02 out. 2022.

ARBEITSKREIS ZUR STANDARDISIERUNG VON APPLIKATIONSSYSTEMEN STANDARDIZATION OF APPLICATION/CALIBRATION SYSTEMS TASK FORCE. **CAN Calibration Protocol Version 2.1**. Hoehenkirchen, 1999.

ASSOCIATION FOR STANDARDISATION OF AUTOMATION AND MEASURING SYSTEMS. **ASAM MCD-2 MC**: Data model for ecu measurement and calibration. Hoehenkirchen, 2010.

ASSOCIATION FOR STANDARDISATION OF AUTOMATION AND MEASURING SYSTEMS. **ASAM MCD-2 MC**. Hoehenkirchen, 2022. Disponível em: https://www.asam.net/standards/detail/mcd-2-mc/. Acesso em: 28 jun. 2022.

ASSOCIATION FOR STANDARDIZATION OF AUTOMATION AND MEASURING SYSTEMS. **The Universal Measurement and Calibration Protocol Family**: Overview. Hoehenkirchen, 2003.

ASSOCIATION FOR STANDARDIZATION OF AUTOMATION AND MEASURING SYSTEMS. **The Universal Measurement and Calibration Protocol Family**: Protocol layer specification. Hoehenkirchen, 2003.

ASSOCIATION FOR STANDARDIZATION OF AUTOMATION AND MEASURING SYSTEMS. **The Universal Measurement and Calibration Protocol Family**: Xcp on can - transport layer specification. Hoehenkirchen, 2003.

BAGHLI, L.; BENMANSOUR, K.; DJEMAI, M. Development of a data acquisition and tracking system for vehicles. In: IEEE. **3rd International Symposium on Environmental Friendly Energies and Applications (EFEA)**. [S.l.], 2014. p. 1–6.

CHEN, H.; TIAN, J. Research on the controller area network. In: IEEE. **Proceedings** of the 2009 INTERNATIONAL CONFERENCE ON NETWORKING AND DIGITAL SOCIETY. Guiyang, 2009. v. 2, p. 251–254. Disponível em: https://ieeexplore.ieee.org/abstract/document/5116731. Acesso em: 09 jun. 2022.

ESPRESSIF SYSTEMS. **ESP32-WROOM-32D & ESP32-WROOM-32U**: Datasheet. Xangai, 2022. Disponível em: https://www.espressif.com/sites/default/files/

documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf. Acesso em: 02 out. 2022.

ETAS GmbH. **About ETAS**. Stuttgart, 2022. Disponível em: https://www.etas.com/en/company/about-etas.php. Acesso em: 25 jun. 2022.

ETAS GmbH. **ES58x**: Basic interface modules. Stuttgart, 2022. Disponível em: https://www.etas.com/en/products/es58x.php. Acesso em: 25 jun. 2022.

ETAS GmbH. **ETK**: Ecu interface. Stuttgart, 2022. Disponível em: https://www.etas.com/en/products/etk.php. Acesso em: 25 jun. 2022.

ETAS GmbH. **INCA Base Product**. Stuttgart, 2022. Disponível em: https://www.etas.com/en/products/inca.php. Acesso em: 25 jun. 2022.

FAIZAN, M.; PILLAI, A. S. Dynamic task allocation and scheduling for multicore electronics control unit (ecu). In: IEEE. **Proceedings** of the 2019 3RD INTERNATIONAL CONFERENCE ON ELECTRONICS, COMMUNICATION AND AEROSPACE TECHNOLOGY (ICECA). Coimbatore, 2019. p. 821–826. Disponível em: https://ieeexplore.ieee.org/abstract/document/8822086. Acesso em: 11 mai. 2022.

GAILLY, J. loup; ADLER, M. **zlib**: A massively spiffy yet delicately unobtrusive compression library. 2022. Disponível em: https://www.zlib.net/. Acesso em: 24 out. 2022.

GRACIOLI, G.; FRöHLICH, A. A. **Sistema Inteligente de aquisição e análise de dados para controladores automotivos**. Projeto de pesquisa (Laboratório de Integração Software/Hardware) — Centro Tecnológico de Joinville, Universidade Federal de Santa Catarina, Joinville, 2020.

HERPEL, T. et al. Assessing the can communication startup behavior of automotive ecus by prototype measurements. In: IEEE. **Proceedings** of the 2009 IEEE INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE. Singapore, 2009. p. 928–932. Disponível em: https://ieeexplore.ieee.org/abstract/document/5168584. Acesso em: 12 jun. 2022.

JI-HUI, Z.; HUI, X.; YING, Y. Research and development of electric vehicle data collection and calibration platform based on gprs and internet. In: IEEE. **Proceedings** of the 2008 IEEE VEHICLE POWER AND PROPULSION CONFERENCE. Harbin, 2008. p. 1–5. Disponível em: https://ieeexplore.ieee.org/abstract/document/4677595. Acesso em: 12 jun. 2022.

LABORATóRIO DE INTEGRAçãO SOFTWARE/HARDWARE. **IoT Platform**. Florianópolis, 2022. Disponível em: https://epos.lisha.ufsc.br/IoT+Platform#SmartData. Acesso em: 02 out. 2022.

LISHA. **Android Application**. Joinville, 2022. Disponível em: https://lisha.ufsc.br/Renault+IASE+-+Documentation+-+Android+Application. Acesso em: 02 out. 2022.

LISHA. **IoT Platform**. Florianópolis, 2022. Disponível em: https://epos.lisha.ufsc.br/IoT+Platform. Acesso em: 02 out. 2022.

LISHA. **Zynq UltraScale+ MPSoC Configuration**. Joinville, 2022. Disponível em: https://lisha.ufsc.br/Renault+IASE+-+Documentation+-+Zynq+UltraScale+MPSoC+Configuration. Acesso em: 02 out. 2022.

MYIR TECH LIMITED. **FZ3 Card – Deep Learning Accelerator Card**. Shenzhen, 2022. Disponível em: http://www.myirtech.com/download/ZU3EG/FZ3Card.pdf. Acesso em: 02 out. 2022.

NASERIAN, M.; KRUEGER, K. A test-bed for verification of vehicle safety communications applications. In: IEEE. **Proceedings** of the 2009 IEEE INTERNATIONAL CONFERENCE ON ELECTRO/INFORMATION TECHNOLOGY. Windsor, 2009. p. 327–331. Disponível em: https://ieeexplore.ieee.org/abstract/document/5189637. Acesso em: 11 mai. 2022.

PAN, L. et al. Cyber security attacks to modern vehicular systems. **Journal of Information Security and Applications**, v. 36, p. 90–100, 2017.

QUECTEL WIRELESS SOLUTIONS CO., LTD. **Quectel Antenna Design Guide**. Xangai, 2022. Disponível em: https://www.quectel.com/download/quectel_antenna_design_note_v3-2. Acesso em: 02 out. 2022.

QUECTEL WIRELESS SOLUTIONS CO., LTD. **Quectel EC25 Series**: Iot/m2m-optimized lte cat 4 module. Xangai, 2022. Disponível em: https://www.quectel.com/wp-content/uploads/2021/09/Quectel_EC25_Series_LTE_Standard_Specification_V2.4.pdf. Acesso em: 02 out. 2022.

ROSE, D. **Converting between Earth-Centered, Earth Fixed and Geodetic Coordinates**. 2014. Disponível em: https://danceswithcode.net/engineeringnotes/geodetic_to_ecef/geodetic_to_ecef.html. Acesso em: 24 out. 2022.

SAUTER, F.; TRAUB, K. **Curl for People**. 2022. Disponível em: https://docs.libcpr.org/. Acesso em: 24 out. 2022.

SHARMA, P.; MÖLLER, D. P. Protecting ecus and vehicles internal networks. In: IEEE. **Proceedings** of the 2018 IEEE International Conference on Electro/Information Technology (EIT). Rochester, 2018. p. 0465–0470. Disponível em: https://ieeexplore.ieee.org/abstract/document/8500295. Acesso em: 12 jun. 2022.

SILVA, M. E. R. da. **Seleção de atributos em aprendizado de máquina para identificação de falhas em motores de combustão interna**. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) — Centro Tecnológico de Joinville, Universidade Federal de Santa Catarina, Joinville, 2022.

SILVA, T. M.; SPOHN, M. A.; PADILHA, A. S. Uma análise comparativa entre os protocolos canopen, devicenet e smart distributed system. **Revista Brasileira de Computação Aplicada**, v. 9, n. 1, p. 2–14, 2017.

VECTOR INFORMATIK GMBH. **VX1000 Manual**. Stuttgart, 2021. Disponível em: https://cdn.vector.com/cms/content/products/vx1000/Docs/VX1000_Manual.pdf.

VECTOR INFORMATIK GmbH. **ECU Calibration with CANape**. Stuttgart, 2022. Disponível em: https://www.vector.com/br/pt/produtos/products-a-z/software/canape/. Acesso em: 25 jun. 2022.

VECTOR INFORMATIK GmbH. **Network Interfaces**. Stuttgart, 2022. Disponível em: https://www.vector.com/br/pt/produtos/products-a-z/hardware/network-interfaces/. Acesso em: 25 jun. 2022.

WANG, J. et al. Development of a new calibration and monitoring system for in-vehicle electronic control units based on controller area network calibration protocol. **Journal of Automobile Engineering**, Sage Publications Sage UK: London, England, v. 219, n. 12, p. 1381–1389, 2005. Disponível em: https://journals.sagepub.com/doi/abs/10.1243/095440705X35044. Acesso em: 12 jun. 2022.

XIE, Y. et al. Stm32-based vehicle data acquisition system for internet-of-vehicles. In: IEEE. **2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)**. [S.l.], 2017. p. 895–898.

XILINX. **PetaLinux Tools Documentation**: Reference guide. San José, 2020. Disponível em: https://docs.xilinx.com/v/u/2020.1-English/ug1144-petalinux-tools-reference-guide. Acesso em: 04 out. 2022.

YU, J.; WILAMOWSKI, B. M. Recent advances in in-vehicle embedded systems. In: IEEE. **Proceedings** of the IECON 2011-37TH ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY. Melbourne, 2011. p. 4623–4625. Disponível em: https://ieeexplore.ieee.org/abstract/document/6120072. Acesso em: 12 jun. 2022.