



UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CAMPUS CARAÚBAS
CURSO BACHARELADO EM ENGENHARIA ELÉTRICA

LUIZ SOARES DA SILVA NETO

**APLICAÇÃO DE INTERNET DAS COISAS NO DESENVOLVIMENTO
DE UMA BANCADA DE TESTE PARA AVALIAÇÃO DA
ACELERAÇÃO E VELOCIDADE EM VEÍCULOS OFF-ROAD DO TIPO
BAJA UTILIZANDO A PLATAFORMA ARDUINO**

CARAÚBAS

2025

LUIZ SOARES DA SILVA NETO

**APLICAÇÃO DE INTERNET DAS COISAS NO
DESENVOLVIMENTO DE UMA BANCADA DE TESTE PARA
AVALIAÇÃO DA ACELERAÇÃO E VELOCIDADE EM
VEÍCULOS OFF-ROAD DO TIPO BAJA UTILIZANDO A
PLATAFORMA ARDUINO**

Monografia apresentada a Universidade
Federal Rural do Semi-Árido – UFERSA,
Campus Caraúbas para obtenção do título de
Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Walber Medeiros Lima

CARAÚBAS

2025

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei nº 9.279/1996 e Direitos Autorais: Lei nº 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

S586a Silva Neto, Luiz Soares da.
Aplicação de Internet das Coisas no desenvolvimento de uma bancada de teste para avaliação da aceleração e velocidade em veículos off-road do tipo Baja utilizando a plataforma Arduino / Luiz Soares da Silva Neto. - 2025.
65 f. : il.

Orientador: Walber Medeiros Lima.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de --Selecione um Curso ou Programa--, 2025.

1. Internet das Coisas. 2. Telemetria. 3. BAJA SAE. 4. Arduino NANO 33 IoT. 5. Sensores Fotoelétricos. I. Lima, Walber Medeiros, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo(a) autor(a).
Biblioteca Campus Caraúbas
Bibliotecária: Sarah Freire Bezerra
CRB: 15/1052

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC's) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

LUIZ SOARES DA SILVA NETO

**APLICAÇÃO DE INTERNET DAS COISAS NO
DESENVOLVIMENTO DE UMA BANCADA DE TESTE PARA
AVALIAÇÃO DA ACELERAÇÃO E VELOCIDADE EM
VEÍCULOS OFF-ROAD DO TIPO BAJA UTILIZANDO A
PLATAFORMA ARDUINO**

Monografia apresentada a Universidade
Federal Rural do Semi-Árido – UFERSA,
Campus Caraúbas para obtenção do título de
Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Walber Medeiros Lima

Defendida em 26 / 03/ 2025

Banca Examinadora:

Documento assinado digitalmente
 **WALBER MEDEIROS LIMA**
Data: 27/03/2025 13:07:25-0300
Verifique em <https://validar.it.gov.br>

Prof. Dr. Walber Medeiros Lima
Presidente

Documento assinado digitalmente
 **HUGO MICHEL CAMARA DE AZEVEDO MAIA**
Data: 27/03/2025 18:47:25-0300
Verifique em <https://validar.it.gov.br>

Prof. Dr. Hugo Michel C. de A. Maia
Membro Examinador

Documento assinado digitalmente
 **CHARLES AUGUSTO SANTOS DO CARMO**
Data: 27/03/2025 19:09:54-0300
Verifique em <https://validar.it.gov.br>

Esp. Charles Augusto S. do Carmo
Membro Examinador

“Confie no Senhor de todo o seu coração e não se apoie em seu próprio entendimento; em todos os seus caminhos, submeta-se a ele, e ele endireitará as suas veredas. ”

Provérbios 3:5-6.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por me conceder força, resiliência, paz e perseverança, elementos essenciais ao longo desta jornada transformadora. Cada tropeço, cada adversidade, cada colega de caminhada, cada apoio recebido e cada aprendizado contribuíram para minha convicção e para este momento presente.

Gostaria de expressar minha eterna gratidão ao meu orientador, professor Dr. Walber Medeiros Lima, por sua paciência, sabedoria e solicitude. Graças ao seu apoio fundamental, mais um desafio foi superado. Seus ensinamentos foram essenciais para minha formação profissional.

À minha família, especialmente ao meu pai, minha mãe, meu filho e minha esposa, agradeço pelo suporte e incentivo incondicionais, fundamentais em toda a minha jornada acadêmica e profissional.

Dedico este trabalho, com profundo amor e saudade, aos meus falecidos avós e padrinhos, Gercina Oliveira de Freitas (in memoriam) e Aluísio Alves (in memoriam), que me criaram desde a infância até minha entrada na universidade. Infelizmente, não poderão estar presentes em minha formatura, mas permanecerão vivos em minhas ações, palavras, pensamentos e em meu coração.

Agradeço também à minha banca examinadora por sua presença, disponibilidade e valiosas contribuições, que enriqueceram significativamente a qualidade deste estudo. Suas críticas construtivas e sugestões foram fundamentais para o aprimoramento do conteúdo e da abordagem deste trabalho.

Por fim, quero agradecer a todos os professores que me auxiliaram até aqui, aos colegas, técnicos, terceirizados, funcionários da universidade e demais pessoas do meu convívio diário nesta batalha. Todos participaram, direta ou indiretamente, deste capítulo da minha vida e têm minha eterna gratidão. As bifurcações na estrada dividiram nossos caminhos, mas suas influências foram determinantes no desfecho dessa aventura emocionante. Estarei sempre à disposição de cada um, caso um dia precisem.

RESUMO

Este trabalho tem como objetivo o desenvolvimento de um protótipo de bancada de testes para a avaliação de veículos *off-road* do tipo BAJA SAE, focando nas provas de aceleração e velocidade. A solução proposta utiliza a plataforma Arduino Nano 33 IoT e sensores fotoelétricos para capturar dados dinâmicos em tempo real. A metodologia consiste na implementação de um sistema de telemetria que permite a avaliação quantitativa de parâmetros como velocidade e aceleração, com sincronização temporal via protocolo *Network Time Protocol*. A comunicação sem fio é realizada utilizando uma rede WiFi utilizando o protocolo *User Datagram Protocol* para uma transmissão mais rápida de dados. Para facilitar a análise remota, foi desenvolvida uma interface de programação de aplicações baseada em transferência representacional de estado, acompanhada de uma interface web responsiva, garantindo acessibilidade e suporte durante os testes. Os resultados demonstraram que a combinação de componentes de baixo custo com técnicas de Internet das Coisas oferece uma solução tecnicamente viável, aplicável a cenários reais da engenharia automotiva. Além disso, essa abordagem proporciona maior agilidade e precisão nas avaliações realizadas.

Palavras-chave: Telemetria, Arduino Nano 33 IoT, BAJA SAE, Sensores Fotoelétricos, Internet das Coisas (IoT).

ABSTRACT

This work aims to develop a test bench prototype for evaluating off-road vehicles of the BAJA SAE type, focusing on acceleration and speed tests. The proposed solution utilizes the Arduino Nano 33 IoT platform and photoelectric sensors to capture dynamic data in real-time. The methodology consists of implementing a telemetry system that enables the quantitative assessment of parameters such as speed and acceleration, with time synchronization via the Network Time Protocol. Wireless communication is carried out using a WiFi network with the User Datagram Protocol to ensure faster data transmission. To facilitate remote analysis, an application programming interface based on Representational State Transfer (API REST) was developed, accompanied by a responsive web interface, ensuring accessibility and support during testing. The results demonstrated that combining low-cost components with Internet of Things techniques offers a technically viable solution applicable to real-world scenarios in automotive engineering. Additionally, this approach provides greater agility and accuracy in the evaluations performed.

Keywords: Telemetry, Arduino Nano 33 IoT, BAJA SAE, Photoelectric Sensors, Internet of Things (IoT).

LISTA DE ILUSTRAÇÕES

Figura 1: Representação dos testes de aceleração e velocidade.	16
Figura 2: Arduino NANO 33 IoT.	17
Figura 3: Descrição dos pinos do Arduino Nano 33 IoT.	17
Figura 4: Fluxograma da estrutura do <i>firmware</i> do Arduino.	19
Figura 5: Camadas do modelo TCP/IP	23
Figura 6: Analogia do funcionamento comunicação via UDP.	24
Figura 7: Sensor de proximidade fotoelétrico difuso PNP.	31
Figura 8: Posicionamento dos sensores ao longo do percurso.	32
Figura 9: Esquema de ligação do Arduino servidor e clientes.	34
Figura 10: Rotina de <i>setup()</i> do <i>firmware</i> da unidade central.	35
Figura 11: Rotina de <i>loop()</i> do <i>firmware</i> da unidade central.	36
Figura 12: Rotina de <i>setup()</i> do <i>firmware</i> dos nós de aquisição de dados	38
Figura 13: Rotina de <i>loop()</i> do <i>firmware</i> dos nós de aquisição de dados.	39
Figura 14: Resposta da Unidade Central via Serial.	40
Figura 15: Resposta da Unidade Central após a sinalização dos clientes.	40
Figura 16: Resposta HTML a partir de um <i>smartphone</i>	41
Figura 17: Resposta JSON utilizando o <i>Firefox</i>	41

LISTA DE TABELAS

Tabela 1 – Comparação TCP x UDP.....	25
Tabela 2 – Relação de componentes.....	33

SUMÁRIO

LISTA DE TABELAS	9
1. INTRODUÇÃO	12
1.1. OBJETIVO GERAL	13
1.2. OBJETIVO ESPECÍFICO	13
1.3. MOTIVAÇÃO	13
1.4. ESTRUTURA DO TRABALHO.....	14
2. REVISÃO BIBLIOGRÁFICA	15
2.1 COMPETIÇÕES BAJA SAE E PROVA DE VELOCIDADE E ACELERAÇÃO.....	15
2.1.1 BAJA SAE BRASIL	15
2.1.2 TESTE DE VELOCIDADE E ACELERAÇÃO	15
2.2 MICROCONTROLADORES E A PLATAFORMA ARDUINO.....	16
2.2.1 BIBLIOTECAS NA PROGRAMAÇÃO	17
2.2.2 FIRMWARE	18
2.3 INTERNET DAS COISAS	19
2.4 REDES DE COMUNICAÇÃO SEM FIO.....	20
2.5 PROTOCOLOS DE COMUNICAÇÃO.....	22
2.5.1 MODELO TCP/IP.....	22
2.5.2 PROTOCOLO UDP	23
2.5.3 COMPARAÇÃO ENTRE UDP E TCP	24
2.6 SINCRONIZAÇÃO TEMPORAL E RTC	25
2.6.1 NETWORK TIME PROTOCOL	25
2.6.2 REAL TIME CLOCK.....	26
2.7 SERVIDOR HTTP E API REST PARA MONITORAMENTO DOS DADOS	27
2.7.1 SERVIDOR HTTP	27

2.7.2	API REST.....	29
2.8	SENSOR DE PROXIMIDADE FOTOELÉTRICO.....	30
3.	METODOLOGIA.....	32
3.1	COMPONENTES E CIRCUITOS	33
3.1.1	DESENVOLVIMENTO DOS FIRMWARES	34
3.1.1.1	UNIDADE CENTRAL (ARDUINO SERVIDOR)	34
3.1.1.2	ARDUINOS CLIENTES.....	38
4.	RESULTADOS	40
5.	CONCLUSÃO E PERSPECTIVAS FUTURAS.....	42
5.1	CONCLUSÃO.....	42
5.2	PERSPECTIVAS FUTURAS.....	42
6.	REFERÊNCIAS BIBLIOGRÁFICAS	44
APÊNDICE A		47
APÊNDICE B.....		63
APÊNDICE C		65

1. INTRODUÇÃO

A crescente adoção de tecnologias digitais pela sociedade tem impulsionado o desenvolvimento de soluções inovadoras para uma variedade de problemas cotidianos, à medida que o uso de computadores e sistemas de comunicação se torna cada vez mais eficiente e acessível (PEREIRA, 2024). Esse avanço, em parte impulsionado pela rápida evolução da informática e o aprimoramento de técnicas em engenharia de *software* e da Internet das Coisas, possibilitou a criação de ferramentas e plataformas que integram tecnologias emergentes de forma cada vez mais prática.

Dentro desse contexto, a evolução da eletrônica, especialmente no que diz respeito a microcontroladores e sensores, tem transformado significativamente a coleta, o processamento e a utilização de informações em diversas aplicações práticas (GOUVEIA, 2013). No setor automotivo, esses avanços desempenham um papel fundamental no desenvolvimento de soluções para o monitoramento e otimização do desempenho dos veículos, especialmente em ambientes de alta exigência, como os encontrados em competições de veículos *off-road*.

Nesse cenário, a competição BAJA SAE se destaca como um desafio internacional que incentiva estudantes de engenharia a projetar, construir e competir com veículos *off-road* em condições extremas, exigindo rigorosos critérios de desempenho, confiabilidade e inovação tecnológica. O time CARAUBAJA, da UFERSA - Campus Caraúbas, participa ativamente dessa competição, buscando incessantemente inovações que aprimorem o desempenho e a eficiência de seus protótipos.

Este trabalho propõe o desenvolvimento de uma bancada de teste para a avaliação de veículos *off-road* do tipo BAJA SAE. A solução consiste na implementação de um sistema de telemetria, baseado na Tecnologia da Internet das Coisas (IoT), que possibilita a avaliação quantitativa de parâmetros dinâmicos, como velocidade e aceleração. O sistema foi especificamente projetado para medir esses parâmetros com precisão, levantando o perfil cinemático do veículo durante os testes. A solução implementada combina sensoriamento distribuído, comunicação sem fio via protocolo UDP e processamento de dados em tempo real, permitindo a aquisição sincronizada de eventos em múltiplos pontos ao longo do percurso.

1.1. OBJETIVO GERAL

O objetivo deste trabalho é desenvolver uma bancada de teste para avaliação de veículos *off-road* BAJA SAE, implementando um sistema de telemetria baseado em IoT para mensuração quantitativa de parâmetros dinâmicos, como velocidade e aceleração, utilizando a plataforma Arduino Nano 33 IoT e sensores fotoelétricos.

1.2. OBJETIVO ESPECÍFICO

- Projetar e desenvolver um protótipo bancada de teste para avaliação de veículos *off-road* do tipo BAJA SAE, focando nas provas de aceleração e velocidade;
- Implementar um sistema de telemetria baseado na Tecnologia da Internet das Coisas (IoT) para a aquisição e análise de dados dinâmicos, como velocidade e aceleração;
- Integrar sensores fotoelétricos e plataforma Arduino Nano 33 IoT para garantir medições precisas e em tempo real;
- Desenvolver um ambiente de comunicação sem fio utilizando o protocolo UDP, assegurando a transmissão eficiente dos dados coletados;
- Criar uma API REST e uma *interface web* para visualização e monitoramento remoto dos parâmetros do veículo;
- Validar o desempenho do sistema por meio de testes experimentais, analisando a precisão dos registros, a latência da comunicação e a viabilidade da aplicação em competições reais.

1.3. MOTIVAÇÃO

A motivação para este trabalho surge da necessidade de desenvolver soluções de monitoramento de baixo custo e alta precisão, utilizando tecnologias modernas como IoT. Tais soluções são essenciais para equipes de competições estudantis, como a BAJA SAE, que, no caso da equipe CARAUBAJA/UFERSA Campus Caraúbas, requerem ferramentas capazes de otimizar o desempenho com base em dados reais.

1.4. ESTRUTURA DO TRABALHO

O trabalho está organizado em cinco capítulos, conforme descrito a seguir:

- Capítulo 1: Introdução ao tema, abordando a aplicação de sistemas embarcados e IoT na engenharia automotiva. Este capítulo apresenta os objetivos, motivações e justificativas técnicas e econômicas do projeto, além de fornecer uma visão geral dos recursos utilizados.
- Capítulo 2: Fundamentação teórica, abordando conceitos de cinemática veicular, princípios de funcionamento de sensores ópticos, protocolos de comunicação sem fio (Wi-Fi/UDP) e técnicas de sincronização temporal via NTP/RTC. Este capítulo também inclui uma revisão de trabalhos correlatos e as especificações técnicas dos componentes utilizados.
- Capítulo 3: Descrição da parte prática do projeto, detalhando a montagem do circuito de aquisição de dados, os diagramas de conexões, o desenvolvimento do *firmware* para os Arduinos (servidor e clientes), a configuração da comunicação sem fio entre os dispositivos, e o desenvolvimento da API REST e da *interface web* para monitoramento em tempo real.
- Capítulo 4: Análise experimental, apresentando os resultados de testes realizados em condições controladas e em ambiente real. São discutidos aspectos como a precisão dos registros, a latência da comunicação e a viabilidade de implementar o sistema em condições reais de competição.
- Capítulo 5: No capítulo final, são apresentadas as conclusões e as perspectivas futuras, destacando as contribuições do projeto para a telemetria veicular e propondo direções para melhorias e expansão do sistema, com o objetivo de aumentar a robustez e a aplicabilidade da solução desenvolvida.

2. REVISÃO BIBLIOGRÁFICA

2.1 COMPETIÇÕES BAJA SAE E PROVA DE VELOCIDADE E ACELERAÇÃO

2.1.1 BAJA SAE BRASIL

O Programa BAJA SAE BRASIL é uma iniciativa destinada a estudantes de Engenharia, oferecendo-lhes a oportunidade de aplicar conhecimentos teóricos na prática por meio do desenvolvimento de protótipos de veículos *off-road*. Criado nos Estados Unidos em 1976, o projeto foi implementado no Brasil em 1991 pela SAE BRASIL, com a primeira competição nacional realizada em 1995. O evento é estruturado em fases que avaliam tanto aspectos teóricos, como concepção e análise de custos, quanto aspectos práticos, incluindo testes dinâmicos de aceleração, velocidade, tração e resistência (SAE BRASIL, 2025).

As competições, realizadas em níveis regional, nacional e internacional, estimulam habilidades técnicas e interpessoais, como gestão de projetos, inovação tecnológica e trabalho em equipe. Além disso, proporcionam aos participantes uma preparação diferenciada para o mercado de trabalho, além de oportunidades de *networking*. Relatos de ex-participantes destacam a relevância do programa na formação profissional, evidenciando seu impacto positivo na aquisição de competências práticas e na capacidade de enfrentar desafios industriais.

2.1.2 TESTE DE VELOCIDADE E ACELERAÇÃO

Os testes de velocidade e aceleração, ilustrado na Figura 1, têm como objetivo avaliar a performance dinâmica do veículo projetado pelas equipes. Esses testes constituem duas etapas presentes na competição nacional BAJA SAE e são realizados em superfície pavimentada. O teste de aceleração avalia o tempo necessário para o veículo percorrer os primeiros 30 metros, partindo do repouso. Já o teste de velocidade verifica a velocidade final atingida pelo veículo em um trajeto de 100 metros (RODRIGUES, 2011). A Figura 1 ilustra os testes de velocidade e aceleração.

Figura 1: Representação dos testes de aceleração e velocidade.



Fonte: Adaptado de (RODRIGUES, 2011).

2.2 MICROCONTROLADORES E A PLATAFORMA ARDUINO

Os microcontroladores são circuitos integrados (CI) programáveis que reúnem, em um único dispositivo, processador, memória e periféricos de entrada e saída. Suas especificações, como capacidade de processamento, velocidade e quantidade de memória, variam conforme a arquitetura e o fabricante. Entre os modelos mais utilizados, destacam-se o 8051, da Intel; o PIC (*Peripheral Interface Controller*), da Microchip Technology; e o AVR, da Atmel.

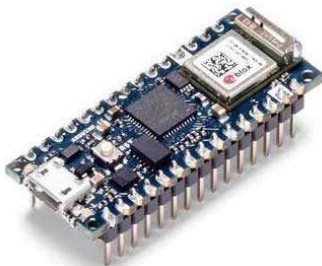
Além disso, os microcontroladores são frequentemente disponibilizados em plataformas de prototipagem, que agregam recursos adicionais e garantem compatibilidade com uma ampla variedade de periféricos, tornando os dispositivos mais robustos e versáteis. Entre essas plataformas, o Arduino se destaca pela sua flexibilidade e excelente relação custo-benefício, sendo amplamente utilizado no desenvolvimento de projetos inovadores.

O Arduino é uma plataforma de prototipagem eletrônica programável que possibilita o desenvolvimento de sistemas interativos e embarcados de forma acessível. Essa plataforma utiliza microcontroladores das famílias Atmel AVR ou ARM e é programada em uma linguagem baseada em C/C++. Sua versatilidade e facilidade de uso fazem com que seja amplamente adotada em aplicações acadêmicas, industriais e de automação (Arduino, 2024).

O Arduino Nano 33 IoT, ilustrado na Figura 2, é uma placa de desenvolvimento compacta projetada para aplicações na Internet das Coisas (IoT). Equipado com o microcontrolador SAMD21 Cortex®-M0+ de 32 bits, oferece 256 KB de memória flash

e 32 KB de SRAM, proporcionando um desempenho eficiente para diversas aplicações. Além disso, a placa integra o módulo u-blox NINA-W102, que possibilita conectividade Wi-Fi 802.11 b/g/n e *Bluetooth Low Energy* (BLE), facilitando a transmissão de dados em tempo real e garantindo versatilidade na comunicação sem fio (Arduino, 2024).

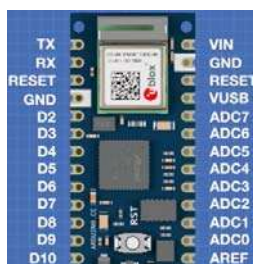
Figura 2: Arduino NANO 33 IoT.



Fonte: (Arduino Official Store, 2025).

Para reforçar a segurança em aplicações IoT, o Arduino Nano 33 IoT incorpora o chip de criptografia ATECC608A, que permite a geração de chaves seguras e autenticação baseada em *hardware*. Com dimensões compactas de 45 x 18 mm e compatibilidade com a IDE do Arduino, a placa pode ser facilmente integrada a diversos projetos. O dispositivo possui 14 pinos digitais, 8 entradas analógicas e 1 saída analógica, além de opções de alimentação via micro-USB ou pelos pinos VIN e 3,3 V, conforme ilustrado na Figura 3.

Figura 3: Descrição dos pinos do Arduino Nano 33 IoT.



Fonte: (DroneBot Workshop, 2022).

2.2.1 BIBLIOTECAS NA PROGRAMAÇÃO

As bibliotecas são coleções de funções, classes e recursos pré-definidos que facilitam o desenvolvimento de *software* para microcontroladores, abstraindo as complexidades do *hardware* e dos protocolos de comunicação. Essas bibliotecas

permitem que o desenvolvedor se concentre na lógica do sistema, sem precisar se preocupar com os detalhes de baixo nível.

No Arduino, as bibliotecas facilitam a integração de funcionalidades avançadas, como comunicação Wi-Fi e manipulação de sensores, sem a necessidade de implementar rotinas de baixo nível. Essas bibliotecas evitam a reescrita de códigos comuns, como a inicialização de protocolos, garantindo a compatibilidade entre dispositivos e placas, além de otimizar o uso dos recursos de *hardware*.

As bibliotecas essenciais para viabilizar a comunicação Wi-Fi, a sincronização temporal e a criação de um servidor HTTP neste trabalho são: **WiFiNINA**, **WiFiUdp**, **RTCZero** e **SPI**.

- A **WiFiNINA** é responsável pela gestão das conexões Wi-Fi no Arduino Nano 33 IoT, utilizando o módulo NINA-W102. Ela permite a conexão a redes Wi-Fi nos modos STA e AP, configuração de IP dinâmico/estático e verificação da intensidade do sinal (RSSI()).
- A **WiFiUdp** facilita a comunicação via protocolo UDP entre os dispositivos da rede, permitindo o envio e recebimento de pacotes sem a necessidade de uma conexão prévia, além de gerenciar portas e *buffers* de dados de maneira eficiente.
- A **RTCZero** controla o Relógio em Tempo Real (RTC) integrado ao Arduino Nano 33 IoT, oferecendo recursos para sincronização com o horário NTP e o armazenamento preciso de datas e horas, com resolução de segundos.
- A **SPI** gerencia a comunicação via barramento SPI (*Serial Peripheral Interface*), sendo essencial para a interação com o módulo Wi-Fi NINA-W102 e facilitando a depuração do código. Essa biblioteca possibilita a transferência de dados em alta velocidade entre o microcontrolador, periféricos e o computador.

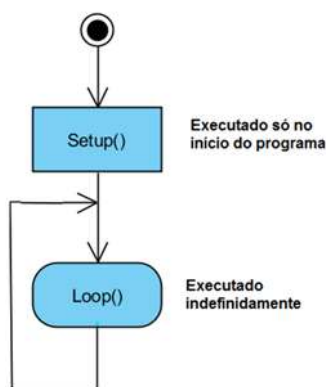
2.2.2 FIRMWARE

A estrutura básica dos *firmwares* desenvolvidos para Arduinos é baseada na IDE Arduino e segue um padrão simplificado, mas eficaz, que facilita o desenvolvimento e implementação de sistemas embarcados. Essa estrutura é composta essencialmente por duas funções obrigatórias: *setup()* e *loop()*.

No *setup()*, o Arduino realiza as configurações iniciais e essa função é executada apenas uma vez, no momento em que a placa é alimentada ou reinicializada. Essa função é similar ao processo de inicialização do BIOS (*Basic Input Output Setup*) em computadores pessoais, onde as configurações de *hardware* e periféricos são definidas. É nesta função que são programados os pinos de entrada e saída do microcontrolador, entre outras definições essenciais. Já no *loop()*, o código é executado repetidamente e é onde reside o código principal da aplicação. A função *loop()* garante que o Arduino execute tarefas contínuas e dinâmicas após a configuração inicial feita no *setup()* (INSTITUTO NCB, 2025).

A Figura 4 ilustra o diagrama de fluxo das funções *setup()* e *loop()*, demonstrando que *setup()* é executada uma vez no início e, em seguida, *loop()* entra em um ciclo contínuo.

Figura 4: Fluxograma da estrutura do *firmware* do Arduino.



Fonte: (INSTITUTO NCB, 2025).

2.3 INTERNET DAS COISAS

A Internet das Coisas (IoT – *Internet of Things*) refere-se à interconexão de dispositivos físicos por meio da internet, permitindo a coleta, transmissão e processamento de dados em tempo real. Essa tecnologia tem papel fundamental em diversas áreas, como automação residencial, industrial, engenharia mecânica e automotiva, possibilitando o monitoramento contínuo de variáveis, a otimização de processos e o aumento da eficiência operacional.

Apesar de sua ampla adoção, não há uma definição única e universal para IoT, pois seu conceito abrange múltiplas abordagens e aplicações em diferentes setores (ROSE, 2015).

O principal objetivo da Internet das Coisas é criar um ecossistema inteligente capaz de conectar bilhões de dispositivos em escala global. No entanto, essa ampla conectividade exige a adoção de medidas rigorosas de segurança para mitigar riscos, prevenir fraudes e garantir a proteção dos dados transmitidos.

Dessa forma, alcançar um alto nível de segurança representa um desafio significativo, sendo um fator essencial para assegurar a confiabilidade da tecnologia e incentivar a adesão de empresas e usuários ao compartilhamento de informações por meio da IoT (YASSER, 2019).

Em competições automobilísticas, a Internet das Coisas desempenha um papel fundamental ao permitir o monitoramento detalhado do desempenho do veículo durante os testes. Esse acompanhamento contínuo facilita o desenvolvimento de soluções mais eficientes e contribui para o aprimoramento do projeto mecânico.

A utilização de microcontroladores compatíveis com IoT, como o Arduino Nano 33 IoT, simplifica a criação de sistemas para coleta e transmissão de dados sem a necessidade de uma infraestrutura complexa. Equipados com módulos de comunicação sem fio, esses dispositivos possibilitam a construção de redes distribuídas, permitindo que múltiplos sensores troquem informações e enviem dados para análise. Dessa forma, é possível viabilizar sistemas de monitoramento remoto de baixo custo e alta eficiência.

Essa integração não apenas viabiliza a coleta de informações precisas e em tempo real, como também permite que equipes estudantis otimizem seus projetos com base em dados concretos. Com isso, promovem-se inovações contínuas e melhorias no desempenho mecânico dos veículos, elevando a competitividade e a eficiência das soluções desenvolvidas.

2.4 REDES DE COMUNICAÇÃO SEM FIO

As redes de comunicação sem fio desempenham um papel essencial na implementação de sistemas IoT, pois permitem a troca de informações entre dispositivos sem a necessidade de conexões físicas. No contexto da telemetria veicular, tecnologias como Wi-Fi e Bluetooth são amplamente empregadas para coletar e transmitir dados de sensores de maneira eficiente e confiável.

Essas redes possibilitam a integração de sistemas de monitoramento capazes de acompanhar parâmetros essenciais do desempenho do veículo. Dessa forma, viabilizam uma análise detalhada e em tempo real, permitindo ajustes estratégicos no funcionamento do veículo e contribuindo para a otimização do desempenho mecânico e eletrônico.

As redes de comunicação sem fio são essenciais para a transmissão de dados em tempo real, possibilitando a interconexão entre dispositivos sem a necessidade de cabos físicos. Dentre as tecnologias disponíveis, o Wi-Fi se destaca por sua alta taxa de transferência de dados e pela capacidade de permitir a comunicação simultânea entre múltiplos dispositivos conectados a uma mesma rede, tornando-se uma solução eficiente para aplicações que demandam transmissão contínua e confiável de informações (AGARWAL, 2020).

Entre as diversas tecnologias disponíveis, o Wi-Fi se destaca por oferecer alta taxa de transmissão de dados, alcance adequado para ambientes internos e externos e ampla compatibilidade com diferentes dispositivos. Essa tecnologia tornou-se essencial em aplicações que exigem transmissão em tempo real de grandes volumes de informações, como na telemetria veicular.

Além disso, a utilização da rede Wi-Fi local permite uma comunicação contínua e sem fio entre o Arduino servidor e os Arduinos clientes. Isso não apenas facilita a troca de dados de forma eficiente, mas também garante que os dados coletados, como o tempo de interrupção de cada sensor, estejam prontamente disponíveis para análise em tempo real. Essa arquitetura oferece flexibilidade ao permitir o acesso remoto aos dados, tornando possível o monitoramento e a gestão dos parâmetros de desempenho de forma prática e sem a necessidade de conexões físicas.

Outra alternativa para comunicação sem fio é o *Bluetooth Low Energy* (BLE), que se destaca pelo baixo consumo de energia. No entanto, suas limitações de alcance, geralmente inferiores a 100 metros, e sua menor taxa de transmissão de dados tornam essa tecnologia menos adequada para aplicações que exigem comunicação entre dispositivos distribuídos por longas distâncias.

Por outro lado, a escolha do Wi-Fi para essas aplicações é motivada principalmente por sua capacidade de conectar múltiplos dispositivos em uma mesma rede. Essa característica permite a criação de uma topologia modular e distribuída, onde cada sensor, equipado com microcontroladores como o Arduino Nano 33 IoT, pode transmitir informações essenciais, como tempos de passagem e outros parâmetros dinâmicos, para uma análise centralizada (LIMA, 2025). Dessa forma, o Wi-Fi não só

garante a comunicação eficiente entre os dispositivos, mas também contribui para a escalabilidade do sistema, permitindo a adição de novos dispositivos à medida que o projeto se expande.

2.5 PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação de dados consistem em um conjunto de regras e convenções usadas para coordenar a troca de informações entre dispositivos em uma rede. Essas normas definem o processo de reconhecimento, o formato e a sequência das mensagens enviadas e recebidas, além de estabelecer como os dispositivos devem reagir ao enviar ou receber dados. Tais protocolos garantem a interoperabilidade entre sistemas heterogêneos, permitindo que dispositivos de diferentes fabricantes e arquiteturas se comuniquem de maneira padronizada. Além disso, os protocolos desempenham um papel crucial na implementação de mecanismos de segurança, como criptografia e autenticação, protegendo as informações transmitidas contra acessos não autorizados ou interceptações (KUROSE & ROSE, 2014).

2.5.1 MODELO TCP/IP

O modelo TCP/IP, ilustrado na Figura 5, *Transmission Control Protocol/Internet Protocol*, consiste em um conjunto de camadas de protocolos amplamente utilizado para a comunicação entre dispositivos. Esse conjunto é fundamental para a troca de dados na internet e em redes privadas. No modelo TCP/IP, antes de iniciar a troca de dados, os dispositivos devem estabelecer uma conexão confiável por meio de um processo chamado de *Acknowledgement* (ACK), ou confirmação de recebimento. Esse procedimento garante que ambas as partes envolvidas na comunicação estejam prontas para a troca de informações, estabelecendo a confiança necessária para o envio efetivo dos dados (KUROSE & ROSE, 2014).

O TCP e o IP são protocolos distintos que, juntos, formam a família TCP/IP, mas não são os únicos responsáveis pelas comunicações na rede. Cada protocolo dentro dessa família tem a função de habilitar um recurso específico da rede. O TCP (*Transmission Control Protocol*) é o protocolo responsável por garantir uma comunicação confiável entre dispositivos. Esse protocolo assegura que os pacotes de dados enviados cheguem ao destino correto, na ordem correta, e sem erros. Para garantir essa confiabilidade, o TCP

utiliza mecanismos como controle de congestionamento e retransmissão de pacotes perdidos, assegurando a integridade e a continuidade da comunicação.

Figura 5: Camadas do modelo TCP/IP



Fonte: (VITOR PÁDUA/TECNOBLOG)

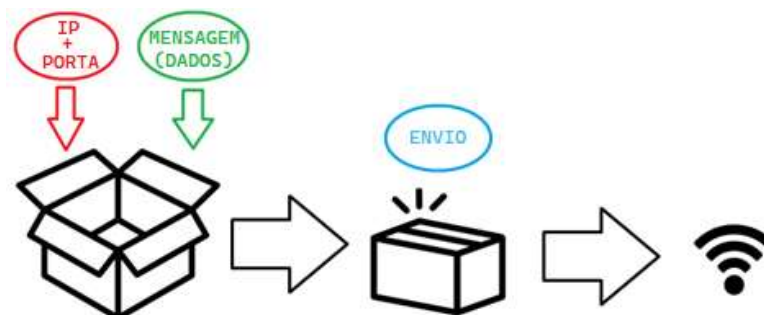
Por sua vez, o IP (*Internet Protocol*) é responsável pelo endereçamento e roteamento dos pacotes de dados na rede. Esse protocolo determina o caminho que os pacotes devem seguir para chegar ao seu destino, utilizando endereços IP exclusivos para identificar cada dispositivo. A combinação do TCP e do IP forma a espinha dorsal da internet moderna, sendo crucial para aplicações em que a precisão dos dados é essencial, como transações bancárias, e-mails e transferência de arquivos. A família de protocolos TCP/IP engloba diversos protocolos essenciais, incluindo TCP, IP, UDP (*User Datagram Protocol*), DNS (*Domain Name System*), ICMP (*Internet Control Message Protocol*), ARP (*Address Resolution Protocol*), RIP (*Routing Information Protocol*), SMTP (*Simple Mail Transfer Protocol*), Telnet, entre outros (NEIVA, 2011).

2.5.2 PROTOCOLO UDP

O Protocolo UDP (*User Datagram Protocol*) é um dos principais protocolos da camada de transporte do modelo TCP/IP, priorizando a velocidade e a eficiência. Ao ser orientado a datagramas, o UDP permite a comunicação entre dispositivos sem a necessidade de estabelecer uma conexão prévia, caracterizando-se como um protocolo sem estado (*stateless*). Isso simplifica o processo de comunicação, pois os pacotes são enviados diretamente ao destinatário sem confirmação de recebimento ou retransmissão em caso de perda. Sua simplicidade e baixa latência são suas principais características, tornando-o altamente eficiente e rápido. Por essas razões, o UDP é ideal para aplicações que exigem alta performance, como *streaming* de áudio e vídeo, transmissões ao vivo,

jogos online e sistemas de telemetria, onde a perda eventual de pacotes não compromete significativamente a funcionalidade (ARNETT & DULANEY, 1996). Na Figura 5, é ilustrado o fluxograma da estrutura de comunicação via protocolo UDP.

Figura 6: Analogia do funcionamento comunicação via UDP.



Fonte: (PORTAL VIDA DE SILICIO, 2025).

2.5.3 COMPARAÇÃO ENTRE UDP E TCP

A principal diferença entre o UDP e o TCP está no nível de confiabilidade e na abordagem de comunicação. Enquanto o TCP é projetado para conexões confiáveis, realizando um “*handshake*” inicial e garantindo a entrega de dados de forma ordenada e sem perdas, o UDP prioriza a velocidade e eficiência, dispensando a verificação de erros e a ordem dos pacotes. Como resultado, o UDP é mais rápido, devido à sua simplicidade e ausência de controles adicionais, enquanto o TCP sacrifica a velocidade para assegurar a entrega confiável e ordenada dos pacotes. Além disso, o UDP utiliza menos recursos computacionais, o que o torna mais adequado para dispositivos de baixa potência ou redes congestionadas. Em contrapartida, o TCP consome mais recursos, devido às verificações extras que realiza para garantir a integridade da comunicação (TANENBAUM & WETHERALL, 2021).

Portanto, o UDP é ideal para aplicações onde a velocidade e a simplicidade da comunicação são mais importantes do que a garantia da entrega de cada pacote individualmente. Esse protocolo é amplamente utilizado em cenários onde a perda de pacotes não impacta significativamente o desempenho, como em transmissões de áudio e vídeo em tempo real. Em contraste, o TCP é um protocolo orientado à conexão, oferecendo um conjunto de recursos específicos para estabelecer, manter e finalizar a comunicação de forma confiável. O TCP é capaz de identificar e corrigir erros, evitar sobrecarga no fluxo de dados, negociar o tamanho dos pacotes e realizar a retransmissão

de pacotes danificados, assegurando a entrega ordenada e sem falhas de dados (NEVES & ORENSTEIN, 2007). A comparação entre as características desses protocolos está descrita na Tabela 1.

Tabela 1 – Comparação TCP x UDP.

Característica	TCP	UDP
Confiabilidade	Alta (ACK, controle de erros, ...)	Baixa (Não garante entrega em conexões instáveis)
Velocidade	Mais lento (mais recursos)	Rápido (Simples, baixa latência)
Ordenação dos pacotes	Garante a ordem de chegada dos pacotes	Não garante a ordenação
Aplicação	Transações bancárias, transferência de arquivos, etc.	<i>Streaming</i> , jogos online, telemetria, etc.

Fonte: Adaptado de (SANTOS, 2015)

2.6 SINCRONIZAÇÃO TEMPORAL E RTC

A sincronização temporal é fundamental para assegurar a precisão de sistemas que dependem de eventos temporais registrados. Em contextos como a telemetria veicular, onde sensores distribuídos coletam dados em diversos pontos do percurso, pequenas discrepâncias temporais podem afetar significativamente os cálculos de velocidade e aceleração. Para superar esse desafio, técnicas como o *Network Time Protocol* (NTP) e a implementação de relógio em tempo real (RTC) são cruciais, pois garantem uma referência temporal precisa e consistente, permitindo que os dados coletados sejam corretamente alinhados e analisados.

2.6.1 NETWORK TIME PROTOCOL

O *Network Time Protocol* (NTP) é amplamente utilizado para a sincronização de tempo em redes, operando na camada de aplicação do modelo TCP/IP. Usando o protocolo UDP na porta padrão 123, o NTP permite que dispositivos ajustem seus relógios com uma precisão de milissegundos em redes públicas e até microssegundos em redes locais.

O NTP é baseado em uma hierarquia de servidores de tempo organizados em estratos, sendo que os servidores de estrato 1 estão diretamente conectados a fontes de tempo de alta precisão, como relógios atômicos ou sinais de GPS, e distribuem essa

informação para servidores de estratos inferiores. Além disso, o NTP tem a capacidade de identificar servidores de tempo não confiáveis e de ajustar os relógios de forma gradual, evitando alterações abruptas no tempo, o que é crucial para aplicações que exigem consistência nos registros de eventos, como sistemas financeiros, redes de telecomunicações e sistemas distribuídos (Mills, 1991).

O funcionamento do NTP é baseado na troca de mensagens entre clientes e servidores de tempo, que operam em uma hierarquia de camadas chamadas “*stratum*”. Nessa hierarquia, os dispositivos ajustam seus relógios internos de acordo com um servidor de tempo com maior precisão.

Os servidores de *stratum* 1 são diretamente sincronizados com fontes de tempo de alta precisão, como relógios atômicos ou sinais de GPS, enquanto os servidores de *stratum* 2, 3, etc., propagam a informação para dispositivos finais em níveis inferiores. O protocolo leva em consideração os atrasos na comunicação entre os dispositivos e as compensações necessárias, o que assegura maior exatidão nos ajustes de tempo. Em sistemas típicos, o NTP pode oferecer sincronização com precisão de milissegundos ou até mesmo microssegundos, dependendo da qualidade da rede e da proximidade dos servidores de tempo.

2.6.2 REAL TIME CLOCK

Os *Real Time Clocks* (RTC) são dispositivos especializados em manter a contagem precisa do tempo, mesmo quando o sistema principal está desligado, graças à utilização de baterias de backup. Esses dispositivos são fundamentais em sistemas que necessitam de um acompanhamento contínuo do tempo, como agendamento de tarefas ou registro de eventos. Em microcontroladores, os RTCs são amplamente utilizados para garantir a sincronização de operações e precisão temporal em sistemas embarcados. Além disso, eles desempenham um papel crucial em sistemas que requerem registro de eventos com precisão, como em sistemas de monitoramento e controle (ALCIATORE & HISTAND, 2011).

Microcontroladores como o Arduino Nano 33 IoT vêm equipados com circuitos RTC integrados, permitindo a manutenção da contagem do tempo mesmo em caso de quedas de energia. No entanto, os osciladores de cristal de baixo custo presentes nesses dispositivos podem apresentar deriva temporal, o que resulta em um erro acumulado de alguns segundos por dia. Para corrigir essa deriva e garantir a precisão do tempo, é

necessária a sincronização periódica. A sincronização pode ser realizada de 2 (duas) maneiras principais:

- **Sincronização via Servidor NTP Externo:** O Arduino Nano 33 IoT, atuando como unidade central do sistema, pode se conectar a servidores NTP na internet para obter uma referência de tempo confiável. Este método é eficiente, garantindo uma sincronização precisa com fontes externas de tempo, mas depende de uma conexão estável com a internet.
- **Sincronização por Broadcast Local:** Em cenários onde o acesso contínuo à internet não está disponível, um dos microcontroladores pode ser configurado para atuar como servidor de tempo local. Ele distribuirá informações de *timestamp* para os outros dispositivos da rede via Wi-Fi, permitindo a sincronização de todos os dispositivos de maneira independente de uma infraestrutura externa e mantendo a precisão dentro da rede local.

2.7 SERVIDOR HTTP E API REST PARA MONITORAMENTO DOS DADOS

2.7.1 SERVIDOR HTTP

Um servidor HTTP (*Hypertext Transfer Protocol* – Protocolo de Transferência de Hipertexto) é um sistema responsável por receber e responder a requisições feitas por clientes, como navegadores web, utilizando o protocolo HTTP. Esse protocolo foi desenvolvido para a transferência de informações entre dispositivos conectados via rede, sendo fundamental no fluxo de dados entre clientes e servidores. Ele permite que aplicações web e navegadores façam solicitações e que os sistemas de computadores enviem respostas para essas requisições (FOROUZAN, 2008). As principais características de um servidor HTTP incluem:

- **Recepção de Requisições:** O servidor recebe solicitações no formato HTTP/HTTPS, enviadas por clientes (como navegadores ou outros sistemas de *software*).
- **Processamento:** O servidor analisa os dados contidos na solicitação, identificando o método HTTP (como GET, POST, PUT, DELETE) e os parâmetros associados, para determinar como tratar a requisição.

- **Envio de Respostas:** O servidor retorna o recurso solicitado (como uma página HTML, imagem, ou outro arquivo) ou uma mensagem de erro, caso o recurso não esteja disponível ou algum problema tenha ocorrido durante o processamento da solicitação.

O protocolo HTTP opera na camada de aplicação do modelo TCP/IP, sendo responsável por transferir dados entre servidores e clientes na rede. No contexto de um servidor HTTP, o protocolo TCP é utilizado para estabelecer conexões confiáveis, garantindo que os dados sejam entregues de forma correta e ordenada. Já o protocolo IP gerencia o endereçamento e o roteamento dos pacotes de dados, localizando os dispositivos na rede e permitindo que as mensagens cheguem ao destino correto.

O servidor HTTP recebe solicitações de recursos, como páginas HTML (*HyperText Markup Language*), dados estruturados em formato JSON (*JavaScript Object Notation*), imagens, scripts e outros tipos de conteúdo. Após processar a solicitação, o servidor retorna o conteúdo solicitado ao cliente, juntamente com informações adicionais, como o status da requisição (por exemplo, código de status 200 para “OK” ou 404 para “Não encontrado”) e os cabeçalhos HTTP, que contêm metadados sobre a resposta, como tipo de conteúdo, tamanho e data de modificação (KUROSE & ROSE, 2014).

Esses servidores desempenham um papel essencial na entrega de recursos e conteúdo ao usuário, proporcionando flexibilidade e acessibilidade no tratamento e monitoramento dos dados. Além disso, facilitam a comunicação entre dispositivos cliente e servidores por meio de métodos que garantem a interoperabilidade e escalabilidade de sistemas web. Os principais métodos, que orientam o funcionamento do servidor HTTP e determinam qual ação será executada, incluem:

- **GET:** Solicita dados de um recurso específico. Este método é utilizado quando o cliente deseja ler ou obter informações, como a leitura de sensores ou a obtenção de uma página da web.
- **POST:** Envia dados para serem processados pelo servidor. Este método é comumente usado quando o cliente deseja enviar dados, como configurações do sistema ou informações de formulários.

Quando um cliente, como um navegador web, faz uma solicitação a uma URL específica, essa requisição é processada pelo servidor HTTP. O servidor HTTP interpreta o pedido recebido, localiza o recurso correspondente (como uma página HTML, imagem

ou script) e o retorna ao cliente em forma de resposta. Antes de atender a solicitação, o servidor pode aplicar regras de segurança, como autenticação, para garantir que o cliente tenha permissão para acessar o recurso solicitado.

Além disso, o servidor HTTP pode implementar técnicas de compactação de dados, quando suportado, para otimizar o tráfego e reduzir a latência, tornando a comunicação mais eficiente. Em sistemas mais avançados, o servidor pode executar scripts do lado do servidor ou interagir com bancos de dados para gerar respostas dinâmicas. Isso permite que o servidor produza conteúdo personalizado (como páginas de usuários específicos) ou execute consultas em tempo real, tornando-o crucial para a entrega de conteúdo dinâmico e interativo na web.

Portanto, o servidor HTTP não apenas serve conteúdo estático, mas também permite a criação de aplicações web dinâmicas e personalizadas, que respondem de forma flexível às solicitações dos usuários (KUROSE & ROSE, 2014).

2.7.2 API REST

Uma API (*Interface de Programação de Aplicações*) é uma interface que permite a comunicação entre sistemas de *software*, geralmente por meio do protocolo HTTP. Muitas APIs seguem os princípios da arquitetura REST (*Representational State Transfer*), que visa criar serviços web escaláveis, simples e de fácil manutenção. Os principais princípios REST são:

- **Statelessness (Sem Estado):** Cada requisição feita ao servidor contém todas as informações necessárias para o seu processamento. O servidor não armazena nenhum estado entre as requisições, o que significa que cada solicitação é independente e não depende de informações de solicitações anteriores.
- **Uniformidade de Interface:** A API utiliza de maneira consistente os métodos HTTP (como GET, POST, PUT, DELETE) e formatos de resposta, como JSON ou HTML. Essa uniformidade facilita o entendimento e a integração de diferentes sistemas.
- **Recursos Identificáveis por URLs:** Cada dado ou recurso da API é acessado por meio de identificadores exclusivos (URLs). Por exemplo, em uma API de sensores, um recurso pode ser acessado com a URL `http://[IP]/sensores/30m`, onde

"30m" representa a identificação do recurso. Isso garante que cada recurso tenha uma localização única na rede.

Esses princípios tornam as APIs REST mais simples, flexíveis e eficientes para integração entre diferentes sistemas, facilitando a troca de dados e funcionalidades entre aplicativos.

2.8 SENSOR DE PROXIMIDADE FOTOELÉTRICO

Os sensores de proximidade fotoelétricos são dispositivos amplamente utilizados na automação industrial para detectar a presença, ausência ou movimentação de objetos sem contato físico. O funcionamento desses sensores baseia-se na emissão e recepção de um feixe de luz, geralmente infravermelho ou visível, que interage com o objeto-alvo. Dependendo do princípio de detecção, os sensores fotoelétricos podem operar nos seguintes modos:

- **Modo Difuso:** O sensor emite um feixe de luz e detecta a reflexão do feixe de volta ao receptor quando um objeto está presente. O objeto não precisa ser altamente reflexivo para que o sensor detecte sua presença, o que torna esse modo ideal para detecção de objetos próximos.
- **Modo Retrorreflexivo:** O sensor emite um feixe de luz e o direciona para um refletor colocado em frente ao sensor. O feixe de luz reflete no refletor e volta ao sensor. Quando um objeto interrompe esse feixe, ele é detectado.
- **Modo Barreira:** Neste modo, o sensor emite um feixe de luz contínuo, e a presença de um objeto é detectada quando ele interrompe o feixe. Este modo é ideal para detectar objetos que atravessam uma linha definida.

Esses sensores são essenciais para diversas aplicações na indústria, incluindo controle de processos, monitoramento de movimentos e automação de sistemas, oferecendo precisão, rapidez e redução do desgaste mecânico.

Dentre os sensores fotoelétricos, um dos tipos mais comuns é o sensor PNP (positivo-negativo-positivo), cuja configuração elétrica fornece o sinal à carga de saída proveniente do seu polo positivo (+VCC). Isso significa que, quando o sensor detecta um objeto, ele fecha o circuito elétrico, conectando a carga ao potencial positivo da fonte de

alimentação (+VCC). Em outras palavras, a saída do sensor PNP gera um sinal positivo sempre que a detecção é realizada, permitindo a ativação ou o acionamento de dispositivos ligados ao sistema de controle.

Esse tipo de sensor é amplamente utilizado em diversas aplicações industriais, pois proporciona um método simples e eficaz para a detecção de objetos sem contato físico, utilizando um sinal positivo para interagir com sistemas de controle.

Na Figura 4, ilustra-se um exemplo de um sensor de proximidade difuso do tipo PNP. Estes sensores PNP difusos, por sua vez, possuem 3 (três) fios de conexão padrão:

- **Marrom (+VCC):** Alimentação positiva, com uma faixa de tensão geralmente entre 5V e 30V DC.
- **Azul (GND):** Conexão ao terra da fonte de alimentação.
- **Preto (Saída PNP):** Quando o sensor detecta um objeto, o fio preto conecta-se ao positivo (+VCC), transmitindo o sinal para a carga.

Este modelo tem uma corrente de saída especificada pelo fabricante de 150mA e conta com proteção IP67, oferecendo resistência à água.

Figura 7: Sensor de proximidade fotoelétrico difuso PNP.



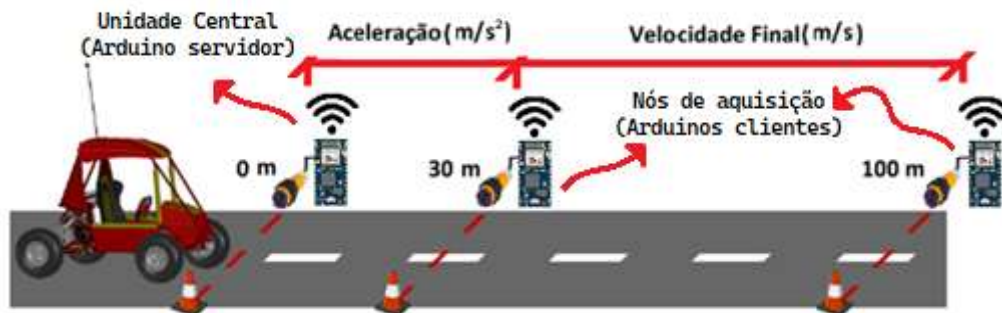
Fonte: (PBVIRTUAL, 2025).

3. METODOLOGIA

Este trabalho propõe o desenvolvimento de uma bancada de teste para avaliação da aceleração e velocidade em veículos *off-road* do tipo BAJA, visando contribuir para a preparação e análise de desempenho dos veículos nas competições BAJA SAE, que possuem grande relevância para a comunidade acadêmica. A avaliação da aceleração e velocidade são aspectos fundamentais dentro da competição e representam o foco deste estudo.

Para a implementação da bancada de testes, foram adotadas tecnologias acessíveis, como a Internet das Coisas (IoT), combinadas com a conectividade Wi-Fi. Isso possibilita a criação de uma rede distribuída, como ilustrada na Figura 7, composta por uma Unidade Central (servidor) e múltiplos Nós de Aquisição (clientes), que coletam os dados gerados durante os testes da aceleração e velocidade descritos na Seção 2.1.2.

Figura 8: Posicionamento dos sensores ao longo do percurso.



Fonte: Adaptado de (RODRIGUES, 2011).

Os Nós de Aquisição (clientes) consistem em Arduinos Nano 33 IoT conectados a sensores de proximidade fotoelétricos ópticos difusos do tipo PNP, com alcance de 0 a 100 cm e resolução temporal de 1 ms. Os sensores são posicionados em pontos estratégicos do percurso (0 m, 30 m e 100 m), de modo que cada sensor detecta a passagem do veículo e envia um sinal elétrico para a porta digital do respectivo Arduino cliente. Este, por sua vez, sinaliza o evento correspondente à Unidade Central via protocolo UDP sobre Wi-Fi.

A Unidade Central (servidor) é responsável por receber os pacotes UDP enviados pelos nós de aquisição, consolidar os dados, registrar o horário de detecção, calcular os intervalos de tempo entre as detecções e, simultaneamente, disponibilizar as informações para monitoramento por meio de um servidor HTTP com API REST. Esse

servidor retorna, como resposta, uma página HTML ou dados estruturados em JSON (padrão), de acordo com os parâmetros informados na requisição.

Além disso, a Unidade Central realiza a sincronização temporal utilizando o protocolo NTP, ajustando seu relógio (RTC) interno, caso haja conectividade com a Internet, garantindo “*timestamps*” precisos para cada evento. A Unidade Central também funciona indiretamente como um nó de partida, pois contém um sensor de proximidade fotoelétrico de largada que sinaliza o momento em que o veículo parte do repouso (ponto de partida: 0 metros). Após o evento de largada, a Unidade Central permanece monitorando mensagens enviadas por cada cliente através do protocolo UDP, sinalizando a detecção da passagem do veículo nas posições de 30 m ou 100 m.

3.1 COMPONENTES E CIRCUITOS

A relação dos componentes utilizados na bancada de teste para avaliação da aceleração e velocidade está apresentada na Tabela 2.

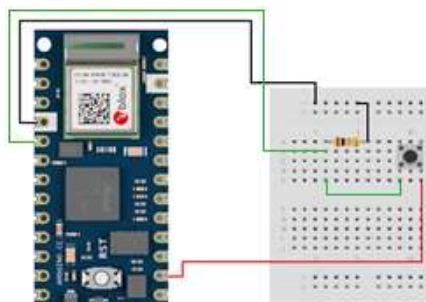
Tabela 2 – Relação de componentes.

Componentes	Qtd	Especificação	Função
Arduino Nano 33 IoT	03	Microcontrolador SAMD21, Wi-Fi/BLE	Nós de aquisição e servidor central
<i>Powerbank</i>	03	Capacidade: 2000mAh Saída: 5V, 2A	Alimentação portátil
Resistor	03	10 k Ω	Resistor de <i>pull-down</i>
Botão tátil	03	<i>Push-button</i> para 3,3 V	Iniciar medição
Protoboard	03	830 furos	Comportar os componentes
<i>Jumpers</i>	20	Conectores 24 AWG	Realizar a conexão elétrica dos componentes

Fonte: Autoria própria.

Para fins de testes e simplificação no desenvolvimento da bancada, os sensores de proximidade fotoelétricos foram substituídos por botões de pressão (*push-buttons*), que simularão o evento de detecção realizado pelos sensores. A Figura 8 ilustra o esquema de ligação desses botões, juntamente com o resistor de *pull-down* de 10 K Ω , no Arduino servidor, utilizando uma placa de ensaio (*protoboard*).

Figura 9: Esquema de ligação do Arduino servidor e clientes.



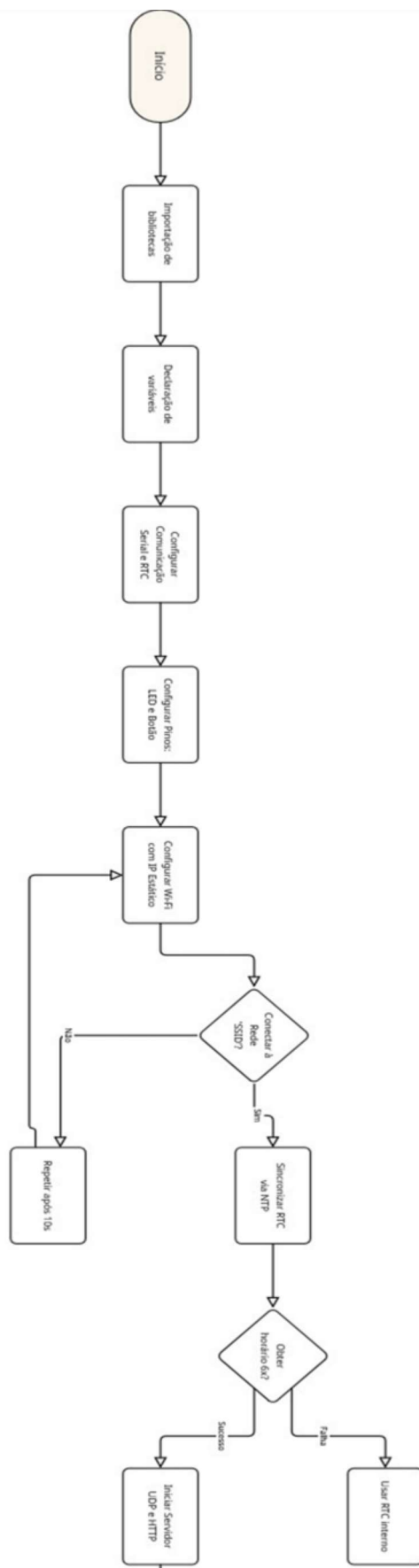
Fonte: (Autoria própria, 2025).

3.1.1 DESENVOLVIMENTO DOS FIRMWARES

Todos os códigos utilizados neste projeto estão listados no Apêndice A, B e C. Os *firmwares* foram desenvolvidos essencialmente utilizando a linguagem C++ na IDE Arduino, estruturados em algumas etapas principais, comuns a ambos os blocos: a importação das bibliotecas e declaração das variáveis necessárias, rotina de configuração (*setup*) e a rotina principal de repetição infinita (*loop*). Além disso, linguagens complementares, como HTML e CSS, também foram empregadas na implementação do monitoramento dos dados.

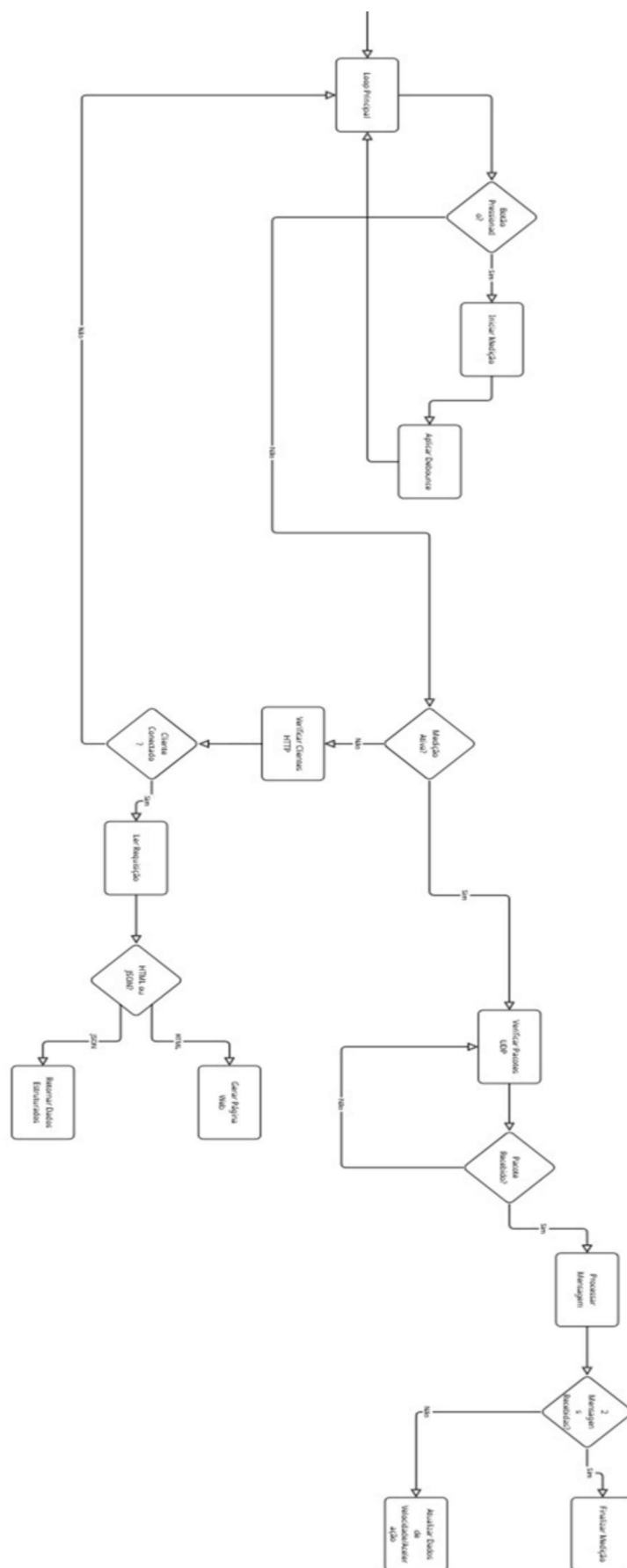
3.1.1.1 UNIDADE CENTRAL (ARDUINO SERVIDOR)

Nas Figuras 9 e 10, ilustram-se os fluxogramas que representam a lógica de funcionamento do *firmware* da Unidade Central.

Figura 10: Rotina de *setup()* do *firmware* da unidade central.

Fonte: (Autoria própria, 2025).

Figura 11: Rotina de *loop()* do *firmware* da unidade central.



Fonte: (Autoria própria, 2025).

Após a importação das bibliotecas necessárias, mencionadas na seção de bibliotecas no tópico de microcontroladores do Capítulo 2 (Referencial Teórico), são definidas algumas variáveis pertinentes à aplicação, como as *strings* das mensagens recebidas dos clientes, contadores de mensagens, índice de mensagens, *string* de marcações de tempo, variáveis de configuração da conexão Wi-Fi, variáveis de estado de sincronização, fuso horário, configuração da porta do servidor HTTP, entre outras.

Após essa etapa, o Arduino entra na rotina de configuração (*setup*), na qual o dispositivo inicia a comunicação serial, utilizando a biblioteca SPI para depuração; configura os pinos do LED como saída e o botão como entrada; inicializa o RTC por meio da biblioteca RTCZero; estabelece a configuração de rede (IP estático, gateway, máscara e DNS) e tenta estabelecer uma conexão à rede Wi-Fi local utilizando a biblioteca WiFinINA. O servidor aguarda até que a conexão seja estabelecida, exibindo o IP e a potência do sinal (RSSI).

Caso a conexão seja bem-sucedida, o Arduino Servidor tentará sincronizar seu relógio interno com um servidor NTP, realizando até 6 (seis) tentativas. Se a sincronização for bem-sucedida, o tempo será ajustado e utilizado para gerar as marcações temporais dos eventos. Caso contrário, será exibida uma mensagem sinalizando a falha na sincronização.

Em seguida, o serviço UDP é iniciado na porta definida, e o servidor HTTP é ativado para escutar requisições na porta 80. Após isso, o servidor entra na rotina de loop.

Na rotina *loop*, o sistema monitora o estado de um botão, utilizado para sinalizar o início de um percurso. Ao detectar o botão pressionado e caso a medição não esteja ativa, a função *iniciarMedicao()* é chamada, registrando o tempo inicial e ajustando os indicadores de *status*. Quando a medição está ativa, o servidor verifica a chegada de pacotes UDP. Cada pacote recebido corresponde à detecção de um sensor, e a função *processarMensagem()* é chamada para:

- Calcular o tempo decorrido desde o início da medição utilizando a função *millis()*;
- Registrar o *timestamp* de detecção utilizando o tempo sincronizado;
- Incrementar um contador de mensagens recebidas e o índice das mensagens;
- Acionar um *feedback* visual, piscando o LED embutido na placa.

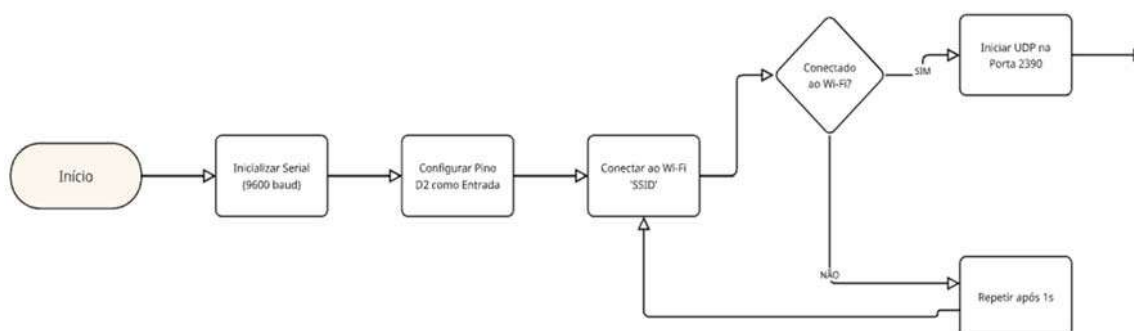
Após a recepção das mensagens de todos os sensores (neste caso, duas mensagens correspondentes aos sensores de 30 m e 100 m), a função `finalizarMedicao()` é executada para calcular os intervalos entre os eventos e exibir os resultados. Paralelamente, o servidor HTTP aguarda a chegada de requisições de clientes. Quando uma requisição é recebida, o *firmware*:

- Responde a requisições GET com uma página HTML atualizada a cada 3 segundos, exibindo tempos, velocidades e acelerações em *cards* estilizados com CSS; ou no formato JSON, com dados estruturados para integração com ferramentas externas, como scripts Python ou bancos de dados), conforme o parâmetro *format* atribuído na requisição;
- Prepara a resposta dinâmica, incorporando os dados consolidados dos sensores, como tempos de detecção, *timestamps*, intervalos calculados, velocidade média, aceleração, etc.;
- Envia a resposta ao cliente e encerra a conexão.

3.1.1.2 ARDUINOS CLIENTES

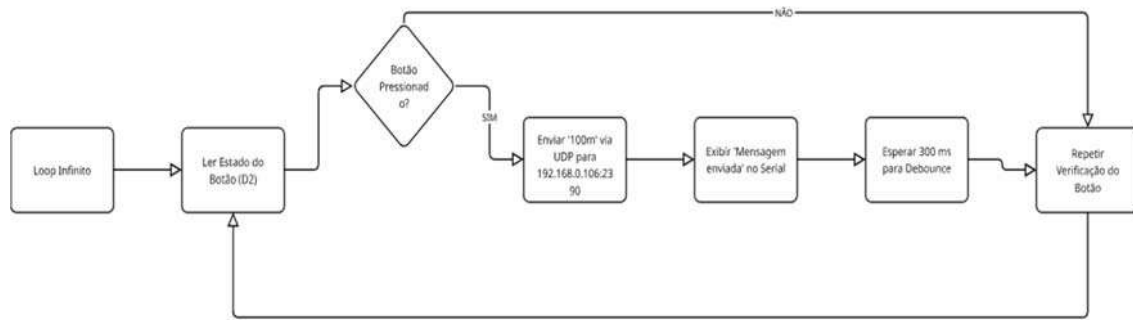
Nas Figuras 11 e 12, ilustram-se os fluxogramas que representam a lógica de funcionamento do *firmware* dos Nós de Aquisição (clientes).

Figura 12: Rotina de `setup()` do *firmware* dos nós de aquisição de dados



Fonte: (Autoria própria, 2025).

Figura 13: Rotina de *loop()* do *firmware* dos nós de aquisição de dados



Fonte: (Autoria própria, 2025).

O código inicia importando a biblioteca WiFiNINA e definindo as variáveis dos parâmetros da comunicação via UDP, tal como na Unidade Central: porta local do cliente, o IP do Arduino servidor, que recebe a mensagem, e sua porta local; bem como os dados de *login* e a senha da rede Wi-Fi e a variável do estado da conexão. Além disso, também foi declarada uma variável para contagem de detecções, para fins de depuração do código.

Posteriormente, o Arduino cliente entra então em sua rotina de configuração (*setup*) e tenta estabelecer uma conexão à rede Wi-Fi. O cliente aguarda até que a conexão seja estabelecida, exibindo o IP e a potência do sinal (RSSI) via comunicação serial, no mesmo padrão da unidade central. Somente após essa etapa, a comunicação UDP é inicializada.

Em seguida, o fluxo do código segue para a rotina de repetição infinita (*loop*), na qual o dispositivo verifica continuamente se há detecção por parte do sensor de proximidade (neste caso, simulado pelos botões) conectados à porta digital 2.

4. RESULTADOS

Como resultado, foi possível simular o percurso do veículo e atestar a fidelidade das medições. A Figura 13 exibe a resposta do Arduino servidor após o botão de largada ser pressionado, marcando o início do percurso do veículo.

Figura 14: Resposta da Unidade Central via Serial

```
-----
Conectado ao Wi-Fi
Endereço IP do servidor: 192.168.0.106
signal strength (RSSI): -60 dBm
Relógio sincronizado: (16:54:20) - 21/03/2025

--- Medição Iniciada ---
Início do percurso! | ((16:54:23) - 21/03/2025)
Aguardando sensores...
```

Fonte: (Autoria própria, 2025).

Após um determinado tempo, os botões correspondentes às distâncias de 30 m e 100 m, conectados aos clientes, foram pressionados, simulando a passagem do veículo e marcando o final do percurso. A Figura 14 ilustra a resposta via serial da Unidade Central.

Figura 15: Resposta da Unidade Central após a sinalização dos clientes

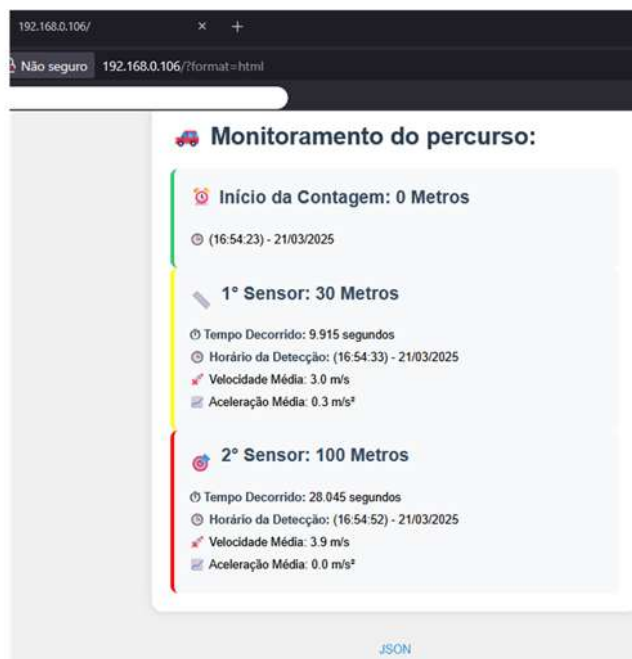
```
Sensor 1 detectado: 30m | Tempo: 9.915s ((16:54:33) - 21/03/2025)
Sensor 2 detectado: 100m | Tempo: 28.045s ((16:54:52) - 21/03/2025)

Resultados finais:
Tempo até primeiro sensor: 9.915s
Tempo até segundo sensor: 28.045s
Intervalo entre sensores: 18.130s
-----
```

Fonte: (Autoria própria, 2025).

Após verificar o endereço IP fornecido no início do programa pelo Arduino servidor, ou simplesmente utilizando o IP estático definido nas configurações, é possível acessar a API do servidor. Ao fazer a requisição a partir de um dispositivo móvel (smartphone) utilizando um navegador e fornecendo o parâmetro format=html (por exemplo, <http://192.168.0.1/?format=html>), foi retornada a seguinte resposta, exibida na Figura 15.

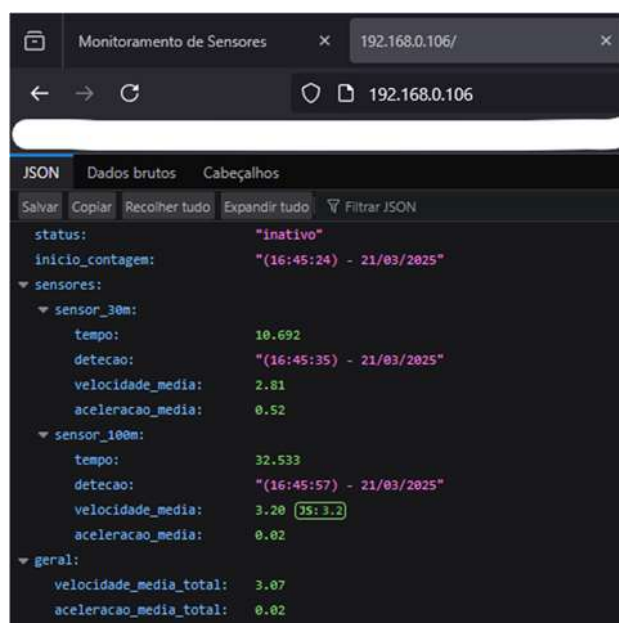
Figura 16: Resposta HTML a partir de um *smartphone*



Fonte: (Autoria própria, 2025).

Agora, ao digitar simplesmente o endereço IP (192.168.0.1) no navegador Mozilla Firefox em um computador ou notebook, sem fornecer nenhum parâmetro, são exibidos, por padrão, os dados estruturados no formato JSON, conforme representado na Figura 16.

Figura 17: Resposta JSON utilizando o *Firefox*.



Fonte: (Autoria própria, 2025).

5. CONCLUSÃO E PERSPECTIVAS FUTURAS

5.1 CONCLUSÃO

Este trabalho desenvolveu uma bancada de teste para avaliação do desempenho de veículos *off-road* do tipo BAJA SAE, com foco nas provas de aceleração e velocidade. Para isso, foram empregadas tecnologias acessíveis, como a plataforma Arduino Nano 33 IoT e sensores fotoelétricos ópticos difusos do tipo PNP, possibilitando a aquisição precisa de dados cinemáticos.

A integração da tecnologia IoT e da conectividade Wi-Fi viabilizou uma rede distribuída de aquisição de dados, permitindo análises detalhadas e em tempo real. A implementação de comunicação sem fio via UDP, sincronização temporal com NTP e disponibilização dos dados por meio de uma API REST resultou em um sistema de telemetria preciso e eficiente. O sistema destacou-se pela confiabilidade na medição, com sincronização temporal apresentando erro inferior a 10 ms e algoritmos de *debounce* minimizando ruídos nos sensores. Além disso, sua interface web viabilizou o monitoramento remoto e a integração com outras ferramentas analíticas.

A solução desenvolvida se mostrou modular e escalável, permitindo a incorporação de novos sensores sem grandes modificações na estrutura central. Com um custo de hardware inferior ao de sistemas comerciais e desempenho comparável a tais, tornando-se uma alternativa viável e acessível.

Os resultados obtidos demonstram o potencial desta abordagem para a otimização contínua dos protótipos veiculares, fornecendo dados quantitativos para ajustes mecânicos e estratégias de pilotagem em competições. A versatilidade do sistema, aliada à sua compatibilidade com formatos de dados como HTML e JSON, reforça sua aplicabilidade e integração com outras ferramentas de análise, consolidando-se como uma solução eficiente para a telemetria de veículos *off-road*.

5.2 PERSPECTIVAS FUTURAS

Apesar dos bons resultados, melhorias podem ser implementadas para otimizar a precisão, segurança e eficiência do sistema. Entre os principais aprimoramentos propostos, destacam-se:

- **Otimização da Comunicação:** adoção de protocolos híbridos (MQTT + UDP) e redes *mesh* para maior alcance.
- **Aprimoramento da Robustez:** filtros para redução de ruídos e *bluetooth low energy* (BLE) como canal de backup para evitar perda de dados.
- **Segurança na Transmissão:** autenticação na API, validação de pacotes e filtragem por IP.
- **Sustentabilidade:** uso de painéis solares e modos de baixo consumo energético.
- **Validação em Competições:** testes em provas oficiais da BAJA SAE para refinamento do sistema.

Esses aprimoramentos tornarão a solução mais precisa, segura e escalável, ampliando sua aplicabilidade e contribuindo para o desenvolvimento contínuo dos protótipos veiculares.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO NANO 33 IoT <<https://docs.arduino.cc/hardware/nano-33-iot/#tech-specs/>> . Acesso em 30/12/2024 às 10:43

Arduino Official Store <<https://store.arduino.cc/en-cy/products/arduino-nano-33-iot-with-headers>> Acesso em 21/02/2025 às 22:58.

AGARWAL, Tarun (2020). Different Types of Wireless Communication with Applications. Elprocus. Disponível em: <<https://www.elprocus.com/types-of-wireless-communication-applications/>>. Acesso em 30/12/2024 às 11:14

ARNETT, Flint Matthew; DULANEY, Emmett (1996). “Desvendando o TCP/IP”. 2ª edição. Editora Campus

DRONEBOT WORKSHOP, Arduino Nano 33 IoT - Getting Started, 2022. Disponível em https://www.youtube.com/watch?v=KjETPxv_VR4&ab_channel=DroneBotWorkshop. Acesso em: 16 de Janeiro de 2025.

FOROUZAN, Behrouz A. (2008). “Comunicação de Dados e Redes de Computadores”. 4ª ed. São Paulo. McGrawHill Brasil

GOUVEIA, Paulo R. N. T. Convergência de Redes Sem Fios para Comunicações M2M e Internet das Coisas em Ambientes Inteligentes. Covilhã: 2013.

KUROSE, James F.; ROSS, Keith W. (2014). “REDES DE COMPUTADORES E A INTERNET”. 6ª Edição. 656p. Pearson. ISBN: 978-85-8143-677-7

LIMA, André Chastel. PROGRAMAÇÃO DISTRIBUÍDA - Arquitetura. 2025. Disponível em: <<https://www.comp.uems.br/~chastel/pd/aula5.pdf>>. Acesso em: 23/03/2025.

NEIVA, Sílvia Beatriz. “INTRODUÇÃO À COMUNICAÇÃO DE DADOS”, 2008. Disponível em: <<https://www.gta.ufrj.br/grad/silvia/silvia.htm/>>. Acesso em: 22 fev. 2025.

NEVES, Alex Fernandes; ORENSTEIN, Leonardo Poubel. “Wireless Application Protocol – WAP” (2007). Disponível em: <https://www.gta.ufrj.br/grad/07_1/wap/>. Acesso em: 04 de março de 2025.

PBVIRTUAL <https://www.pbvirtual.com.br/MLB-2189143388-sensor-proximidade-fotoeletrico-optico-0-100cm-difuso-pnp-_JM?srsId=AfmBOor4UpFtyHXzCjg1-zSGvCB8tal43mVPG4MZsVJe9GwFi2QCAzg_>. Acesso em 21/02/2025 às 23:13.

PEREIRA, Paulo Ricardo Cavalcanti Coelho (2024). Identificação e análise das habilidades necessárias para implementar a Indústria 4.0. – Angicos, 2024. p.64 Monografia – Universidade Federal Rural do Semi-Árido. Departamento de Engenharia de Produção. Disponível em: <<https://repositorio.ufersa.edu.br/server/api/core/bitstreams/fl2ade3b-6073-4482-85ba-6ce962613de7/content>>. Acesso em 16/03/2025 às 22:15

PORTAL VIDA DE SILICIO. COMUNICAÇÃO WIRELESS UDP COM ESP. 2025. Disponível em: <<https://portal.vidadesilicio.com.br/comunicacao-wireless-esp-protocolo-udp>>. Acesso em: 23/03/2025.

RODRIGUES, Matheus Ribeiro (2011). Estudo teórico e experimental de uma transmissão continuamente variável para veículo Baja Sae / M.R. Rodrigues. – São Paulo, 2011. p.81 Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecânica. Disponível em: <<https://bdta.abcd.usp.br/directbitstream/3eca7b8e-92a5-4683-a7b9->

3e897d9148f0/MATHEUS%20RIBEIRO%20RODRIGUES...>. Acesso em 21/02/2025 às 22:15

Rose, K., Elridge, S. and Chapin, L. (2015) The Internet of Things: An Overview: Understanding the Issues and Challenges of a More Connected World. Disponível em: <https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf>> acesso em 21/12/2024 às 09:55

SANTOS, André. TCP/UDP COMUNICAÇÃO DE DADOS (2015). Disponível em: < <https://slideplayer.com.br/slide/3382646/>>. Acesso em: 04 de março de 2025.

TANENBAUM, A. S.; WETHERALL, D. (2011). “Redes de Computadores”. 5ª ed. Pearson. ISBN 978-85-7605-924-0.

YASSER, Ismail (2019). Capítulo introdutório: Importância da Internet das coisas (IoT) e seus aplicativos, Internet das coisas (IoT) para aplicativos automatizados e inteligentes, YASSER, Ismail, Intech Open, DOI: 10.5772 /intechopen. 90022.

APÊNDICE A

Código 1: Código da Unidade Central (Arduino servidor) completo.

```
#include <SPI.h>

#include <WiFiNINA.h>

#include <WiFiUdp.h>

#include <RTCZero.h>


#define MAX_MENSAGENS 2

String listaMensagens[MAX_MENSAGENS];

int i_mensagens = 0;

String horario_data;

String inicio_contagem;


// Configurações WiFi

char ssid[] = "SSID";    // Wifi SSID

char pass[] = "SENHA";    // Wifi password


// Defina o IP estático, gateway, máscara de sub-rede e DNS para este dispositivo

IPAddress ip(192, 168, 0, 106);    // IP desejado para o Arduino

IPAddress gateway(192, 168, 0, 1);    // Geralmente o IP do roteador

IPAddress subnet(255, 255, 255, 0);    // Máscara de sub-rede

IPAddress dns(192, 168, 0, 1);    // Servidor DNS (pode ser o do roteador)


// Variáveis de tempo

unsigned long startTime = 0;

unsigned long elapsedTimes[2];
```

```

int mensagensRecebidas = 0;

String hora_dececao[2];

bool medicaoAtiva = false;


// Configurações de hardware

const int botaoPin = 2;      // Botão no pino D2 (pull-down)

const int ledPin = LED_BUILTIN;

const int GMT = -3;

bool NTP_suceded = false;


// Variáveis de cálculo

float velocidade_0_30 = 0.0;

float aceleracao_0_30 = 0.0;

float velocidade_30_100 = 0.0;

float aceleracao_30_100 = 0.0;


// Comunicação

WiFiUDP Udp;

unsigned int localPort = 2390;

RTCZero rtc;

int status = WL_IDLE_STATUS;

unsigned int serverPort = 2390;


WiFiServer server(80); // Servidor HTTP


void calcularMetricas() {

    if (mensagensRecebidas >= 1) {

```

```

float tempo_0_30 = elapsedTimes[0] / 1000.0;

velocidade_0_30 = 30.0 / tempo_0_30; // Velocidade média (m/s)

aceleracao_0_30 = (velocidade_0_30) / tempo_0_30; // Aceleração média (m/s²)
}

```

```

if (mensagensRecebidas >= 2) {

    float tempo_30_100 = (elapsedTimes[1] - elapsedTimes[0]) / 1000.0;

    velocidade_30_100 = 70.0 / tempo_30_100;

    aceleracao_30_100 = (velocidade_30_100) / tempo_30_100;

}

}

```

```

void print2digits(int number) {

    if (number < 10) Serial.print("0");

    Serial.print(number);

}

```

```

void setup() {

    Serial.begin(115200);

    pinMode(ledPin, OUTPUT);

    pinMode(botaoPin, INPUT); // Botão com pull-down externo

```

```

WiFi.config(ip, dns, gateway, subnet);

```

```

// Conecta ao WiFi

```

```

while (status != WL_CONNECTED) {

    Serial.print("Conectando a ");

```

```

Serial.println(ssid);

status = WiFi.begin(ssid, pass);

delay(10000);

}

Serial.println("-----");

Serial.println("Conectado ao Wi-Fi");

// Exibe o endereço IP atribuído ao Arduino

IPAddress ip = WiFi.localIP();

Serial.print("Endereço IP do servidor: ");

Serial.println(ip);

// Print the received signal strength

long rssi = WiFi.RSSI();

Serial.print("signal strength (RSSI):");

Serial.print(rssi);

Serial.println(" dBm");


// Configura RTC

rtc.begin();

unsigned long epoch;

int tentativas = 0;

do {

    epoch = WiFi.getTime();

    tentativas++;

} while (epoch == 0 && tentativas < 6);

if (tentativas < 6) {

```

```

rtc.setEpoch(epoch);

Serial.print("Relógio sincronizado: ");

NTP_suceded = true;
} else {

  Serial.print("Falha na sincronização do relógio (NTP)...");

  //while (1);

}

Udp.begin(localPort);

server.begin(); // Inicia o servidor HTTP

horario_data = (NTP_suceded ? formatarDataHora() : "" ); // Cria a string combinada

// Serial.print("Data/hora sincronizadas: ");

Serial.println(horario_data);

}

void loop() {

  // Verifica pressionamento do botão

  if (digitalRead(botaoPin)) {

    if (!medicaoAtiva) {

      iniciarMedicao();

    }

    delay(300); // Debounce simples

  }

  // Processa mensagens apenas se a medição estiver ativa

  if (medicaoAtiva) {

    int packetSize = Udp.parsePacket();

```

```

if (packetSize) {
    processarMensagem();
    calcularMetricas();
    if (mensagensRecebidas >= 2) {
        finalizarMedicao();
    }
}
}

// Servidor HTTP para requisições
WiFiClient client = server.available();

if (client) {
    String request = client.readStringUntil('\r');
    client.flush();

    // Determinar o formato da resposta
    bool useHTML = request.indexOf("format=html") != -1;
    bool forceJSON = request.indexOf("format=json") != -1;

    // Headers comuns
    String response = "HTTP/1.1 200 OK\r\n";
    response += "Connection: close\r\n";

    if (useHTML && !forceJSON) {
        // =====

        // Versão HTML

```

```
// =====

response += "Content-Type: text/html; charset=utf-8\r\n\r\n";

response += "<!DOCTYPE html><html><head>";

response += "<meta charset='UTF-8'>";

response += "<meta http-equiv='refresh' content='3'>";

response += "<title>Monitoramento de Sensores</title>";

response += "<style>";

response += "body {font-family: Arial, sans-serif; padding: 20px; background: #f0f0f0;}";

response += ".container {background: white; padding: 25px; border-radius: 15px; box-shadow: 0 0 15px rgba(0,0,0,0.1); max-width: 600px; margin: 0 auto;}";

response += ".sensor-container {display: grid; gap: 20px; margin-top: 25px;}";

response += ".sensor-card {padding: 20px; border-radius: 10px; border-left: 5px solid; background: #f8f9fa;}";

response += ".sensor-30m {border-color: #ffff00;}";

response += ".sensor-100m {border-color: #ff0000;}";

response += ".inicio-0m {border-color: #2ecc71;}";

response += ".sensor-header {display: flex; align-items: center; gap: 10px; margin-bottom: 15px;}";

response += ".sensor-icon {font-size: 24px;}";

response += "h1 {color: #2c3e50; margin-top: 0;}";

response += "h2 {color: #34495e; margin: 0 0 15px 0;}";

response += "p {margin: 8px 0; font-size: 16px;}";

response += ".dados-sensor strong {color: #2c3e50;}";

response += ".pendente {color: #95a5a6; font-style: italic;}";

response += ".footer-links {margin-top: 25px; padding-top: 15px; border-top: 1px solid #ecf0f1; text-align: center;}";

response += "a {color: #3498db; text-decoration: none; margin: 0 10px;}";
```

```

response += "a:hover {text-decoration: underline;}";

response += "</style>";

response += "</head><body><div class='container'>";

// Cabeçalho

response += "<div class='header'>";

response += "<h1><img alt='car' data-bbox='235 275 255 290' /> Monitoramento do percurso:</h1>";

response += "</div>";


response += "<div class='sensor-container'>";

//response += "<div class='inicio-contagem'>";

response += "<div class='sensor-card inicio-0m'>";

response += "<div class='sensor-header'>";

response += "<h2><img alt='clock' data-bbox='330 490 350 505' /> Início da Contagem: 0 Metros</h2>";

response += "</div>";

if (medicaoAtiva || (mensagensRecebidas == 2)){

    response += "<div class='dados-sensor'>";

    response += "<p><img alt='clock' data-bbox='415 615 435 630' /> " + inicio_contagem + "</p>";

    response += "</div>";

} else {

    response += "<p class='pendente'><img alt='hourglass' data-bbox='465 705 485 720' /> Aguardando início do percurso...</p>";

}

response += "</div>";

response += "</div>";

// Container dos Sensores

```

```

// Sensor 30m

response += "<div class='sensor-card sensor-30m'>";

response += "<div class='sensor-header'>";

response += "<span class='sensor-icon'><img alt='30m sensor icon' data-bbox='498 181 521 196'></span>";

response += "<h2>1° Sensor: 30 Metros</h2>";

response += "</div>";

if(mensagensRecebidas >= 1) {

    response += "<div class='dados-sensor'>";

    response += "<p><strong><img alt='clock icon' data-bbox='414 336 437 351'> Tempo Decorrido:</strong> " +
String(elapsedTimes[0]/1000.0, 3) + " segundos</p>";

    response += "<p><strong><img alt='clock icon' data-bbox='414 386 437 401'> Horário da Detecção:</strong> " +
hora_deteccao[0] + "</p>";

    response += "<p><img alt='speedometer icon' data-bbox='341 439 364 454'> Velocidade Média: " + String(velocidade_0_30, 2) + "
m/s</p>";

    response += "<p><img alt='accelerometer icon' data-bbox='336 491 361 506'> Aceleração Média: " + String(acceleracao_0_30, 2) + "
m/s²</p>";

    response += "</div>";

} else {

    response += "<p class='pendente'><img alt='hourglass icon' data-bbox='476 604 499 619'> Aguardando detecção...</p>";

}

response += "</div>"; // Fecha sensor-card

// Sensor 100m

response += "<div class='sensor-card sensor-100m'>";

response += "<div class='sensor-header'>";

response += "<span class='sensor-icon'><img alt='100m sensor icon' data-bbox='508 821 531 836'></span>";

response += "<h2>2° Sensor: 100 Metros</h2>";

response += "</div>";

```

```

if(mensagensRecebidas >= 2) {

    response += "<div class='dados-sensor'>";

    response += "<p><strong>⌚ Tempo Decorrido:</strong> " +
String(elapsedTimes[1]/1000.0, 3) + " segundos</p>";

    response += "<p><strong>🕒 Horário da Detecção:</strong> " +
hora_deteccao[1] + "</p>";

    response += "<p>🚗 Velocidade Média: " + String(velocidade_30_100, 2) + "
m/s</p>";

    response += "<p>📊 Aceleração Média: " + String(acceleracao_30_100, 2) + "
m/s²</p>";

    response += "</div>";

} else {

    response += "<p class='pendente'>⌚ Aguardando detecção...</p>";

}

response += "</div>"; // Fecha sensor-card

//Container metricas

response += "<div class='sensor-card total-card'>";

response += "<h2>📊 Métricas Gerais</h2>";

if(mensagensRecebidas >= 2) {

    // Container para Métricas Gerais

    float intervalo = (elapsedTimes[1] - elapsedTimes[0])/1000.0;

    float velocidadeMediaTotal = 100.0 / (elapsedTimes[1]/1000.0);

    float aceleracaoMediaTotal = (velocidade_30_100 - velocidade_0_30) / intervalo;

    response += "<p class='destaque'>🚗 Velocidade Média Total: " +
String(velocidadeMediaTotal, 2) + " m/s</p>";

    response += "<p class='destaque'>📊 Aceleração Média Total: " +
String(acceleracaoMediaTotal, 2) + " m/s²</p>";

    response += "<p>⌚ Intervalo entre Sensores: " + String(intervalo, 3) + " s</p>";

```

```

        response += "</div>";
    } else {
        response += "<p class='pendente'>⌚ Aguardando detecção...</p>";
    }

    response += "</div>"; // Fecha sensor-card

    response += "</div>"; // Fecha sensor-container

    // Links de Formato
    response += "<div class='footer-links'>";

    // response += "<a href='/?format=html'>HTML</a>";
    response += "<a href='/?format=json'>JSON</a>";
    response += "</div>";

    response += "</div></body></html>";
} else {
    // =====
    // Versão JSON (padrão)
    // =====

    response += "Content-Type: application/json\r\n\r\n";

    response += "{";
    response += "\"status\": " + String(medicacaoAtiva ? "\"ativo\"" : "\"inativo\"") + ",";
    response += "\"inicio_contagem\": \"" + inicio_contagem + "\",";
    response += "\"sensores\": {";

    // Sensor 30m
    response += "\"sensor_30m\": {";

```

```

        response += "\"tempo\": " + (mensagensRecebidas >= 1 ?
String(elapsedTimes[0]/1000.0, 3) : "null") + ",";

        response += "\"detecao\": \"\" + hora_deteccao[0] + "\",";

        response += "\"velocidade_media\": " + (mensagensRecebidas >= 1 ?
String(velocidade_0_30, 2) : "null") + ",";

        response += "\"aceleracao_media\": " + (mensagensRecebidas >= 1 ?
String(aceleracao_0_30, 2) : "null");

        response += "},";

        // Sensor 100m

        response += "\"sensor_100m\": {";

        response += "\"tempo\": " + (mensagensRecebidas >= 2 ?
String(elapsedTimes[1]/1000.0, 3) : "null") + ",";

        response += "\"detecao\": \"\" + hora_deteccao[1] + "\",";

        response += "\"velocidade_media\": " + (mensagensRecebidas >= 1 ?
String(velocidade_30_100, 2) : "null") + ",";

        response += "\"aceleracao_media\": " + (mensagensRecebidas >= 1 ?
String(aceleracao_30_100, 2) : "null");

        response += "}";

        response += "},";

        response += "\"geral\": {";

        response += "\"velocidade_media_total\": " + (mensagensRecebidas >= 2 ?
String(100.0 / (elapsedTimes[1]/1000.0), 2) : "null") + ",";

        response += "\"aceleracao_media_total\": " + (mensagensRecebidas >= 2 ?
String(((100.0 / (elapsedTimes[1]/1000.0))/(elapsedTimes[1]/1000.0)), 2) : "null");

        response += "}";

        response += "}";

    }

    client.print(response);

    client.stop();

```

```

    }
}

```

```

String formatarDataHora() {
    char buffer[34];
    snprintf(buffer, sizeof(buffer), "(%02d:%02d:%02d) - %02d/%02d/%04d",
        rtc.getHours() + GMT, rtc.getMinutes(), rtc.getSeconds(),
        rtc.getDay(), rtc.getMonth(), 2000 + rtc.getYear());
    return String(buffer);
}

void iniciarMedicao() {
    startTime = millis();
    mensagensRecebidas = 0;
    i_mensagens = 0;
    medicaoAtiva = true;

    inicio_contagem = (NTP_suceded ? formatarDataHora() : "Sem informação de
    horário/data") ;

    digitalWrite(ledPin, HIGH);

    Serial.println("\n--- Medição Iniciada ---");
    Serial.print("Início do percurso! | ");
    Serial.print("(");
    Serial.print(inicio_contagem);
    Serial.println(")");
    Serial.println("Aguardando sensores...");
    delay(300);
}

```

```

digitalWrite(ledPin, LOW);
}

void processarMensagem() {
    char packetBuffer[255];
    int len = Udp.read(packetBuffer, 255);

    if (len > 0) {
        packetBuffer[len] = 0;

        // Armazena tempo decorrido
        elapsedTimes[mensagensRecebidas] = millis() - startTime;

        hora_deteccao[mensagensRecebidas] = (NTP_suceded ? formatarDataHora() : "Sem
informação de horário/data");
        mensagensRecebidas++;

        // Exibe informações
        Serial.print("Sensor ");
        Serial.print(mensagensRecebidas);
        Serial.print(" detectado: ");
        Serial.print(packetBuffer);
        Serial.print(" | Tempo: ");
        Serial.print(elapsedTimes[mensagensRecebidas-1] / 1000.0, 3);
        Serial.print("s");
        Serial.print(" (");
        Serial.print(hora_deteccao[mensagensRecebidas-1]);
        Serial.println(") ");
    }
}

```

```

// Armazena mensagem

if (i_mensagens < MAX_MENSAGENS) {
    listaMensagens[i_mensagens] = String(packetBuffer);
    i_mensagens++;
}

digitalWrite(ledPin, HIGH);
delay(150);
digitalWrite(ledPin, LOW);
}
}

void finalizarMedicao() {
    Serial.println("\nResultados finais:");
    Serial.print("Tempo até primeiro sensor: ");
    Serial.print(elapsedTimes[0] / 1000.0, 3);
    Serial.println("s");

    Serial.print("Tempo até segundo sensor: ");
    Serial.print(elapsedTimes[1] / 1000.0, 3);
    Serial.println("s");

    Serial.print("Intervalo entre sensores: ");
    Serial.print((elapsedTimes[1] - elapsedTimes[0]) / 1000.0, 3);
    Serial.println("s");

    Serial.println("-----");
}

```

```
medicaoAtiva = false;
```

```
}
```

APÊNDICE B

Código 2: Código do Arduino cliente (30 metros).

```
#include <WiFiNINA.h>

// Configurações de rede
char ssid[] = "SSID";    // Nome da sua rede Wi-Fi
char pass[] = "SENHA";   // Senha da sua rede Wi-Fi
int status = WL_IDLE_STATUS;    //variável indicadora da conexão
int contador = 0;

// Configurações de UDP
WiFiUDP Udp;              //configura o protocolo UDP
unsigned int localPort = 2390;    // Porta local para o cliente
IPAddress serverIP(192, 168, 0, 106); // Endereço IP do servidor
unsigned int serverPort = 2390;   // Porta do servidor

// Pino do botão
const int buttonPin = 2;          //botão push-button ligado na porta digital 2 do arduino
nano 33 IoT

void setup() {
    // Inicializa a serial
    Serial.begin(9600);

    // Configura o pino do botão como entrada com pull-up interno
    pinMode(buttonPin, INPUT);

    // Conecta ao Wi-Fi
    while (status != WL_CONNECTED) {
        Serial.print("Conectando a ");
        Serial.println(ssid);
```

```
    status = WiFi.begin(ssid, pass);  
    delay(1000);  
}  
Serial.println("Conectado ao Wi-Fi");  
  
// Inicializa o UDP  
Udp.begin(localPort);  
}  
  
void loop() {  
    // Verifica se o botão foi pressionado  
    if (digitalRead(buttonPin) == HIGH) {  
        contador++;  
        String mensagem = "30m";  
        //+ String(contador) + "ª vez. ";  
        // Envia uma mensagem UDP para o servidor  
        Udp.beginPacket(serverIP, serverPort);  
        Udp.write(mensagem.c_str());  
        Udp.endPacket();  
        Serial.print("Mensagem enviada: ");  
        Serial.println(mensagem);  
  
        // Debounce do botão  
        delay(300);  
    }  
}
```

APÊNDICE C

Código 3: Código do Arduino cliente (100 metros).

```
#include <WiFiNINA.h>

// Configurações de rede
char ssid[] = "SSID";    // Nome da sua rede Wi-Fi
char pass[] = "SENHA";   // Senha da sua rede Wi-Fi
int status = WL_IDLE_STATUS;    //variável indicadora da conexão
int contador = 0;

// Configurações de UDP
WiFiUDP Udp;              //configura o protocolo UDP
unsigned int localPort = 2390;    // Porta local para o cliente
IPAddress serverIP(192, 168, 0, 106); // Endereço IP do servidor
unsigned int serverPort = 2390;   // Porta do servidor

// Pino do botão
const int buttonPin = 2;          //botão push-button ligado na porta digital 2 do arduino
nano 33 IoT

void setup() {
  // Inicializa a serial
  Serial.begin(9600);

  // Configura o pino do botão como entrada com pull-up interno
  pinMode(buttonPin, INPUT);

  // Conecta ao Wi-Fi
  while (status != WL_CONNECTED) {
    Serial.print("Conectando a ");
    Serial.println(ssid);
```

```
    status = WiFi.begin(ssid, pass);  
    delay(1000);  
}  
Serial.println("Conectado ao Wi-Fi");  
  
// Inicializa o UDP  
Udp.begin(localPort);  
}  
  
void loop() {  
    // Verifica se o botão foi pressionado  
    if (digitalRead(buttonPin) == HIGH) {  
        contador++;  
        String mensagem = "100m";  
        //+ String(contador) + "ª vez. ";  
        // Envia uma mensagem UDP para o servidor  
        Udp.beginPacket(serverIP, serverPort);  
        Udp.write(mensagem.c_str());  
        Udp.endPacket();  
        Serial.print("Mensagem enviada: ");  
        Serial.println(mensagem);  
  
        // Debounce do botão  
        delay(300);  
    }  
}
```