UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA E TECNOLOGIA

MANOEL ANTONIO DE SOUZA NETO

RASTREAMENTO DE DRONES POR GEOLOCALIZAÇÃO UTILIZANDO APLICAÇÃO IOT COM MICROCONTROLADOR ESP32

CARAÚBAS-RN
2023

MANOEL ANTONIO DE SOUZA NETO

RASTREAMENTO DE DRONES POR GEOLOCALIZAÇÃO UTILIZANDO APLICAÇÃO IOT COM MICROCONTROLADOR ESP32

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência e Tecnologia.

Orientador: Antônio Alisson Alencar Freitas, Prof. Me.

Coorientador: Francisco de Assis Brito Filho, Prof. Dr.

CARAÚBAS-RN

2023

N469r Neto, Manoel Antonio de Souza.
Rastreamento de drones por geolocalização utilizando aplicação IoT com microcontrolador ESP32 / Manoel Antonio de Souza Neto. - 2023.
84 f. : il.

Orientador: Antonio Alisson Alencar Freitas.
Coorientador: Francisco de Assis Brito Filho.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia Elétrica, 2023.

1. Internet das coisas. 2. IoT. 3. Sistema de rastreamento. 4. ESP32. 5. Geolocalização. I. Freitas, Antonio Alisson Alencar, orient. II. Filho, Francisco de Assis Brito, co-orient. III. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Caraúbas
Bibliotecária: Dalvanira Brito Rodrigues
CRB: 15/700

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.

MANOEL ANTONIO DE SOUZA NETO

RASTREAMENTO DE DRONES POR GEOLOCALIZAÇÃO UTILIZANDO APLICAÇÃO IOT COM MICROCONTROLADOR ESP32

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência e Tecnologia.

Defendida em: 19 / 05 / 2023.

BANCA EXAMINADORA

[redacted]

Antonio Alisson Alencar Freitas, Prof. Me. Presidente.

[redacted]

Francisco de Assis Brito Filho, Prof. Dr. Membro Examinador.

gov.br Documento assinado digitalmente
JOSE AILTON LEAO BARBOZA JUNIOR
Data: 20/05/2023 20:02:06-0300
Verifique em https://validar.iti.gov.br

José Ailton Leão Barboza Junior, Prof. Me. Membro Examinador.

[redacted]

Hugo Michel Câmara De Azevedo Maia, Prof. Dr. Membro Examinador.

AGRADECIMENTOS

Agradeço primeiramente à Deus, por me conceder a oportunidade de estar aqui hoje e por me fazer persistir e acreditar, mesmo a caminhada sendo longa e árdua.

Agradeço à toda minha família, que sempre esteve comigo em todos os momentos e são os maiores e melhores motivos que me mantem firme em busca de alcançar meus objetivos, em especial à minha mãe, Fatima Sousa, aos meus irmãos, Michael Maia e Sara Vitória, e a minha namorada, Julia Nicole, eu amo muito vocês.

Agradeço ao meu orientador e coorientador Antônio Alisson Alencar Freitas e Francisco de Assis Brito Filho, por toda paciência, dedicação e por todo conhecimento repassado. Embora tenha sido um tempo curto, foi o suficiente para conhecer os excelentes e dedicados profissional que são.

Agradeço à Banca Examinadora, Francisco de Assis Brito Filho e Rafael Luz Espindola, pela disponibilidade em avaliar meu trabalho. Profissionais que eu tive o prazer de conhecer.

Agradeço também a todos os meus amigos, da UFERSA, que tive o prazer de conhecer/manter durante esses anos. Em especial aos meus amigos Daniel Santos, Anderson Picasso, Ellyson Jacson, Matheus Thomas, Yam Pablo e aos demais.

A todos deixo meu muito obrigado.

O mais importante é a mudança, o movimento, o dinamismo, a energia. Só o que está morto não muda.

Clarice Lispector

RESUMO

As tecnologias de rastreamento e *internet* das coisas (*Internet of Things* - IoT) são abordagens cada vez mais importantes para o gerenciamento eficiente de cadeias de suprimentos, permitindo que as empresas reduzam custos, aumentem a produtividade e ofereçam serviços de melhor qualidade aos clientes. Nesse contexto, o objetivo deste trabalho é desenvolver um sistema de rastreamento de drones baseado em geolocalização, utilizando o conceito de IoT aplicado ao transporte de produtos. Além disso, o sistema inclui uma verificação automatizada de localização, que identifica se algum rastreador está próximo do destino. O intuito principal é a realização de um sistema de rastreamento de drones aplicado para transporte de produtos, além de fornecer dados relevantes para a implementação de projetos semelhantes e a partir disso, contribuir para a evolução do uso de tecnologias de rastreamento e IoT em processos de logística e transporte, visando à melhoria da eficiência, redução de custos e aprimoramento da experiência do cliente. Para o desenvolvimento da aplicação, foram utilizadas linguagens de programação como *C++*, *JavaScript* e *React*, além de dispositivos de alta relação custo-benefício, como o ESP32. Os resultados dos testes foram analisados e mostram casos de uso do sistema proposto.

Palavras-chave: *Internet* das coisas. IoT. Sistema de rastreamento. ESP32. Geolocalização.

ABSTRACT

Tracking technologies and the Internet of Things (IoT) are increasingly important for efficient supply chain management, allowing companies to reduce costs, increase productivity, and provide better quality services to customers. In this context, the objective of this study is to develop a geolocation-based drone tracking system using IoT concepts applied to product transportation. Furthermore, the system includes an automated location verification that identifies if any tracker is in proximity to the destination. The main aim is to create a drone tracking system for product transportation while providing relevant data for the implementation of similar projects, thus contributing to the advancement of tracking technologies and IoT utilization in logistics and transportation processes, with the goal of improving efficiency, reducing costs, and enhancing the customer experience. The development of the application involved programming languages such as C++, JavaScript, and React, as well as cost-effective devices like the ESP32. Test results were analyzed, showcasing use cases of the proposed system.

Keywords: Internet of Things. IoT. Tracking system. ESP32. Geolocation.

LISTA DE FIGURAS

LISTA DE QUADROS

Nenhuma entrada de sumário foi encontrada.

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

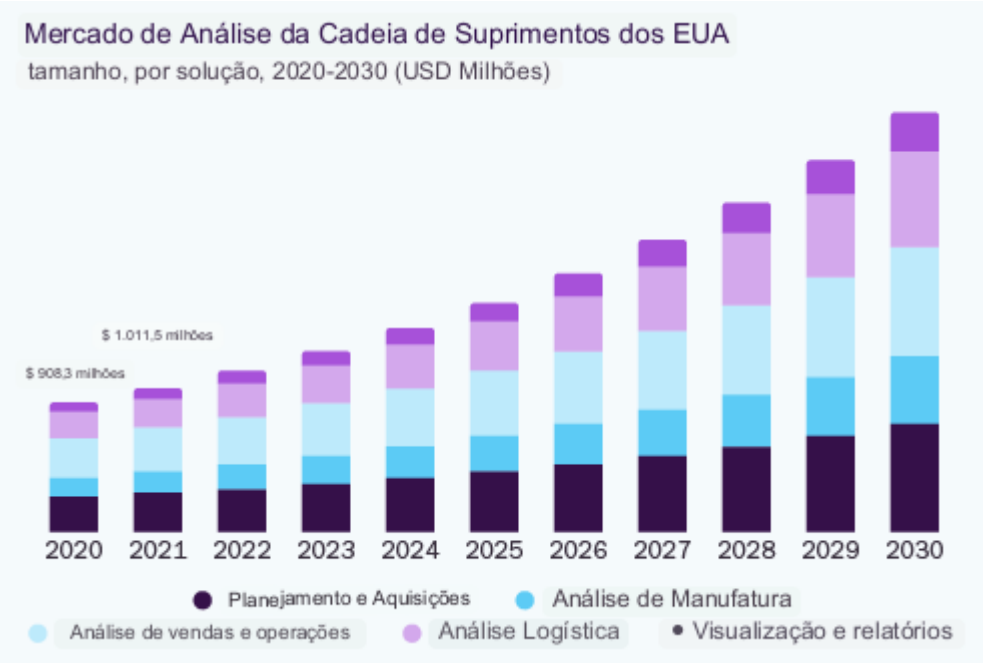
Com a sociedade em processo acelerado de desenvolvimento, as tecnologias de rastreamento utilizadas em gerenciamento de cadeias de suprimento têm sido objeto de diversas pesquisas, visando ao desenvolvimento e à otimização desses processos. As cadeias de suprimento correspondem ao conjunto de atividades e entidades envolvidas na produção e entrega de produtos ou serviços aos clientes, incluindo desde a aquisição de matérias-primas até o pós-venda. Embora possam ser considerados termos intercambiáveis, logística e cadeia de suprimentos (*Supply Chain*) têm definições distintas, segundo Bertaglia:

> "A cadeia de abastecimento corresponde ao conjunto de processos requeridos para obter materiais, agregar-lhes valor de acordo com a concepção dos clientes e consumidores e disponibilizar os produtos para o lugar (onde) e para a data (quando) que os clientes e consumidores os desejarem" (BERTAGLIA, 2009, p. 5).

Diante disso, o gerenciamento da cadeia de suprimentos se torna crucial, uma vez que esse processo possui grande importância estratégica, abrangendo desde a previsão da demanda e seleção dos fornecedores até a gestão do fluxo de materiais, contratos, análise de informações financeiras e movimentações, bem como a criação de novas instalações, tais como fábricas, armazéns e centros de distribuição. Além disso, essa cadeia tem como objetivo estabelecer e manter relacionamentos com clientes, bem como lidar com questões mais abrangentes, incluindo a economia, a sociedade e o meio ambiente.

Segundo o relatório da *Grand View Research* (2021), o mercado global de Gestão da Cadeia de Suprimentos foi avaliado em US$ 6,12 bilhões em 2022 e deve crescer a uma taxa de crescimento anual composta de 17,8% de 2023 a 2030, conforme retrata a Figura 1.

Figura 1 – Taxa de crescimento anual para Gestão de Cadeia de Suprimentos



Fonte: Adaptada de *Grand View Research*, 2022.

A demanda por análise da cadeia de suprimentos tem crescido devido à conscientização dos benefícios gerados por essa análise, como a precisão de previsão, otimização da cadeia de suprimentos, minimização de desperdício e síntese significativa de dados de negócios. Atualmente, diversas estratégias têm colaborado para maior eficiência nos processos logísticos e respostas mais rápidas às necessidades do cliente, como a terceirização, alianças estratégicas e operadores logísticos, entre outros. Para auxiliar no gerenciamento dessas estratégias, surgiram diversas tecnologias, como código de barras, intercâmbio eletrônico de dados (*Electronic Data Interchange* - EDI), sistema de gerenciamento de armazéns (*Warehouse Management System* - WMS), identificação via radiofrequência (*Radio Frequency Identification* - RFID) e rastreamento de frotas com tecnologia (*Global Positioning System* - GPS).

Cada modelo de tecnologia possui seus prós e contras, e dependendo da aplicação, podem ser combinados para promover uma melhor aplicabilidade no fornecimento das informações para auxiliar os profissionais de logística a gerenciarem suas operações ao longo da cadeia de suprimentos. A rastreabilidade também consiste em uma ferramenta importante para o gerenciamento das cadeias de suprimentos, oferecendo diversos benefícios como a redução de perdas e

desperdícios, melhorias da qualidade, segurança do produto e redução de tempo inativo, entre outros. Essa ferramenta diz respeito à capacidade de seguir o movimento de produtos e informações ao longo de uma cadeia de suprimento. No entanto, também apresenta alguns desafios, como a falta de interoperabilidade entre sistemas, a falta de padrões globais, alto custo e falta de confiança nos dados coletados.

De acordo com a pesquisa da Deloitte (2021), mais de 79% das empresas identificaram a gestão da cadeia de suprimentos como um fator importante para alcançar seus objetivos de sustentabilidade. Considerando que os estudos voltados para esses processos e ferramentas citados anteriormente têm tomado grandes proporções, torna-se relevante a realização de novas aplicações que visem auxiliar no crescimento da tecnologia e, se possível, otimizá-las.

Neste estudo, serão abordadas as principais bases de rastreamento por geolocalização, visando a utilização de *internet* das coisas (*Internet of Things* - IoT) para auxiliar na entrega de produtos com drones. Para ilustrar essa proposta, será apresentado um protótipo de projeto e, a partir disso, será desenvolvido um sistema de rastreamento de drones, utilizando IoT acoplada a diversas tecnologias acessíveis, com o intuito principal de fornecer conhecimento para a comunidade e impulsionar novos estudos sobre o tema.

Os principais objetivos que este trabalho visa promover com seu desenvolvimento serão apresentados no capítulo 2, enquanto o capítulo 3 abordará os principais embasamentos para tornar possível o desenvolvimento do projeto. A proposta do trabalho será detalhada no capítulo 4, e o capítulo 5 descreverá os procedimentos e tecnologias utilizados na execução do projeto.

Por fim, as discussões e resultados obtidos após a realização do projeto serão apresentados no capítulo 6. E, para finalizar, as conclusões finais serão apresentadas no capítulo 7.

2 OBJETIVOS

2.1 OBJETIVO GERAL

O objetivo do presente trabalho envolve a realização de um sistema de rastreamento por geolocalização de drones que é baseado no conceito *internet* das coisas aplicado comumente para transporte de produtos. Também visa promover o setor utilizando conceitos atuais, e ainda, nortear e caracterizar projetos desse âmbito para as pessoas que pensam em inovar utilizando essas tecnologias ou similares.

2.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um sistema de rastreamento visando o custo-benefício;
- Implementar funcionalidades de verificação automatizada de localização no sistema de rastreamento de drones, com o objetivo de identificar a proximidade dos rastreadores em relação ao destino;
- Coletar dados relevantes durante o processo de rastreamento de drones e utilizá-los para fornecer informações úteis na implementação de projetos semelhantes;
- Contribuir para a evolução do uso de tecnologias de rastreamento e Internet das Coisas (IoT) em processos de logística e transporte, visando à melhoria da eficiência, redução de custos e aprimoramento da experiência do cliente.

3 REFERENCIAL TEÓRICO

3.1 *Internet* das Coisas (*Internet of Things* - IoT)

A *Internet* das Coisas é um conceito que se refere à interconexão de objetos físicos, dispositivos, sensores e outros itens do dia a dia com a *internet*, permitindo que estes se comuniquem entre si e com outros sistemas conectados. Segundo Ashton (2009), a IoT pode ser definida como "a conexão de qualquer objeto com um identificador exclusivo e a capacidade de transmitir dados automaticamente pela *internet* sem necessidade de interação humano-humano ou humano-computador".

A IoT também pode ser vista como uma rede de objetos conectados que podem interagir uns com os outros para realizar tarefas úteis. Os objetos conectados podem ser dispositivos como smartphones, tablets, eletrodomésticos, veículos, sensores, entre outros, que possuem capacidade de comunicação e processamento de dados (Atzori et al., 2010).

Os benefícios da IoT incluem a coleta de dados em tempo real e a tomada de decisões baseadas em informações precisas e em tempo hábil. A IoT tem sido aplicada em diversos setores, incluindo saúde, transporte, agricultura, automação residencial, indústria e segurança, entre outros. A aplicação da IoT em diferentes setores tem o potencial de trazer melhorias significativas em termos de eficiência, segurança e qualidade de vida das pessoas.

3.1.1 Arquitetura de sistemas IoT

A arquitetura de sistemas IoT é composta por diversos elementos que trabalham em conjunto para permitir a conexão e a comunicação entre dispositivos, sensores e sistemas. Segundo Atzori et al. (2010), a arquitetura de sistemas IoT pode ser dividida em quatro camadas: percepção, rede, middleware e aplicação.

A camada de percepção é composta pelos dispositivos IoT e pelos sensores que são responsáveis pela coleta de dados e informações do ambiente. Segundo Gubbi et al. (2013), os dispositivos IoT possuem a capacidade de coletar informações a partir de sensores e enviar essas informações para a camada de rede.

A camada de rede é responsável pela comunicação entre os dispositivos IoT e a nuvem, permitindo a transmissão dos dados coletados pelos dispositivos. Segundo Atzori et al. (2010), essa camada é responsável por permitir a conectividade entre dispositivos de diferentes tecnologias e plataformas.

A camada de *middleware* é responsável por gerenciar a comunicação entre a camada de rede e a camada de aplicação, permitindo o processamento e a análise dos dados coletados pelos dispositivos IoT. Segundo Gubbi et al. (2013), o middleware é responsável por garantir a interoperabilidade entre diferentes plataformas e protocolos de comunicação.

A camada de aplicação é responsável por processar e analisar os dados coletados pelos dispositivos IoT e fornecer informações e serviços aos usuários finais. Segundo Atzori et al. (2010), essa camada é responsável por fornecer serviços como monitoramento remoto, automação residencial, gestão de ativos, entre outros.

Em resumo, a arquitetura de sistemas IoT é composta por dispositivos, sensores, redes de comunicação, middleware e aplicações, trabalhando em conjunto para permitir a coleta, a transmissão, o processamento e a análise de dados em tempo real.

3.1.2 Tecnologias de comunicação para IoT

As tecnologias de comunicação são fundamentais para a implementação de sistemas IoT, permitindo a comunicação entre dispositivos e a transferência de dados. Segundo Gubbi et al. (2013), as principais tecnologias de comunicação utilizadas em sistemas IoT são *Wi-Fi*, *Bluetooth*, *Zigbee*, RFID, NB-IoT, *LoRa*.

De acordo com Gubbi et al. (2013), a escolha da tecnologia de comunicação adequada para um sistema IoT depende das características do projeto, como distância de comunicação, taxa de transferência de dados, consumo de energia e custo.

3.1.3 Aplicações em diferentes setores

A IoT tem sido aplicada em diversos setores, incluindo saúde, transporte, agricultura, automação residencial, indústria e segurança, entre outros. A aplicação

da IoT em diferentes setores tem o potencial de trazer melhorias significativas em termos de eficiência, segurança e qualidade de vida das pessoas.

Na área da saúde, a IoT tem sido aplicada em dispositivos médicos para monitoramento remoto de pacientes, permitindo a detecção precoce de sintomas e o gerenciamento de doenças crônicas. De acordo com a pesquisa realizada por Al-Fuqaha et al. (2015), a IoT pode melhorar a qualidade do atendimento médico, reduzir custos e aumentar a eficiência dos serviços de saúde.

Na indústria, a IoT tem sido aplicada em sistemas de monitoramento de máquinas e equipamentos, permitindo a detecção precoce de falhas e a manutenção preventiva. Segundo a pesquisa realizada por Zhou et al. (2010), a IoT pode melhorar a eficiência da produção, reduzir custos e aumentar a segurança dos trabalhadores.

Na área de transportes, a IoT tem sido aplicada em sistemas de monitoramento de veículos e gestão de tráfego, permitindo a redução de congestionamentos e a melhoria da segurança nas estradas. Segundo a pesquisa realizada por Wang et al. (2015), a IoT pode melhorar a eficiência dos sistemas de transporte, reduzir custos e aumentar a segurança dos usuários.de comunicação, taxa de transferência de dados, consumo de energia e custo.

3.2 Microcontroladores

Os microcontroladores são circuitos integrados que incorporam um processador, memória e dispositivos de entrada/saída em um único *chip*. Eles são projetados para controlar dispositivos e sistemas em diferentes aplicações, desde dispositivos domésticos até equipamentos industriais complexos. Eles podem ser programados para realizar tarefas específicas e executam essas tarefas de forma autônoma, sem a necessidade de intervenção humana constante.

Segundo Mazidi et al. (2016), um microcontrolador é composto por quatro componentes principais: a unidade central de processamento (CPU), a memória, os dispositivos de entrada e os dispositivos de saída. A CPU é responsável por executar as instruções do programa e controlar o funcionamento do microcontrolador. A memória é utilizada para armazenar o programa e os dados necessários para a execução das tarefas. Os dispositivos de entrada são responsáveis por enviar

informações para o microcontrolador, enquanto os dispositivos de saída são utilizados para enviar informações para outros dispositivos ou sistemas.

De acordo com Raj Kamal (2014), os microcontroladores são dispositivos eletrônicos que possuem um processador, memória, periféricos e interfaces de entrada e saída em um único *chip*. Eles são utilizados em uma variedade de aplicações, desde automação industrial até dispositivos de consumo.

Entre as principais características dos microcontroladores, podemos destacar o baixo custo em comparação com outros sistemas de computação, o baixo consumo de energia, a integração de periféricos como portas de entrada/saída, conversores analógico-digitais, temporizadores, comunicação serial e USB, o que os torna altamente flexíveis e versáteis para diversas aplicações, e a possibilidade de programação, o que permite o desenvolvimento de *software* personalizado para controlar o comportamento dos dispositivos.

Liu et al. (2016) destacam que os microcontroladores são amplamente utilizados em aplicações de IoT devido à sua capacidade de integrar diferentes periféricos em um único chip, o que permite que sejam utilizados em uma ampla variedade de dispositivos e sistemas de IoT. Além disso, os microcontroladores são ideais para aplicações de IoT devido ao seu baixo consumo de energia, o que permite que sejam usados em dispositivos que requerem bateria ou energia limitada.

Assim, conclui-se que os microcontroladores são fundamentais para as aplicações de IoT devido à sua flexibilidade, versatilidade e baixo consumo de energia, permitindo que sejam utilizados em uma ampla variedade de dispositivos conectados à *internet*.

3.3 *Message Queuing Telemetry Transport* (MQTT)

O MQTT é um protocolo de comunicação de mensagens projetado para dispositivos com recursos limitados e conexões de rede com largura de banda reduzida. Ele foi criado por Andy Stanford-Clark da IBM e Arlen Nipper da Eurotech em 1999 (STANFORD-CLARK; NIPPER, 1999). Segundo Oliveira et al. (2018), o MQTT funciona de forma assíncrona, permitindo que os dispositivos conectados enviem e recebam mensagens em tempo real, sem a necessidade de conexões persistentes. Além disso, o MQTT utiliza o modelo de publicação e assinatura, onde

os dispositivos publicam mensagens em tópicos e outros dispositivos se inscrevem em tópicos específicos para receber essas mensagens. Isso permite uma comunicação eficiente e escalável entre dispositivos em uma rede IoT.

O MQTT é amplamente utilizado em aplicações de IoT devido à sua eficiência, confiabilidade e escalabilidade. Ele permite que dispositivos IoT se comuniquem de forma eficiente e confiável, mesmo em redes com largura de banda limitada e recursos limitados de *hardware*. Além disso, o MQTT é altamente escalável, permitindo que milhões de dispositivos IoT se comuniquem entre si em tempo real (OLIVEIRA et al., 2018).

3.3.1 Principais características do protocolo

O protocolo MQTT é simples, leve e eficiente em termos de largura de banda e consumo de energia (SINGH; SHARMA, 2020). Ele foi projetado para atender às necessidades de comunicação de dispositivos de IoT que operam em condições de rede instáveis e de baixa largura de banda. Segundo Hunkeler et al. (2008), as principais características desse protocolo incluem:

- Leveza: o MQTT é um protocolo de comunicação leve que requer pouca largura de banda e baixo consumo de energia, o que o torna ideal para dispositivos de IoT que possuem recursos limitados de hardware.
- Assinatura de tópicos: o MQTT utiliza o conceito de tópicos para permitir que os dispositivos se inscrevam e publiquem mensagens em um canal específico de comunicação. Isso permite uma comunicação eficiente e seletiva entre os dispositivos.
- Qualidade de serviço (QoS): o MQTT oferece diferentes níveis de QoS que permitem controlar a confiabilidade da entrega das mensagens entre os dispositivos conectados. O nível de QoS pode ser ajustado de acordo com as necessidades de cada aplicação.
- Segurança: o MQTT suporta mecanismos de autenticação e criptografia que garantem a segurança das comunicações entre os dispositivos conectados em uma rede.

A simplicidade, leveza e eficiência em termos de consumo de energia tornaram o MQTT um protocolo de comunicação popular em sistemas de IoT (HUNKELER et al., 2008).

3.3.2 Arquitetura do protocolo

A arquitetura do protocolo MQTT é baseada no modelo de comunicação *publish*/*subscribe* e é composta por três componentes principais, como descrito por Singh e Sharma (2020):

- *Broker*: é um servidor central que atua como intermediário entre os dispositivos que publicam mensagens e os que as recebem. O *Broker* é responsável por receber e armazenar as mensagens, bem como distribuí-las para os dispositivos que se inscrevem nos tópicos correspondentes. Ele também é responsável por garantir que as mensagens sejam entregues com sucesso aos dispositivos inscritos, de acordo com o nível de QoS especificado.
- *Publisher*: é um dispositivo que publica informações em um tópico específico. Ao publicar uma mensagem, o dispositivo envia-a para o *Broker*, que a armazena e a distribui para os dispositivos inscritos no tópico. O *Publisher* não precisa saber quais dispositivos estão inscritos no tópico, pois o *Broker* é responsável por gerenciar as assinaturas e garantir que a mensagem seja entregue a todos os dispositivos inscritos.
- *Subscriber*: é um dispositivo que se inscreve em um tópico específico para receber informações publicadas por outros dispositivos. Ao se inscrever em um tópico, o dispositivo envia uma solicitação ao *Broker*, que o adiciona à lista de dispositivos inscritos no tópico correspondente. Quando um *Publisher* publica uma mensagem no tópico, o *Broker* a distribui para todos os dispositivos inscritos.

Ainda de acordo com Hunkeler et al. (2008), a arquitetura do MQTT é flexível e escalável, permitindo que os dispositivos IoT se comuniquem de forma eficiente em redes com recursos limitados de hardware e largura de banda. O modelo *publish*/*subscribe* também permite uma comunicação seletiva e eficiente entre

dispositivos, sem a necessidade de conhecer o endereço IP ou nome do dispositivo de destino.

4 SISTEMA DE RASTREAMENTO PROPOSTO

Utilizando os conhecimentos citados como base, o trabalho foi primeiramente contextualizado e, em seguida, foram definidas as características, dispositivos e padrões que o projeto deveria conter para garantir uma melhor compreensão e fluidez em seu desenvolvimento. Apesar de possuir grande escalabilidade, o projeto foi simplificado para que fosse viável concluir todos os objetivos com qualidade e eficácia, mantendo apenas aspectos voltados para os objetivos gerais e específicos.

O projeto consiste basicamente em um sistema de rastreamento de geolocalização de produtos, utilizando drones ou caminhões como meios de transporte a serem rastreados. Optou-se pelos drones como principal meio de transporte para manter um aspecto mais futurista. Além disso, a automação utilizando o conceito de Internet das Coisas desempenha um papel importante no projeto, permitindo sinalizar e abrir compartimentos, portões ou portas para que o meio de transporte possa realizar a entrega quando entrar na área de destino.

Com essa abordagem, o acesso ao galpão de armazenamento é liberado automaticamente quando o drone chega ao seu destino, garantindo um processo ágil e eficaz. Outro ponto importante, consiste na possibilidade de os produtos em transporte e também a área do galpão serem observados por meio de uma interface.

Para a realização desta aplicação, é necessário utilizar o GNSS (*Global Navigation Satellite System*), um sistema global de navegação por satélite amplamente utilizado em diversas aplicações, como navegação terrestre, marítima e aérea, monitoramento de frotas, mapeamento e georreferenciamento. No âmbito deste projeto, o GPS (*Global Positioning System*), um dos sistemas GNSS mais conhecidos e amplamente utilizados, será adotado para receber as informações de localização em tempo real dos drones utilizados como meio de transporte.

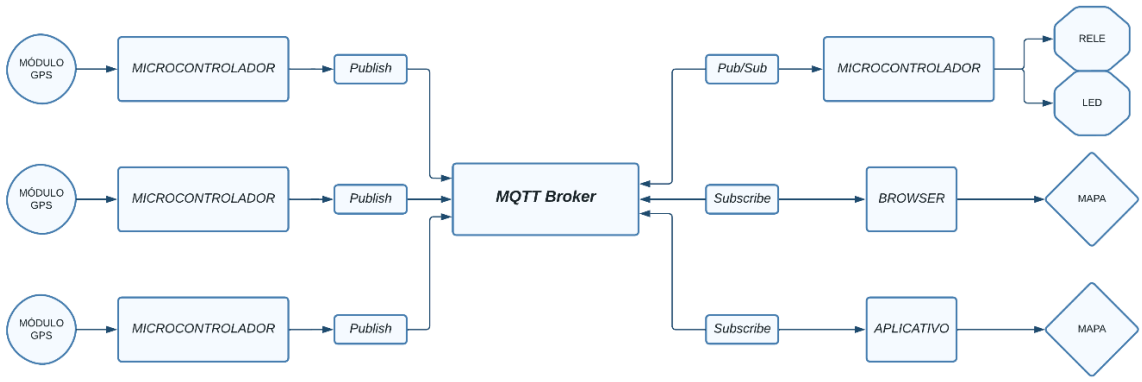
Essas informações de localização precisam ser tratadas e enviadas por meio de um microcontrolador, o qual, em conjunto com o módulo GPS, permitirá a obtenção, tratamento e envio dos dados por meio do conceito de IoT e do protocolo MQTT. O protocolo MQTT se destaca pela confiabilidade na transmissão de dados em redes instáveis e pela capacidade de transportar pacotes leves. Baseado no modelo *publish/subscribe* e operando sobre o protocolo TCP/IP, o MQTT requer uma conexão com a *internet*.

Para estabelecer a conexão com a internet, optou-se por configurar um *WiFi* específico, que permitirá a comunicação entre os dados a serem enviados e/ou recebidos e o *Broker*. Além disso, será necessário ter outro microcontrolador localizado no galpão, responsável por verificar e analisar todas as localizações dos drones enviadas pelo *Broker* MQTT. Quando um drone se aproximar, o microcontrolador no galpão deverá liberar o acesso para que o drone possa realizar a entrega do produto de forma automática.

Esse projeto apresenta muitas aplicações além da indústria. Pode ser particularmente útil quando é necessário que os itens sejam entregues rapidamente e sem obstáculos em locais de difícil acesso ou com um alto grau de urgência, como em caso de medicamentos como a insulina ou adrenalina.

A Figura 2 ilustra o diagrama de como o projeto deve funcionar e como as diferentes topologias são conectadas.

Figura 2 – Diagrama do projeto



Fonte: O autor.

Ao lado esquerdo do diagrama, é possível identificar um conjunto de módulos GPS juntamente com seus microcontroladores, esse conjunto para facilitar o entendimento, chamou-se de "*tracker*", e são responsáveis por obter a localização atual de seus drones e publicá-las (através do *publish*) no MQTT Broker.

Já no lado direto superior, chamou-se o conjunto microcontrolador com relé ou *LED* de "*checker*", pois sua função inicial seria de verificar se algum drone acoplado ao seu *tracker* estaria dentro de uma área especifica, e se estivesse, poderia acionar algum comando. Como buscou-se a simplificação do projeto, o *checker* é responsável

por apenas verificar se possui algum drone perto da posição especificada e se houver, acender um *LED*, caso não haja, o *LED* deve permanecer apagado.

Por fim, os dois últimos conjuntos são responsáveis por exibir as atuais posições dos *trackers*, bem como mostrar também o status do *checker*, ou seja, exibir se existe ou não algum drone na área especifica pelo *checker*.

Para a realização desse trabalho, apesar de poder ser testado com diversos dispositivos, devido ao tempo escasso e pouco recurso, optou-se por apenas um *tracker*, um *checker* e apenas uma das tecnologias de rastreamento, cuja escolhida foi o *Frontend*, que consiste na visualização do mapa através do *browser*, em outras palavras, um *website*.

5 MATERIAIS E MÉTODOS

Buscando o melhor desempenho e fluidez do sistema de rastreamento, alguns dispositivos, procedimentos e tecnologias foram escolhidos e utilizados pelo setor responsável de desenvolvimento, autor, orientador e coorientador do trabalho, detalhados a seguir:

5.1 Tecnologias e materiais utilizados

5.1.1 ESP32

O ESP32 é um microcontrolador de baixo custo desenvolvido pela Espressif Systems (ESPRESSIF SYSTEMS, 2023), que tem sido muito utilizado no desenvolvimento de projetos de IoT. Ele conta com uma arquitetura de dois núcleos e uma grande variedade de interfaces de comunicação, como *Wi-Fi*, *Bluetooth*, UART, SPI e I2C (ESPRESSIF SYSTEMS, 2023), o que o torna muito versátil para aplicações IoT.

O Doit ESP32 DevKit v1 é uma placa de desenvolvimento baseada no ESP32, que conta com uma série de recursos adicionais, como um regulador de tensão, um conector *micro-USB* para alimentação e programação, um botão de *reset* e um *LED* indicador de energia (DOIT.AM, 2023). Essa placa é muito utilizada em projetos de IoT, pois é fácil de usar e já conta com diversos periféricos integrados, como sensores e atuadores (DOIT.AM, 2023).

De acordo com Pérez (2018), "com o ESP32 e o Doit ESP32 DevKit v1, é possível desenvolver uma ampla variedade de projetos de IoT, como sistemas de monitoramento ambiental, dispositivos de controle remoto, sensores de presença e muitos outros". Kolban (2018) também afirma que essas placas permitem o desenvolvimento de projetos de IoT com uma grande variedade de aplicações.

Figura 3 – Placa de desenvolvimento DOIT

Fonte: DOIT.AM, 2023.

Por apresentar um *firmware* embarcado, o DOIT Esp32 DevKit v1 facilita a programação do chip uma vez que o programa pode ser carregado no chip por meio de um cabo *mini-USB*. Ainda, o microcontrolador pode ser programado utilizando a *Integrated Development Environment* (IDE) do Arduino, por meio da inclusão da biblioteca do ESP.

5.1.2 Módulo GPS NE0-6M

O módulo GPS NE0-6 da *u-blox* é um receptor GPS de baixo custo amplamente utilizado em projetos de eletrônica e robótica devido à sua antena embutida e fácil integração. O NEO-6 suporta sinais de GPS, GLONASS, Galileo e *BeiDou* para maior precisão na localização e fornece informações de latitude, longitude, altitude e velocidade (u-blox, 2023).

A fabricante, *u-blox* (2023), enfatiza que esses receptores combinam alta capacidade de integração e opções flexíveis de conectividade em um pacote pequeno, tornando-os ideais para produtos finais de mercado de massa com tamanho e custo restritos.

O módulo simplesmente verifica sua localização na Terra e fornece dados de saída que são a longitude e a latitude de sua posição. GY-NE06MV2 vem com uma pequena bateria para *hot-start*, juntamente com uma EEPROM integrada. (*DatasheetHub*, 2023)

A Tabela 1 expões as informações acerca do módulo escolhido:

Tabela 1 – Parâmetros do NE0-6M com antena externa.

Característica	**Descrição**
Fabricante	u-blox
Modelo	NE0-6M
Tamanho	16 mm x 12.2 mm x 2.4 mm
Peso	1.8 gramas
Conector	Conector IPEX
Protocolos suportados	NMEA, UBX
Sensibilidade	-162 dBm
Velocidade de atualização	Até 10 Hz
Precisão	Até 2.5 metros
Chipset u-blox	UBX-G6010-ST
Faixa de operação	-40 °C a +85 °C
Tensão de alimentação	3.3 V
Consumo de energia	20 mA
Ganho da antena	25 dB
Compatibilidade	Arduino, Raspberry Pi, ESP32

Fonte: O autor.

Já a Figura 4 abaixo ilustra um modelo de receptor GPS.

Figura 4 – Módulo GPS NE0-6M

Fonte: *DatasheetHub*, 2023.

5.1.3 *C++*

De acordo com TIOBE Index (2023), a linguagem de programação *C++* é frequentemente empregada em projetos de microcontroladores, como o ESP32, por possuir uma sintaxe semelhante à linguagem C e recursos avançados de programação orientada a objetos (POO). O uso do *C++* permite criar códigos mais organizados e modulares, simplificando o desenvolvimento de projetos. A ampla adoção dessa linguagem pela comunidade de desenvolvedores de *software* resultou em uma vasta oferta de bibliotecas, acelerando ainda mais o desenvolvimento de projetos.

No caso específico do ESP32, a linguagem *C++* é amplamente utilizada devido ao seu elevado desempenho e capacidade de processamento, e muitas das bibliotecas e exemplos disponíveis na *internet* para esse microcontrolador são escritos em *C++*. Isso torna mais fácil a vida dos desenvolvedores (TIOBE *Index*, 2023).

5.1.4 Arduino IDE

O Arduino IDE é um ambiente de desenvolvimento integrado gratuito e de código aberto utilizado para programar microcontroladores da família Arduino, incluindo o ESP32. Segundo Banzi e Shiloh (2014), o IDE oferece uma interface gráfica simples e intuitiva, permitindo que desenvolvedores sem muita experiência em programação possam programar os microcontroladores de forma fácil e rápida.

O ambiente de programação do Arduino IDE é baseado na linguagem *C++*, que é escrita em um editor de texto. É possível verificar a sintaxe do código com a ferramenta de verificação, que detecta erros de sintaxe antes da compilação. A compilação do código é feita diretamente no IDE, que gera o arquivo binário a ser gravado no microcontrolador.

Segundo Schwartsman (2015), o Arduino IDE possui diversas bibliotecas e exemplos disponíveis, permitindo que os desenvolvedores possam aproveitar o conhecimento da comunidade para acelerar o desenvolvimento de seus projetos. Além disso, o IDE permite que os desenvolvedores possam programar os microcontroladores por meio de uma conexão USB.

Assim, o Arduino IDE é uma ferramenta importante para a programação do ESP32, permitindo que desenvolvedores de diferentes níveis de habilidade possam programar esse microcontrolador de forma fácil e rápida.

5.1.5 JavaScript

O *JavaScript* (*JS*) é uma linguagem de programação utilizada para desenvolver interfaces interativas em páginas da *web*. Segundo Flanagan (2020), o *JavaScript* é uma das principais tecnologias utilizadas na construção de páginas da *web*, sendo responsável pela adição de interatividade, animações e efeitos visuais.

A utilização do *JS* no desenvolvimento de interfaces tem se tornado cada vez mais comum, uma vez que ele permite a criação de elementos interativos, como botões, menus e formulários, sem a necessidade de recarregar a página.

Além disso, o *JavaScript* é uma linguagem de fácil aprendizado e pode ser executada em praticamente qualquer navegador moderno. Por meio do uso de bibliotecas e *frameworks*, como *React* e *Vue.js*, é possível criar interfaces sofisticadas e altamente interativas com *JavaScript*. Essas ferramentas permitem a criação de componentes reutilizáveis e simplificam o processo de desenvolvimento.

5.1.6 Node.js

O *Node.js* consiste em uma plataforma de desenvolvimento de *software* criada em 2009 com o objetivo de permitir que os desenvolvedores criem aplicativos de rede escaláveis e em tempo real com o uso da linguagem *JavaScript* tanto no lado do servidor quanto no cliente. Devido à sua arquitetura baseada em eventos, ele é capaz de manipular uma grande quantidade de conexões simultâneas com eficiência e é amplamente utilizado para a construção de *API*s e aplicativos *web*.

No desenvolvimento de interfaces, o *Node.js* é frequentemente utilizado em conjunto com *frameworks* para criar aplicações *web* de alta performance. Ele permite a criação de servidores *web*, roteamento de URLs, gerenciamento de arquivos e outras funcionalidades importantes para o desenvolvimento de interfaces *web* interativas.

Segundo Dornelas e Vieira (2018), o *Node.js* é uma plataforma de desenvolvimento de *software* que tem ganhado cada vez mais popularidade no

mercado, principalmente por sua capacidade de desenvolver aplicações em tempo real, escaláveis e com alto desempenho. Já Santos e Pinto (2020) afirmam que o uso do *Node.js* em conjunto com *frameworks* como o *React* tem sido uma tendência no desenvolvimento de aplicações *web* modernas.

5.1.7 React

A biblioteca de *JavaScript ReactJS* foi criada pelo Facebook com o objetivo de desenvolver interfaces de usuário (UI) para aplicações *web* de maneira eficiente e escalável (FACEBOOK, 2023). De acordo com a W3TECHS (2023), o *ReactJS* é amplamente utilizado em projetos de desenvolvimento de interfaces para aplicações *web* e é uma escolha popular entre desenvolvedores front-end. Uma das principais vantagens do *ReactJS* é o seu uso do conceito de Virtual DOM, que permite atualizações de interface mais rápidas e eficientes em relação ao DOM tradicional.

O *ReactJS* permite a criação de componentes reutilizáveis que podem ser facilmente integrados em outras partes da aplicação, permitindo uma maior modularidade do código e tornando o processo de desenvolvimento mais ágil (BARTHOLOMEW; FOX; HOLDEN, 2018). Além disso, o *ReactJS* é altamente compatível com outras bibliotecas e *frameworks* de *JavaScript*, permitindo a sua integração em diversas arquiteturas de aplicações.

Grandes empresas, como *Netflix*, *Airbnb* e *Instagram*, utilizam o *ReactJS* em suas aplicações *web*, destacando sua eficiência e popularidade (W3TECHS, 2023). Em suma, o *ReactJS* é uma biblioteca amplamente utilizada no desenvolvimento de interfaces de usuário para aplicações *web*, oferecendo uma maneira eficiente, escalável e modular de desenvolver aplicações com UI (BARTHOLOMEW; FOX; HOLDEN, 2018).

5.1.8 TypeScript

O *TypeScript* (*TS*) é uma linguagem de programação *open source* desenvolvida pela Microsoft que adiciona recursos de tipagem estática ao *JavaScript* (MICROSOFT, 2023). De acordo com Prado (2019), o *TypeScript* permite que os desenvolvedores definam explicitamente os tipos de dados que uma variável pode conter, o que pode ajudar a detectar erros e aprimorar a legibilidade do código.

Além disso, o *TypeScript* também fornece outras funcionalidades, como a capacidade de escrever código mais seguro e confiável, bem como a possibilidade de utilizar recursos avançados de programação orientada a objetos (LOVINE, 2019). Essas características fazem do *TS* uma opção popular entre os desenvolvedores *frontend* que desejam criar interfaces de usuário mais complexas e escaláveis.

Uma das principais vantagens do *TypeScript* é a sua integração com o *ReactJS* (MICROSOFT, 2023), permitindo que os desenvolvedores criem interfaces mais robustas e seguras para aplicações *web*. Além disso, o *TypeScript* também é compatível com outras bibliotecas e *frameworks* populares de *JavaScript*, como *Angular* e *Vue.js*.

Devido à sua popularidade e benefícios, o *TypeScript* tem sido amplamente adotado em projetos de desenvolvimento de interfaces para aplicações *web*, tornando-se uma opção popular entre desenvolvedores front-end. Algumas empresas de renome, como Slack, Asana e Airbnb, utilizam o *TypeScript* em suas aplicações *web* (KREBS; FABRY, 2021).

5.1.9 *JSON*

JSON (*JavaScript Object Notation*) é um formato de dados leve e de fácil leitura e escrita, amplamente utilizado para a troca de informações entre diferentes camadas em projetos de *software*. O *JSON* foi criado como uma alternativa mais simples e eficiente ao formato XML, que era comumente utilizado para essa finalidade.

De acordo com Crockford (2006), o *JSON* é um formato de dados que utiliza uma sintaxe baseada em objetos *JavaScript*, sendo composto por pares de chave e valor. Ele é frequentemente utilizado em aplicações *web* para transferir dados entre um servidor e um cliente, permitindo que as informações sejam formatadas e interpretadas de maneira uniforme em diferentes plataformas e linguagens de programação.

O uso do *JSON* no desenvolvimento de projetos com diferentes camadas traz diversas vantagens, como a facilidade de leitura e escrita dos dados, a compatibilidade com diversas linguagens de programação e a capacidade de suportar

dados complexos em estruturas simples. Além disso, o *JSON* é um formato bastante popular e bem documentado, o que torna sua implementação mais simples e acessível.

5.1.10 NextJS

O *Next.js* é um framework de desenvolvimento *web* que foi criado pela equipe do Vercel e que utiliza o *ReactJS* como base. Ele oferece uma série de recursos adicionais que auxiliam no desenvolvimento de aplicações *web* escaláveis, como a renderização do lado do servidor (SSR) e a geração de sites estáticos (SSG).

Segundo Leal e Borges (2021), uma das principais vantagens do *Next.js* é a sua capacidade de integrar facilmente outras bibliotecas e *frameworks* populares de *JavaScript*, como o *TypeScript* para adicionar recursos de tipagem estática ao código e o *Styled Components* para simplificar a estilização de componentes.

O *Next.js* também é altamente otimizado para a construção de aplicações *web* de alta performance, com recursos como o carregamento de páginas em segundo plano e a divisão de código automática. Essas funcionalidades ajudam a garantir que as aplicações desenvolvidas com o *Next.js* sejam rápidas e eficientes.

Devido à sua popularidade e eficiência, o *Next.js* é amplamente utilizado em projetos de desenvolvimento de interfaces para aplicações *web*, tornando-se uma escolha popular entre desenvolvedores front-end. Segundo Oliveira (2021), empresas de renome, como Airbnb, Uber e TikTok, utilizam o *Next.js* em suas aplicações *web*.

5.1.11 MQTT.JS

A integração do MQTT com o *Next.js* é possível através do uso da biblioteca mqtt.js, que fornece uma API para se conectar a um *Broker* MQTT e enviar e receber mensagens (MQTT.*JS*, 2023). Essa integração é útil em aplicações que exigem comunicação em tempo real, como monitoramento e controle de dispositivos IoT.

Para integrar o mqtt.js ao *Next.js*, é necessário instalar a biblioteca através do gerenciador de pacotes *npm*. Em seguida, pode-se importar a biblioteca e criar uma instância do cliente MQTT, definindo o endereço do *Broker*, as credenciais de acesso e as configurações de conexão.

Uma vez que a conexão for estabelecida, é possível enviar e receber mensagens através de *call-backs* (retornos) que são acionados em resposta a eventos de conexão, subscrição e recebimento de mensagens. Essa comunicação pode ser utilizada para atualizar a interface do usuário em tempo real, por exemplo, exibindo o status de dispositivos conectados.

A integração do MQTT com o *Next.js* é uma maneira poderosa de adicionar recursos de comunicação em tempo real a aplicações *web* escaláveis e de alta performance (HARBISON, 2019). É uma opção especialmente interessante em projetos de IoT, em que a comunicação em tempo real é essencial para a operação adequada dos dispositivos (O'SULLIVAN, 2017).

5.1.12 Leaflet

O *Leaflet.js* é uma biblioteca em *JavaScript* de código aberto para criação de mapas interativos para a *web*. Ele é leve, fácil de usar e personalizável, permitindo a exibição de coordenadas geoespaciais em uma variedade de formatos, incluindo imagens, vetores e tiles. Ele é amplamente utilizado em projetos de mapeamento, geolocalização e análise espacial.

Conforme McCandless (2014) e Garcia e Brown (2014), a integração do *Leaflet* com o *Next.js* pode ser realizada através da criação de um componente *React* para renderizar o mapa, incluindo configurações como centro, zoom e camadas de visualização. Essa solução poderosa e eficiente, útil em diversos cenários, é possível graças à biblioteca *Leaflet* (LEAFLET, 2023) e ao framework *Next.js* (NEXT.*JS*, 2023).

5.1.13 GitHub

GitHub é uma plataforma online de hospedagem de código-fonte e gerenciamento de projetos baseada em *Git*. Ele permite que desenvolvedores e equipes colaborem em projetos de *software*, gerenciando versões do código e colaborando através de revisões de código, rastreamento de bugs e gerenciamento de projetos. A plataforma também oferece ferramentas de integração contínua e entrega contínua, permitindo que os desenvolvedores automatizem a compilação, testes e distribuição de seus aplicativos.

O *GitHub* é uma ferramenta poderosa para desenvolvedores, pois permite que eles colaborem em projetos de código aberto, contribuindo com correções de bugs, novos recursos e documentação. A plataforma também fornece recursos para rastrear problemas e bugs, permitindo que os desenvolvedores resolvam problemas rapidamente e melhorem a qualidade do *software*.

De acordo com O'Reilly e Loukides (2012), o *GitHub* revolucionou a forma como os desenvolvedores trabalham juntos em projetos de *software*, tornando mais fácil a colaboração e a contribuição de código. Além disso, Shields (2019) destaca que o *GitHub* se tornou uma ferramenta essencial para desenvolvedores e empresas, permitindo que equipes de todo o mundo colaborem em projetos de *software* e mantenham um controle rigoroso sobre o código-fonte.

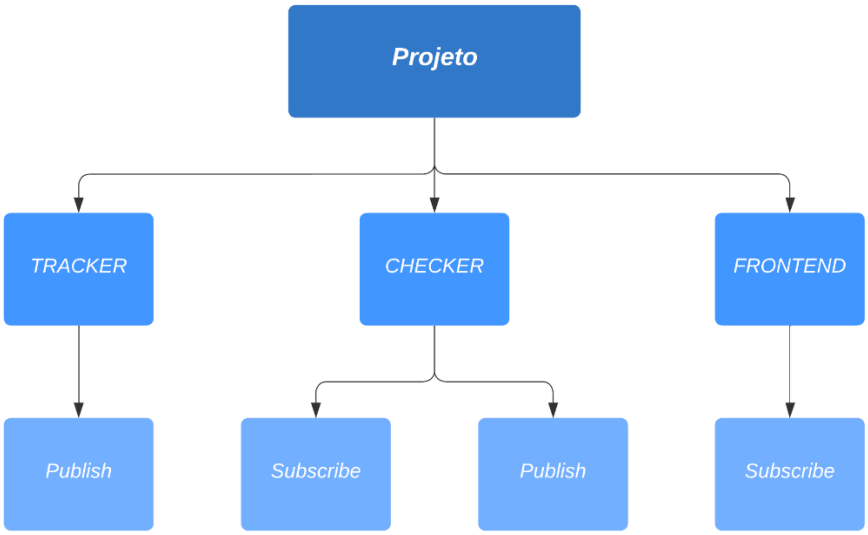
5.2 Metodologia

Para a realização do trabalho, fez-se necessário antes a obtenção de alguns aplicativos com intuito de tornar viável o desenvolvimento do projeto. Primeiro o ambiente de trabalho escolhido foi o *Visual Studio Code* ou *VSCode*, que representa um editor de texto com suporte a diversas linguagens de programação.

Também existe uma extensão necessárias para realizar o desenvolvimento do código do *tracker* e *checker*, a *PlataformIO*, que consiste em uma extensão do *VSCode* cujo intuito é permitir comunicar-se com o dispositivo através das portas do hardware, ou seja, permite gravar os códigos desenvolvidos no ESP32. Vale ressaltar que o Arduino IDE também foi utilizado para a gravação dos códigos desenvolvidos.

Com todos os *software*s, bibliotecas e extensões já disponíveis na máquina, inicialmente, buscou-se fazer a modelagem do projeto, de modo que se obtivesse um modelo organizacional do projeto, a fim de tornar mais fácil o entendimento das subdivisões de cada topologia. Esse modelo é ilustrado pela Figura 5.

Figura 5 – Modelo organizacional do projeto



Fonte: O autor.

De acordo com esse modelo, o projeto basicamente é constituído por três topologias principais, e cada topologia consegue se comunicar com o *Broker* a fim de fornecer ou receber dados. Os tópicos a seguir detalham cada topologia e explica afundo suas funcionalidades.

5.2.1 *Tracker*

A topologia do rastreador de drones, chamada de *tracker*, é composta por dois principais componentes, são eles o microcontrolador ESP32 e o módulo de GPS NEO-6M. Esse rastreador deve estar acoplado a um drone e sua função principal é fornecer a localização atual do drone.

Os componentes, microcontrolador e o módulo GPS, devem ser conectados de acordo com o código gravado no microcontrolador. Então, antes de entender a conexão dos dispositivos, primeiro deve-se entender o código do rastreador, disposto no Anexo A. Para facilitar a compreensão, o código será comentado por etapas.

Inicialmente deve-se instalar e incluir todas as bibliotecas necessárias para o código, mostradas na Figura 6. Essas bibliotecas ajudam na utilização de diversas funções, além de tornar possível a obtenção dos dados fornecidos pelo módulo.

Figura 6 – Bibliotecas necessárias para o *tracker*

```
1   #include <WiFi.h>
2   #include <PubSubClient.h>
3   #include <TinyGPSPlus.h>
4   #include <SoftwareSerial.h>
5   #include <ArduinoJson.h>
```

Fonte: O autor.

Após isso, definiu-se os principais parâmetros para que o rastreador pudesse fornecer dados confiáveis e que não afetasse no desempenho do microcontrolador. A Figura 7 retrata um exemplo de como esses parâmetros foram instanciados.

Figura 7 – Instanciação dos parâmetros do *tracker*

```
7   #define RXD2 16
8   #define TXD2 17
9   #define GPS_BAUDRATE 9600
10  #define WIFI_SSID "redewifi"
11  #define WIFI_PASSWORD "senhawifi"
12  #define MQTT_SERVER "broker.emqx.io"
13  #define MQTT_PORT 1883
14  #define MQTT_TOPIC_PREFIX "data/location"
15  #define MQTT_USERNAME "emqx"
16  #define MQTT_PASSWORD "public"
17  #define LED 2
18  #define LED_EXT 15
19  #define DELAY 5000
```

Fonte: O autor.

Já a Tabela 1, mostra todos os parâmetros e descreve o que cada um representa.

Tabela 2 – Parâmetros da topologia *tracker*

Parâmetro	Descrição
RXD2	Pino RXD2 do ESP32. Observação: deve ser conectado ao TX do GPS
TXD2	Pino TXD2 do ESP32. Observação: deve ser conectado ao RX do GPS
GPS_BAUDRATE	A taxa de transmissão de bits de um sinal serial para o GPS. Observação: Geralmente é de 9600
WIFI_SSID	SSID da rede *WiFi*
WIFI_PASSWORD	Senha da rede *WiFi*
MQTT_SERVER	Endereço do servidor MQTT
MQTT_PORT	Porta do servidor MQTT
MQTT_TOPIC_PREFIX	Prefixo do tópico MQTT para publicar as coordenadas.
MQTT_USERNAME	Usuário do servidor
MQTT_PASSWORD	Senha do usuário do servidor
LED	Pino referente ao *LED* interno do ESP32
LED_EXT	Pino referente ao *LED* externo ao esp32 que será utilizado. Observação: verificar o datasheet
DELAY	Delay em milissegundos (ms) para envio de dados. Exemplo: 5*1000 = 5 segundos

Fonte: O autor.

Feito isso, os objetos necessários foram instanciados, como mostra a Figura 8.

Figura 8 – Instanciação dos objetos do *tracker*

```
21  SoftwareSerial ss(RXD2, TXD2);
22  TinyGPSPlus gps;
23  WiFiClient wifiClient;
24  PubSubClient client(wifiClient);
25  const int json = JSON_OBJECT_SIZE(10);
26  StaticJsonDocument<json> doc;
27  size_t freeHeap = ESP.getFreeHeap();
28  DynamicJsonDocument doc2((freeHeap / 2));
29  JsonArray array = doc2.to<JsonArray>();
```

Fonte: O autor.

Ainda da Figura 8, pode-se verificar que a biblioteca *SoftwareSerial* está sendo usada para criar uma comunicação serial com um módulo GPS. Além disso, estão sendo utilizadas as bibliotecas *TinyGPSPlus*, responsável por fazer o *parsing*

(tratamento) dos dados do GPS; a *WiFiClient*, para criar um cliente *WiFi*; e a *PubSubClient* para criar um cliente MQTT.

Em seguida, é definido um objeto *StaticJsonDocument* chamado *doc* com tamanho igual a um objeto que deve ser enviado. Esse objeto *doc*, será utilizado para armazenar as informações do GPS e posteriormente enviar para o servidor MQTT.

Também é definido um objeto *DynamicJsonDocument* chamado *doc2*, sua função é armazenar objetos do tipo *doc* quando não houver conexão com *internet*, dessa forma, torna-se necessário definir um tamanho máximo para esse objeto. Com isso, pensou-se em usar metade de memória livre disponível na *heap* (área de memória dinâmica usada pelo programa durante a execução) do ESP32, visando garantir que o sistema tenha memória suficiente para operar corretamente.

Feito isso, criou-se o *array* a partir do objeto *doc2*, que será utilizado para armazenar vários objetos *JSON* em um único array quando não houver conexão com a *internet*. Esse array deverá ser enviado quando a conexão for reestabelecida.

A Figura 9 e a Figura 10, mostram os padrões de como serão enviados os objetos para o servidor MQTT, também chamados de *DTO*'s.

Figura 9 – Padrão de envio individual do *tracker*

```
{
    "device": "esp32",
    "sensor": "esp32-00:00:00:00:00:00",
    "cordinates":{
        "lat":"0.000000",
        "lon":"0.000000"
    }
}
```

Fonte: O autor.

Figura 10 – Padrão de envio coletivo do *tracker*

```
[
{
    "device": "esp32",
    "sensor": "esp32-00:00:00:00:00:00",
    "cordinates":{
        "lat":"0.000000",
        "lon":"0.000000"
    }
},{
    "device": "esp32",
    "sensor": "esp32-00:00:00:00:00:00",
    "cordinates":{
        "lat":"0.000000",
        "lon":"0.000000"
    }
}
]
```

Fonte: O autor.

Com todos os objetos instanciados, definiu-se então a função responsável por realizar a conexão do microcontrolador com a rede *WiFi*, possibilitando então o envio dos dados, além disso, para assegurar que a conexão foi estabelecida o *LED* interno da placa de desenvolvimento do ESP32 deve acender. A Figura 11, mostra como é realizada a conexão e também o acionamento do *LED*.

Figura 11 – Conexão com a *internet*

```
32  void setup_wifi()
33  {
34    delay(1000);
35
36    Serial.println();
37    Serial.print("Conectando ao wifi: ");
38    Serial.println(WIFI_SSID);
39
40    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
41
42    // Enquanto nao for estabelecida, fica aguardadando
43    while (WiFi.status() != WL_CONNECTED)
44    {
45      delay(1000);
46      Serial.print(".");
47    }
48
49    // Conexao com wifi estabelecida. Aciona o led interno
50    Serial.println("");
51    Serial.println("WiFi conectado");
52    Serial.println("IP address: ");
53    Serial.println(WiFi.localIP());
54    digitalWrite(LED, HIGH);
55  }
```

Fonte: O autor.

Também foi criada uma função que deve simular um *delay* customizado, disposta na Figura 12, ou seja, é utilizada para realizar uma pausa no código por um determinado tempo (em milissegundos), enquanto também permite que o objeto “gps” possa receber dados do módulo GPS conectado.

Figura 12 – Função customizada para temporização e verificação do GPS

```
58  static void smartDelay(unsigned long ms)
59  {
60    unsigned long start = millis();
61    do
62    {
63      while (ss.available())
64        gps.encode(ss.read());
65    } while (millis() - start < ms);
66  }
```

Fonte: O autor.

Definidas todas as funções, agora são definidas as duas últimas funções, essas que são essenciais do Arduino IDE para o desenvolvimento do código baseado em sistemas embarcados, são elas: *setup* e *loop*.

A função *setup* é executada apenas uma vez no início da execução do programa no microcontrolador e é responsável por configurar as condições iniciais do sistema, como inicializar as portas de entrada e saída, estabelecer a comunicação serial, configurar o acesso à rede, entre outras.

Na Figura 13, a função *setup* implementada inicia a comunicação serial, configura os pinos de *LED*s como saídas digitais, executa a função *setup_wifi*, responsável pela conexão com a rede e configura o cliente MQTT para que seja possível se conectar com o servidor definido. A função também imprime a quantidade de memória livre disponível na *heap*.

Figura 13 – Função *setup*

```
69  void setup()
70  {
71    Serial.begin(115200);
72    ss.begin(GPS_BAUDRATE);
73    pinMode(LED, OUTPUT);
74    pinMode(LED_EXT, OUTPUT);
75    setup_wifi();
76    client.setServer(MQTT_SERVER, MQTT_PORT);
77    Serial.print(freeHeap);
78  }
```

Fonte: O autor.

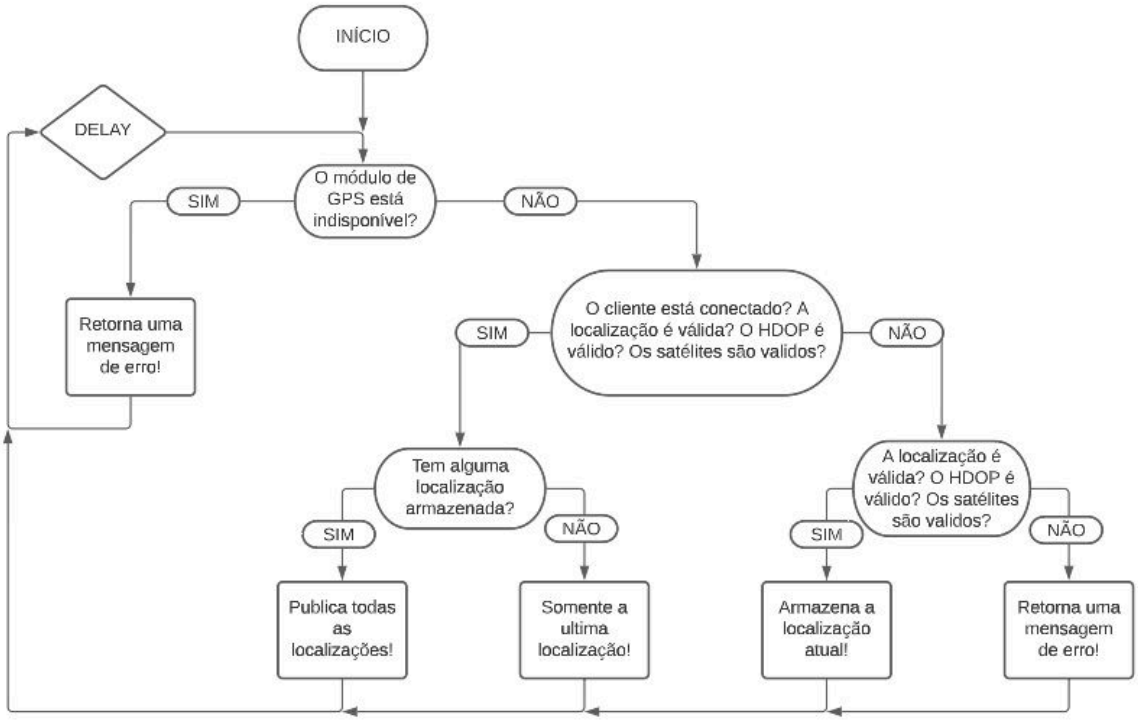
Já a função *loop* é executada continuamente após a execução da função *setup* e é responsável por executar as ações do programa, que podem incluir leitura de sensores, acionamento de atuadores, envio de mensagens, processamento de dados, entre outras. É dentro da função *loop* que a lógica principal do programa é implementada e executada repetidamente até que o microcontrolador seja desligado ou reiniciado.

De maneira resumida, a função *loop* implementada, mostrada na Figura XX, está sendo utilizada para obter dados de GPS, armazenar esses dados em um array ou publicá-los em um servidor MQTT, dependendo se a conexão MQTT está

estabelecida ou não. Além disso, a função está sendo utilizada para ativar/desativar um *LED* externo de acordo com o status da conexão do GPS.

A Figura 14 apresentada abaixo, descreve como a função *loop* deve funcionar através de um fluxograma, vale lembrar que quando há mais de uma pergunta numa entidade, todas devem ser verdadeiras, caso alguma não seja verdadeira, a resposta é tida como um não.

Figura 14 – Fluxograma de funcionamento da função *loop*

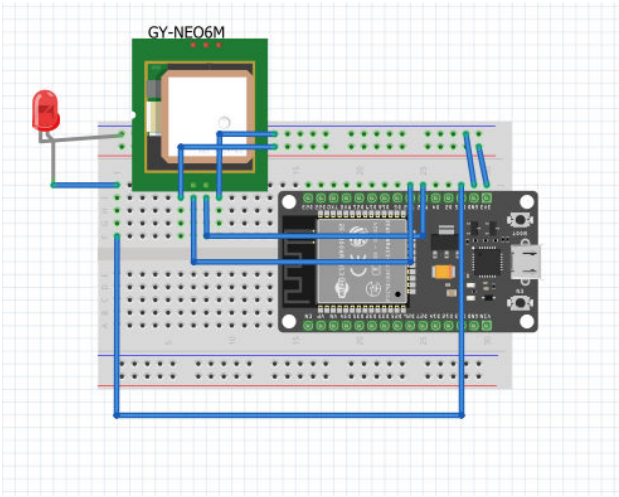


Fonte: O autor.

Outra observação a ser mencionada é o HDOP (*Horizontal Dilution of Precision*), que consiste em um dos parâmetros que descrevem a qualidade do sinal do GPS, e é uma medida da precisão horizontal da posição calculada. Em outras palavras, um HDOP menor indica uma melhor precisão, enquanto um HDOP maior indica uma menor precisão.

Para finalizar a descrição dessa topologia, e considerando os parâmetros definidos na Figura 7 (parâmetros), a conexão entre o módulo NEO-6M, a placa de desenvolvimento do ESP32 e o *LED* externo, devem ser feitas igual ao esquemático mostrado na Figura 15.

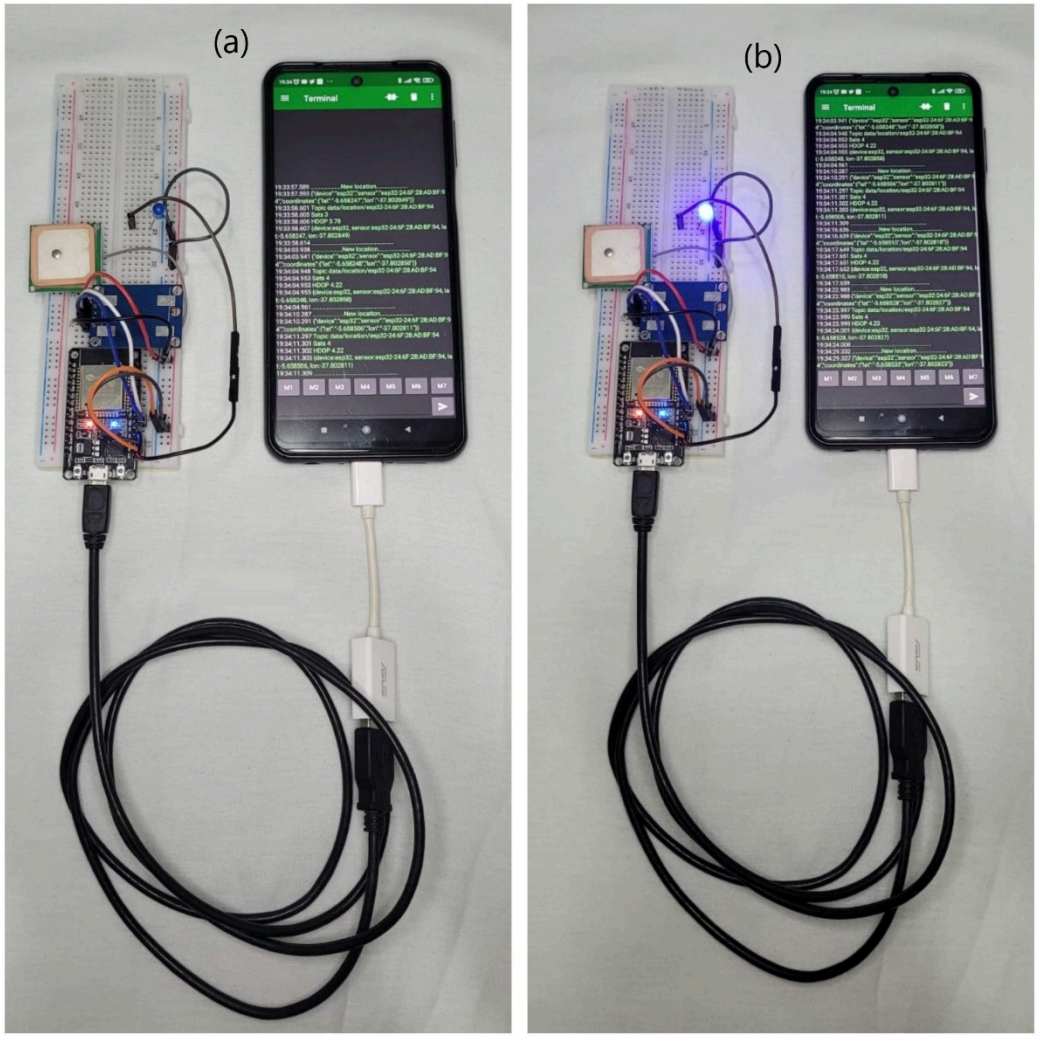
Figura 15 – Esquema de conexões do *hardware*



Fonte: O autor.

Para que haja uma visualização de como foram realizados os testes, a Figura 16, ilustra como o *tracker* foi elaborado suas sinalizações. Nota-se que há uma *protoboard*, o microcontrolador é alimentado por um celular, que além de ser utilizado como fonte de alimentação também serviu como *GATEWAY* de *Internet* (roteador). Seus testes foram realizados fazendo percursos de carro. Com isso, tanto a latitude como a longitude variavam de acordo com o percurso.

Figura 16 – *Tracker* (a) esperando o envio dos dados; (b) enviando os dados.



Fonte: O autor.

5.2.2 *Checker*

Já a topologia de verificação de coordenadas, chamada de *checker*, é composta apenas pelo microcontrolador ESP32 e um *LED* de sinalização. A conexão desses componentes, apesar de simples, deve ser realizada de acordo com o código gravado no microcontrolador.

Então, antes de mostrar o layout do *checker*, primeiro deve-se entender o código, disposto no Anexo B. Para facilitar a compreensão, o código será comentado por etapas, de maneira análoga a topologia anterior.

Inicialmente deve-se instalar e incluir todas as bibliotecas necessárias para o código, mostradas na Figura 17. Essas bibliotecas, como já mencionado, ajudam na

utilização de diversas funções e são incluídas de maneira semelhante ao da topologia anterior.

Figura 17 – Bibliotecas necessárias para o *checker*

```
1   #include <WiFi.h>
2   #include <PubSubClient.h>
3   #include <cmath>
4   #include <ArduinoJson.h>
```

Fonte: O autor.

Após isso, definiu-se os principais parâmetros para que o verificador pudesse fornecer e verificar os dados de maneira confiável. A Figura 18 retrata um exemplo de como esses parâmetros foram instanciados.

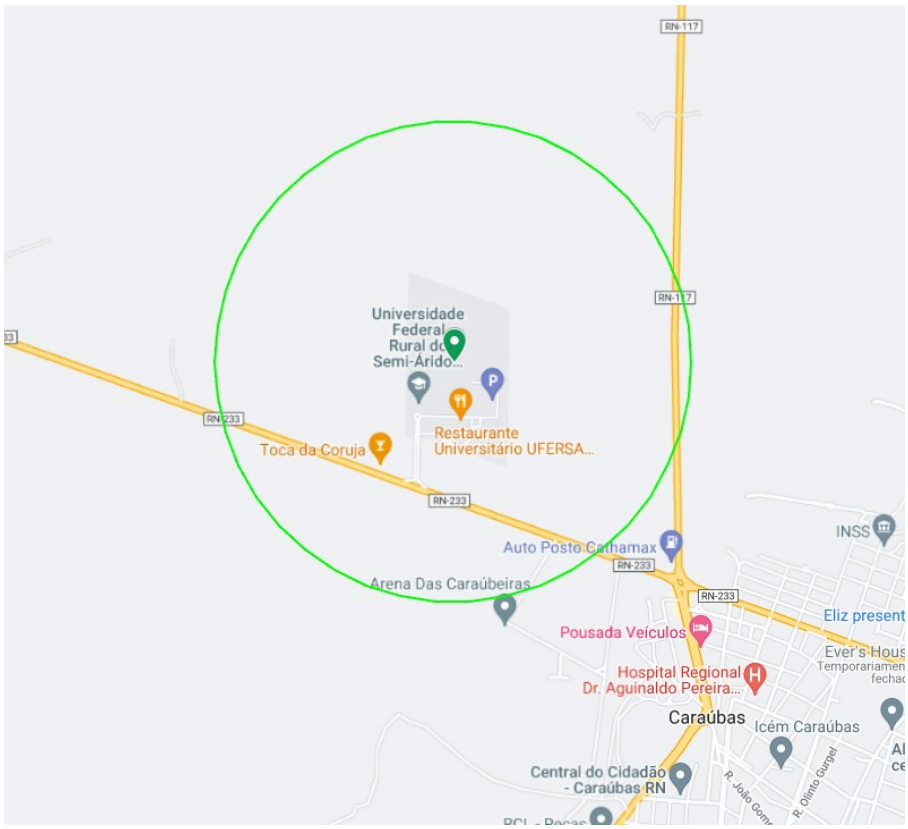
Figura 18 – Instanciação dos parâmetros do *checker*

```
6   #define RXD2 16
7   #define TXD2 17
8   #define GPS_BAUDRATE 9600
9   #define WIFI_SSID "redewifi"
10  #define WIFI_PASSWORD "senhawifi"
11  #define MQTT_SERVER "broker.emqx.io"
12  #define MQTT_PORT 1883
13  #define MQTT_TOPIC_PREFIX "data/checker"
14  #define MQTT_USERNAME "emqx"
15  #define MQTT_PASSWORD "public"
16  #define LED 2
17  #define LED_EXT 15
```

Fonte: O autor.

Além disso, algumas constantes específicas também devem ser parametrizadas. Elas dizem respeito a posição de referência em que o *checker* deve atuar. Ou seja, tanto a latitude como a longitude de determinado local, bem como uma distância máxima desse ponto. A Figura 19 abaixo demonstra, de maneira didática, a área que o *checker* deve verificar.

Figura 19 – Ilustração do funcionamento do *checker*



Fonte: *Google Maps*.

Além disso, também é definida a constante referente ao tópico do drone, ou seja, ela recebe o valor do tópico em que o *tracker* irá publicar suas informações. Essas constantes são definidas no código como mostra a Figura 20, e basicamente são compostas por latitude, longitude e o raio de distância de verificação dado em metros.

Figura 20 – Definição das configurações do *checker*

```
19  const double LATITUDE_REFERENCE = -5.77179;
20  const double LONGITUDE_REFERENCE = -37.56939;
21  const double MAX_DISTANCE = 100;
22  const String TOPIC_DRONE = "data/tracker/esp32-00:00:00:00:00:00";
```

Fonte: O autor.

A Tabela 3, mostra todos os parâmetros citados anteriormente e descreve o que cada um representa.

Tabela 3 – Parâmetros da topologia *checker*

Parâmetro	Descrição
RXD2	Pino RXD2 do ESP32. Observação: deve ser conectado ao TX do GPS
TXD2	Pino TXD2 do ESP32. Observação: deve ser conectado ao RX do GPS
GPS_BAUDRATE	A taxa de transmissão de bits de um sinal serial para o GPS. Observação: Geralmente é de 9600
WIFI_SSID	SSID da rede *WiFi*
WIFI_PASSWORD	Senha da rede *WiFi*
MQTT_SERVER	Endereço do servidor MQTT
MQTT_PORT	Porta do servidor MQTT
MQTT_TOPIC_PREFIX	Prefixo do tópico MQTT para publicar as coordenadas.
MQTT_USERNAME	Usuário do servidor
MQTT_PASSWORD	Senha do usuário do servidor
LED	Pino referente ao *LED* interno do ESP32
LED_EXT	Pino referente ao *LED* externo ao esp32 que será utilizado. Observação: verificar o datasheet
LATITUDE_REFERENCE	Latitude de referência para calcular o distanciamento do dispositivo
LONGITUDE_REFERENCE	Longitude de referência para calcular o distanciamento do dispositivo
MAX_DISTANCE	Raio de distância limite usada para verificar uma área. Observação: dada em metros
TOPIC_DRONE	Tópico MQTT responsável por receber a localização atual do drone de tempos em tempos

Fonte: O autor.

As linhas do código mostradas na Figura 21, criam quatro objetos: um objeto *WiFiClient*, um objeto *PubSubClient* e dois objetos *StaticJsonDocument*.

Figura 21 – Instanciação dos objetos do *checker*

```
24  WiFiClient wifiClient;
25  PubSubClient client(wifiClient);
26  StaticJsonDocument<192> doc;
27  StaticJsonDocument<256> doc2;
```

Fonte: O autor.

O objeto *WiFiClient* é utilizado para criar a conexão com a rede *WiFi*. Ele será passado como argumento para o objeto *PubSubClient*, que utiliza esta conexão para

se conectar ao servidor MQTT. Ou seja, o objeto *PubSubClient* é utilizado para publicar e receber mensagens no servidor MQTT.

Os dois objetos *StaticJsonDocument* são utilizados para manipular dados no formato *JSON*. Eles têm um tamanho máximo pré-definido diferentes devido aos seus diferentes padrões, e seus tamanhos são de 192 e 256 *bytes*, respectivamente. Estes objetos são úteis para criar e manipular mensagens no formato *JSON* antes de publicá-las no servidor MQTT.

As figuras abaixo mostram os diferentes padrões de *JSON*, onde no *doc* ele corresponde a Figura 22 e é responsável por receber um objeto desse tipo, já o *doc2* é referente a Figura 23 e sua função é enviar um objeto com esse formato.

Figura 22 – Padrão de recebimento de dados enviados pelo *tracker*

```
{
    "device": "esp32",
    "sensor": "esp32-00:00:00:00:00:00",
    "cordinates":{
        "lat":"0.000000",
        "lon":"0.000000"
    }
}
```

Fonte: O autor.

Figura 23 – Padrão de envio de dados enviados pelo *checker*

```
{
    "device": "esp32",
    "sensor": "esp32-00:00:00:00:00:00",
    "distance": "10000000000000000000000000000.12921393193",
    "isClose": false,
    "referenceLocation":  {
        "lat":"0.000000",
        "lon":"0.000000",
        "max_distance": "100000"
    }
}
```

Fonte: O autor.

Com todos os objetos instanciados, e parâmetros definidos, criou-se então a função responsável por realizar a conexão do microcontrolador com a rede *WiFi*, de maneira análoga a do *tracker*, ela possibilita então o recebimento e envio dos dados, além disso, para assegurar que a conexão foi estabelecida o *LED* interno da placa de desenvolvimento do ESP32 deve acender. A Figura 24, mostra como é realizada a conexão e também o acionamento do *LED*.

Figura 24 – Função que realiza a conexão com a *internet*

```
30  void setup_wifi()
31  {
32    delay(1000);
33
34    Serial.println();
35    Serial.print("Conectando ao wifi: ");
36    Serial.println(WIFI_SSID);
37
38    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
39
40    // Enquanto nao for estabelecida, fica aguardadando
41    while (WiFi.status() != WL_CONNECTED)
42    {
43      delay(1000);
44      Serial.print(".");
45    }
46
47    // Conexao com wifi estabelecida. Aciona o led interno
48    Serial.println("");
49    Serial.println("WiFi conectado");
50    Serial.println("IP address: ");
51    Serial.println(WiFi.localIP());
52    digitalWrite(LED, HIGH);
53  }
```

Fonte: O autor.

Também foi criada uma função que deve conectar-se ao servidor MQTT, disposta na Figura 25, ou seja, é utilizada para realizar a conexão com servidor MQTT e realizar a inscrição no tópico do drone. Ou seja, ela receberá todas as atualizações de localização que o *tracker* fornece.

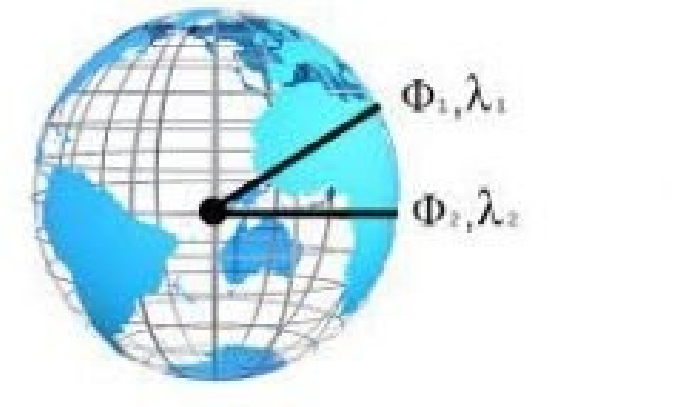
Figura 25 – Função que conecta o *checker* ao *Broker* MQTT

```
56  void connectToMqtt()
57  {
58    while (!client.connected())
59    {
60      String client_id = "esp32-";
61      client_id += String(WiFi.macAddress());
62      Serial.println("Conectando ao broker MQTT...");
63      if (client.connect(client_id.c_str(), MQTT_USERNAME, MQTT_PASSWORD))
64      {
65        Serial.println("Conectado ao broker MQTT!");
66        if (client.subscribe(TOPIC_DRONE.c_str()))
67        {
68          Serial.println("Subscribed");
69        }
70      }
71      else
72      {
73        Serial.print("Falha na conexão MQTT: ");
74        Serial.println(client.state());
75        delay(5000);
76      }
77    }
78  }
```

Fonte: O autor.

Outra função de grande importância foi implementada e deve retornar a distância em metros de duas diferentes localizações. Existem muitos algoritmos que lidam com esse problema de cálculo de distância. Cada algoritmo está relacionado a uma forma ou representação específica. Os principais algoritmos analisados foram as formulas de Haversine e de Vincenty.

Figura 26 – Figura ilustrativa de dois pontos geográficos diferentes



Fonte: NEOVA SOLUTIONS, 2023.

Segundo Neova Solutions (2023), existem prós e contras nas utilizações dessas formulas. Utilizando a fórmula de Haversine obtém-se dados menos precisos, diferentemente da fórmula de Vincenty. Em compensação, a fórmula de Haversine utiliza menos energia, pois utiliza uma computação mais simples devido assumir a Terra sendo uma elipse perfeita.

Já a fórmula de Vincenty oferece dados mais precisos pois leva em consideração que a Terra não é uma elipse perfeita, em contrapartida, por utilizar um modelo mais intensivo computacionalmente, consequentemente, terá um desempenho mais lento e aumentará o consumo de bateria.

Levando em consideração as principais características de cada modelo, optou-se por utilizar Vincenty, uma vez que, o microcontrolador correspondente ao *checker* atuará de maneira fixa em locais de fácil acesso, possibilitando uma fácil manutenção.

A formulação dos problemas direto e inverso de Vincenty pode ser encontrado em toda sua plenitude em Vincenty (1975). Suas fórmulas foram testadas e verificadas com erros menores de 0,115 mm, considerando a distância de 18000 km em todos os casos testados (THOMAS & FEATHERSTONE, 2005).

Sua função foi implementada e disposta no código, como mostra a Figura 27.

Figura 27 – Função que simula a Fórmula de Vincenty

```
double vincenty(double lat1, double lon1, double lat2, double lon2)
{
  const double a = 6378137;          // semieixo maior da Terra em metros
  const double b = 6356752.314245;   // semieixo menor da Terra em metros
  const double f = 1 / 298.257223563; // achatamento da Terra

  double L = (lon2 - lon1) * M_PI / 180;
  double U1 = atan((1 - f) * tan(lat1 * M_PI / 180));
  double U2 = atan((1 - f) * tan(lat2 * M_PI / 180));
  double sinU1 = sin(U1), cosU1 = cos(U1);
  double sinU2 = sin(U2), cosU2 = cos(U2);

  double lambda = L, lambdaP, sinLambda, cosLambda, sinSigma, cosSigma, sigma, sinAlpha, cosSqAlpha, cos2SigmaM, C;
  int iterLimit = 100;
  do
  {
    sinLambda = sin(lambda), cosLambda = cos(lambda);
    sinSigma = sqrt((cosU2 * sinLambda) * (cosU2 * sinLambda) +
                    (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda) * (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda));
    if (sinSigma == 0)
      return 0; // pontos coincidentes
    cosSigma = sinU1 * sinU2 + cosU1 * cosU2 * cosLambda;
    sigma = atan2(sinSigma, cosSigma);
    sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma;
    cosSqAlpha = 1 - sinAlpha * sinAlpha;
    cos2SigmaM = cosSigma - 2 * sinU1 * sinU2 / cosSqAlpha;
    C = f / 16 * cosSqAlpha * (4 + f * (4 - 3 * cosSqAlpha));
    lambdaP = lambda;
    lambda = L + (1 - C) * f * sinAlpha *
                    (sigma + C * sinSigma * (cos2SigmaM + C * cosSigma * (-1 + 2 * cos2SigmaM * cos2SigmaM)));
  } while (fabs(lambda - lambdaP) > 1e-12 && --iterLimit > 0);

  if (iterLimit == 0)
    return 0; // fórmula não convergiu

  double uSq = cosSqAlpha * (a * a - b * b) / (b * b);
  double A = 1 + uSq / 16384 * (4096 + uSq * (-768 + uSq * (320 - 175 * uSq)));
  double B = uSq / 1024 * (256 + uSq * (-128 + uSq * (74 - 47 * uSq)));
  double deltaSigma = B * sinSigma * (cos2SigmaM + B / 4 * (cosSigma * (-1 + 2 * cos2SigmaM * cos2SigmaM) - B / 6 * cos2SigmaM * (-3 + 4 * sinSigma * sinSigma) * (-3 + 4 * cos2SigmaM * cos2SigmaM)));
  double s = b * A * (sigma - deltaSigma);

  return s;
}
```

Fonte: O autor.

Outra função necessária e que implementada, mostrada na Figura 28, consiste em uma validação, pois esta compara o resultado da distância retornada pela Fórmula de Vincenty a uma distância máxima.

Figura 28 – Função que verifica se o *tracker* está ou não dentro da área do *checker*

```
126  bool deviceIsTooFar(float lat, float lon, String *distance)
127  {
128    double dist = vincenty(lat, lon, LATITUDE_REFERENCE, LONGITUDE_REFERENCE);
129
130    *distance = String(dist);
131
132    if (dist > MAX_DISTANCE)
133      return true;
134
135    return false;
136  }
```

Fonte: O autor.

Dessa forma, se a distância entre o *tracker* e o *checker* for maior que a distância parametrizada no escopo do código, então, o drone acoplado ao *tracker* ainda está muito longe, porém, se a distância entre eles for menor ou igual a distância parametrizada, é porque o drone entrou na área analisada.

A função de maior importância nessa topologia é chamada de retorno (call-back), e consiste nas instruções que o *checker* deve realizar quando recebe uma nova localização.

Basicamente, a função desmembra o objeto *JSON* recebido e extrai o nome do sensor, nome do dispositivo, coordenadas de latitude e longitude desse objeto. Em seguida, chama a função que verifica se o dispositivo está muito longe do local de referência, essa função chama outra função realiza os cálculos da formula de Vincenty, e com isso retorna se está perto ou longe, em outras palavras, se está dentro da área ou não.

A Figura 29 mostra a função call-back.

Figura 29 – Função de retorno

```
void callback(char *topic, byte *payload, unsigned int length)
{

  // dados recebidos do tracker
  deserializeJson(doc, payload);
  const char *sensor = doc["sensor"];
  const char *device = doc["device"];
  float latitude = doc["coordinates"]["lat"];
  float longitude = doc["coordinates"]["lon"];

  String distance;

  if (deviceIsTooFar(latitude, longitude, &distance))
  {

    if (client.connected())
    {
      String client_id = "esp32-";
      client_id += String(WiFi.macAddress());
      String topic = String(MQTT_TOPIC_PREFIX) + "/" + client_id;
      Serial.println("..................Device is far.....................");
      String new_data;
      doc2.clear();
      doc2["device"] = "esp32";
      doc2["sensor"] = client_id;
      doc2["distance"] = String(distance);
      doc2["isClose"] = false;
      doc2["referenceLocation"]["lat"] = String(LATITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["lon"] = String(LONGITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["max_distance"] = String(MAX_DISTANCE);
      serializeJson(doc2, new_data);
      client.publish(topic.c_str(), new_data.c_str());
      Serial.println(new_data);
      Serial.println(topic);
      Serial.println("..................................................");
    }

    digitalWrite(LED_EXT, LOW); // apaga o led sinalizando que o dispositivo está muito longe
    Serial.println("{Lat: " + String(latitude, 6) + "," + " Long: " + String(longitude, 6) + "," + distance + "m }");
  }
  else
  {

    if (client.connected())
    {
      String client_id = "esp32-";
      client_id += String(WiFi.macAddress());
      String topic = String(MQTT_TOPIC_PREFIX) + "/" + client_id;
      Serial.println("..................Device is close.....................");
      String new_data;
      doc2.clear();
      doc2["device"] = "esp32";
      doc2["sensor"] = client_id;
      doc2["distance"] = String(distance);
      doc2["isClose"] = true;
      doc2["referenceLocation"]["lat"] = String(LATITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["lon"] = String(LONGITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["max_distance"] = String(MAX_DISTANCE);
      serializeJson(doc2, new_data);
      client.publish(topic.c_str(), new_data.c_str());
      Serial.println(new_data);
      Serial.println(topic);
      Serial.println("..................................................");
    }

    digitalWrite(LED_EXT, HIGH); // acende led sinalizando que o dispositivo está muito perto
    Serial.println("{Lat: " + String(latitude, 6) + "," + " Long: " + String(longitude, 6) + "," + distance + "m }");
  }
}
```

Fonte: O autor.

Se o dispositivo estiver muito longe, publicará uma mensagem *JSON* no *Broker* MQTT com o nome do sensor, nome do dispositivo, distância do local de referência e o sinalizador "*isClose*" como falso. Se o dispositivo estiver perto, publicará uma mensagem *JSON* no *Broker* MQTT com as mesmas informações de antes, mas com "*isClose*" definido como verdadeiro. Por fim, a função imprime as informações de latitude, longitude e distância no monitor serial para fins de depuração.

A função *setup*, como já foi explicado é responsável por configurar todas as parametrizações iniciais, já a função *loop* é responsável por manter a conexão estabelecida e ficar verificando se há algum novo dado. Essas funções são mostradas nas Figuras 30 e 31, respectivamente.

Figura 30 – Função *setup* do *checker*

```
210  void setup()
211  {
212    Serial.begin(115200);
213    pinMode(LED, OUTPUT);
214    pinMode(LED_EXT, OUTPUT);
215    setup_wifi();
216    client.setServer(MQTT_SERVER, MQTT_PORT);
217    client.setCallback(callback);
218
219    connectToMqtt();
220  }
```

Fonte: O autor.

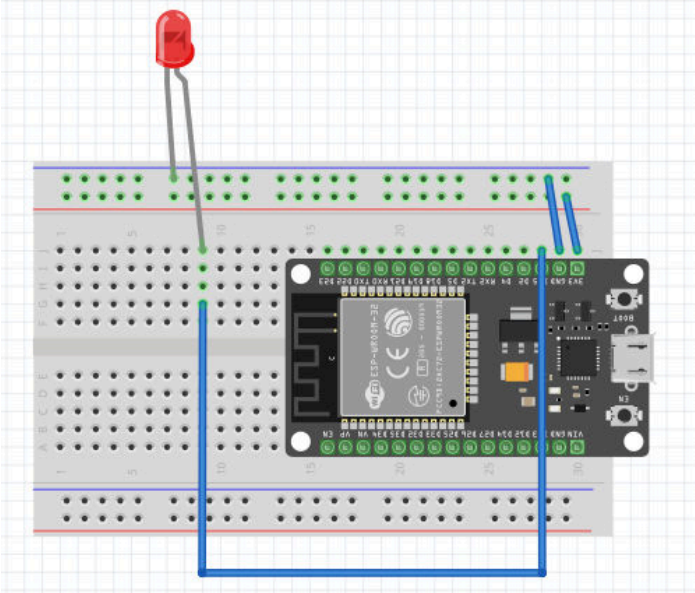
Figura 31 – Função *loop* do *checker*

```
223  void loop()
224  {
225    // Mantenha a conexão MQTT ativa
226    if (!client.connected())
227    {
228      connectToMqtt();
229    }
230    client.loop();
231  }
```

Fonte: O autor.

Para finalizar a descrição dessa topologia, e considerando os parâmetros definidos na Figura 18 (parâmetros), a conexão entre a placa de desenvolvimento do ESP32 e o *LED* externo, devem ser feitas igual ao esquemático mostrado na Figura 32.

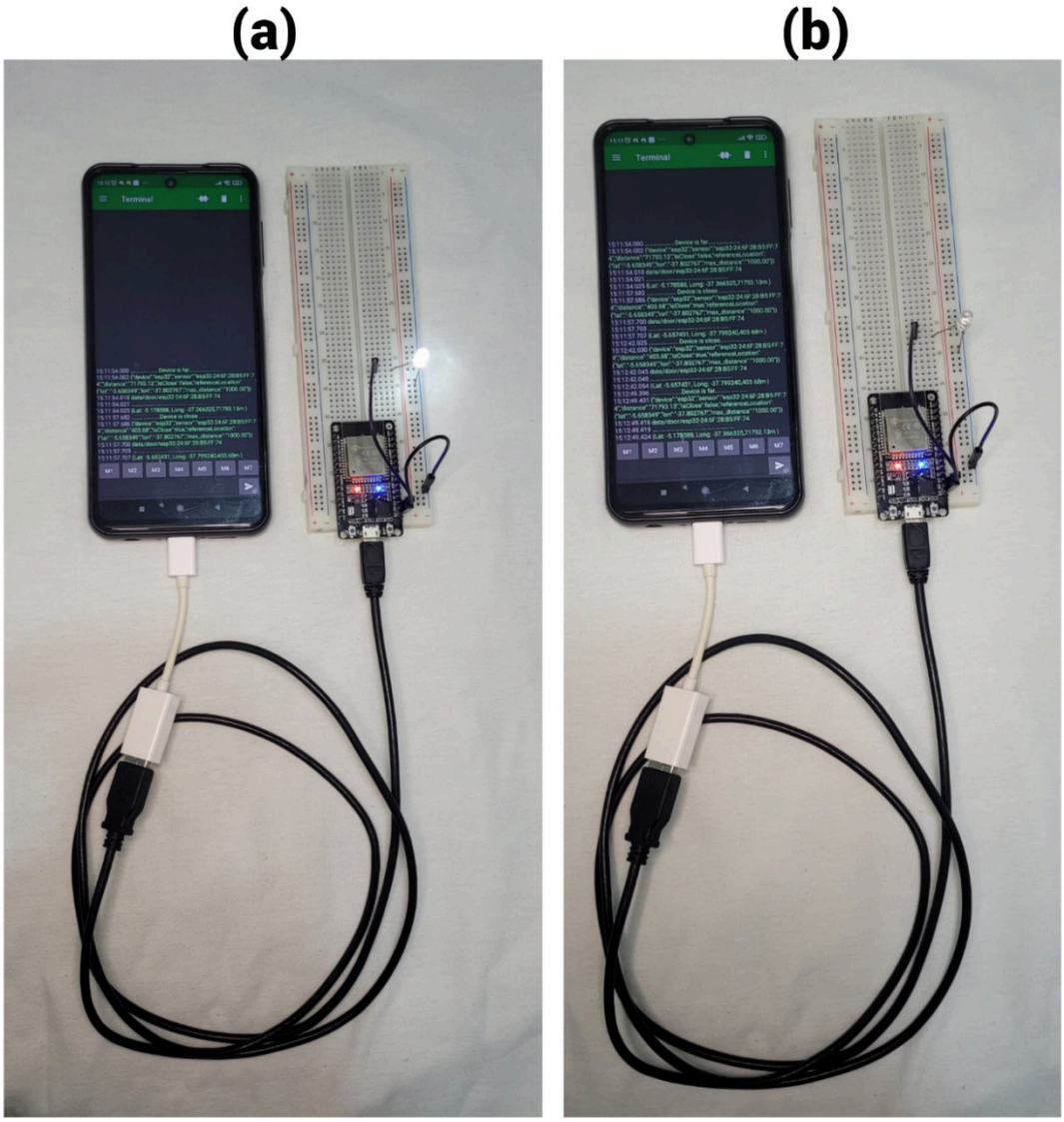
Figura 32 – Esquema de conexão do hardware



Fonte: O autor.

Para que haja uma visualização de como foram realizados os testes, a Figura 33, ilustra como o *tracker* foi elaborado suas sinalizações. Nota-se que há uma protoboard, o microcontrolador é alimentado por um celular, que foi utilizado apenas como fonte de alimentação, o dispositivo foi conectado numa rede local através do wifi que serviu como *GATEWAY* de *Internet*. Seus testes foram realizados obtendo os dados do *tracker* enquanto ele estava em movimento. Com isso, a distância entre ambos variava e o *LED* era acionado quando essa distância era menor do que o raio da área definida.

Figura 33 – *Checker* (a) o drone está dentro da área especificada; (b) o drone está fora da área especificada.



Fonte: O autor.

5.2.3 *Frontend*

A última topologia desenvolvida foi o *Frontend*, que consiste basicamente, em um *website* que exibe um mapa junto com todas as informações de localização e os status dos dispositivos de verificação (*checker*).

Para sua implementação, fez-se necessário antes a instalação de uma dependência necessária, o *Node.js,* que é responsável por fazer o servidor funcionar, e a partir disso, tornar possível o desenvolvimento e teste de todas as alterações.

A construção da aplicação se deu de forma gradativa, onde suas alterações eram armazenadas na plataforma do *GitHub* que, como mencionada, consiste em um sistema de gerenciamento de projetos e versões de códigos, basicamente, uma rede social de criada para desenvolvedores.

O *GitHub* funciona como um deposito online, onde há a hospedagem do código-fonte de acordo com as alterações de versões feitas pelo programador. Para fins, de desenvolvimento, a plataforma inicialmente foi inserida como um projeto privado, com acesso apenas dos participantes do desenvolvimento do projeto.

Definido suas principais características, sua estrutura inicial foi criada a partir de *NextJS*, que como já foi explicado, permite criar aplicações *web* baseadas em *React*. Com isso, deu-se início ao desenvolvimento da plataforma utilizando o projeto criado pelo *NextJS* com suas configurações iniciais.

Em seguida, adicionou-se algumas bibliotecas necessárias para o desenvolvimento, são elas: *Leaflet* e MQTTJs. Essas bibliotecas são responsáveis pela criação e definição do mapa na página *web*, bem como suas marcações; como também pela conexão com o servidor *Broker* MQTT, a fim de obter os dados enviados, tanto pelo *tracker* como pelo *checker*.

Após tudo instalado, implementou-se o código que necessita de conhecimentos sobre programação orientada a objetos (POO); requisição de dados do servidor; laços/iterações para a resolução de problemas, e todo o padrão de lógica de *JavaScript*, *TypeScript* e as demais tecnologias citadas.

Devido ao código ser muito extenso e envolver tecnologias que não são totalmente voltadas para os objetivos gerais e específicos, não é necessário entender o código em si. No entanto, todo o código está disponível no *GitHub* (*link* nas referências) com acesso público, permitindo que qualquer pessoa possa baixá-lo e modificá-lo de acordo com suas necessidades.

A topologia então conta com três componentes principais, são eles:

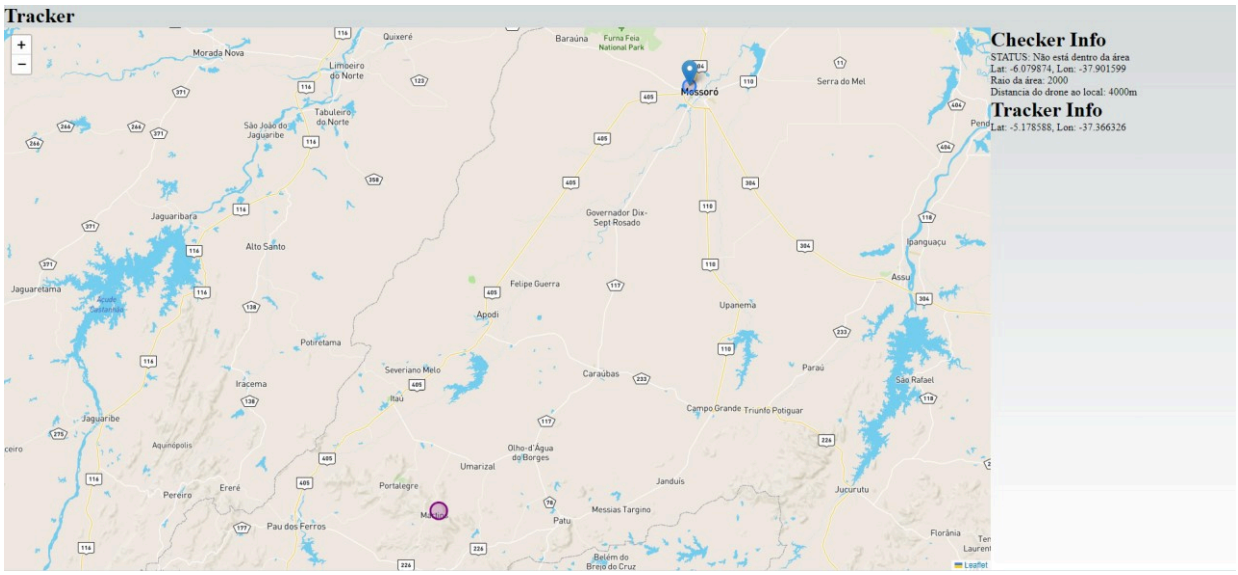
- *Index*
- *Map*
- *Information*

O componente *Index* é responsável por estabelecer a conexão com o servidor *Broker* MQTT, além de tratar os dados recebidos pelos *checker* e *tracker*, ele também deve enviar todas as informações aos outros dois componentes.

O componente *Map* exibe as informações recebidas através de marcadores no mapa criado, ou seja, além de mostrar a localização atual do drone, ele também traça a rota que já foi percorrida por ele. Já o componente *Information* exibe as informações através de texto.

A Figura 34 mostra um exemplo de como esses dados são exibidos em tela, ao lado esquerdo tem-se o mapa junto com uma marcação em azul que corresponde a localização atual de um drone, já a área em roxo, corresponde a área verificada por um *checker*. Já na direta, tem-se as informações principais acerca dos dois dispositivos.

Figura 34 – Exemplo de interface



Fonte: O autor.

6 RESULTADOS E DISCUSSÕES

Após a implementação de todas as topologias, buscou-se elaborar testes a fim de validar todo o desenvolvimento do projeto, para isso considerou-se então realizar três diferentes testes onde os parâmetros seriam iguais, mudando apenas a taxa de transmissão de dados do dispositivo rastreador (*tracker*).

Ou seja, depois de definir todos os parâmetros, mostrados na Tabela 3, realizou-se testes oscilando apenas o *delay* do rastreador.

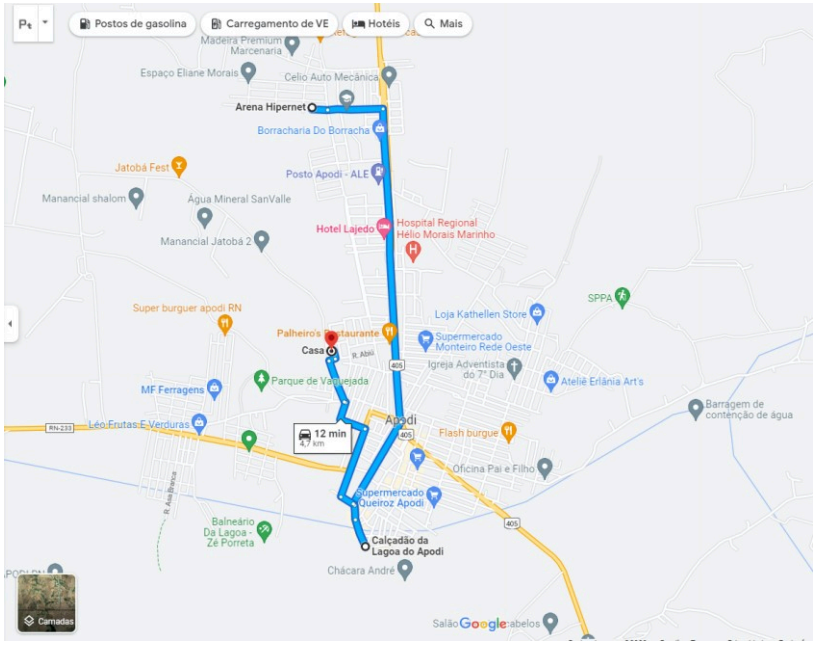
Tabela 4 – Parâmetros da topologia *checker*

Parâmetro	Valor
Latitude de referência do *checker*	-5.658349
Longitude de referência do *checker*	-37.802767
Raio da área do *checker*	50 m
Delay do *tracker*	5 min / 3 min / 1 min

Fonte: O autor.

Os resultados foram compilados e mostrados nos próximos tópicos. O percurso realizado foi o mesmo para todos os testes e consiste em aproximadamente 5km de distância. A Figura 35 ilustra o percurso de acordo com o *Google Maps*.

Figura 35 – Rota realizada para os testes

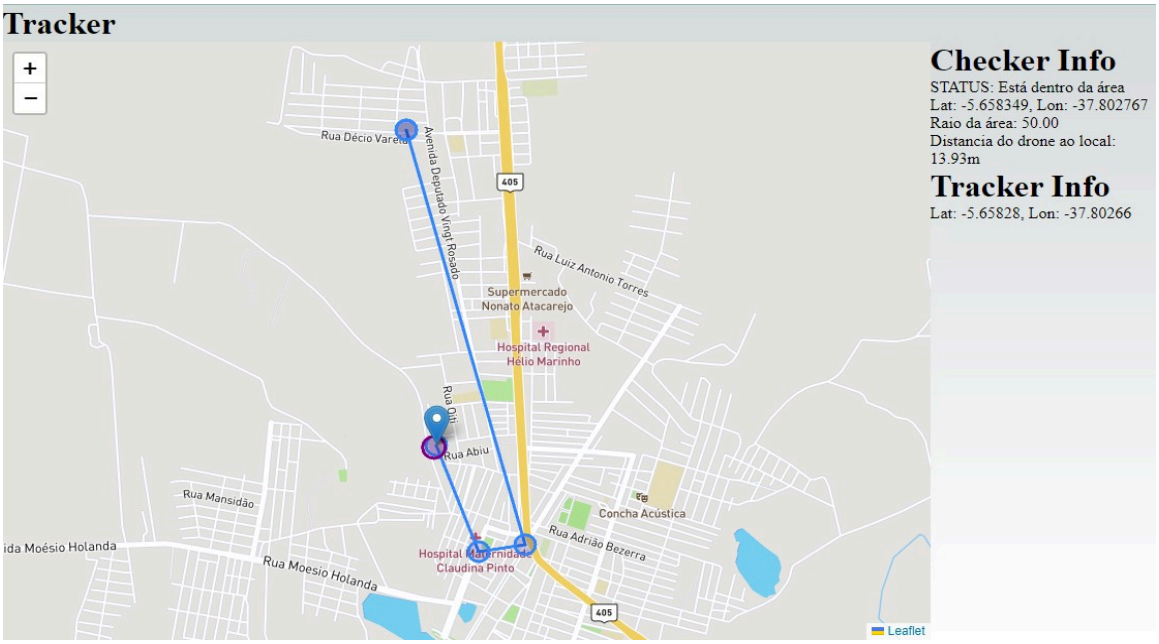


Fonte: *Google Maps*.

6.1 Teste para *delay* de 5 minutos

A figura abaixo ilustra os dados fornecidos pelo *tracker* realizado na rota definida anteriormente com um *delay* configurado em 5 minutos, ou seja, o *tracker* deve enviar sua localização a cada 5 minutos. Vale lembrar que o percurso foi feito de carro, e o tempo variou de acordo com o trânsito e outros empecilhos do cotidiano.

Figura 36 – Teste da aplicação para o *delay* de 5 minutos

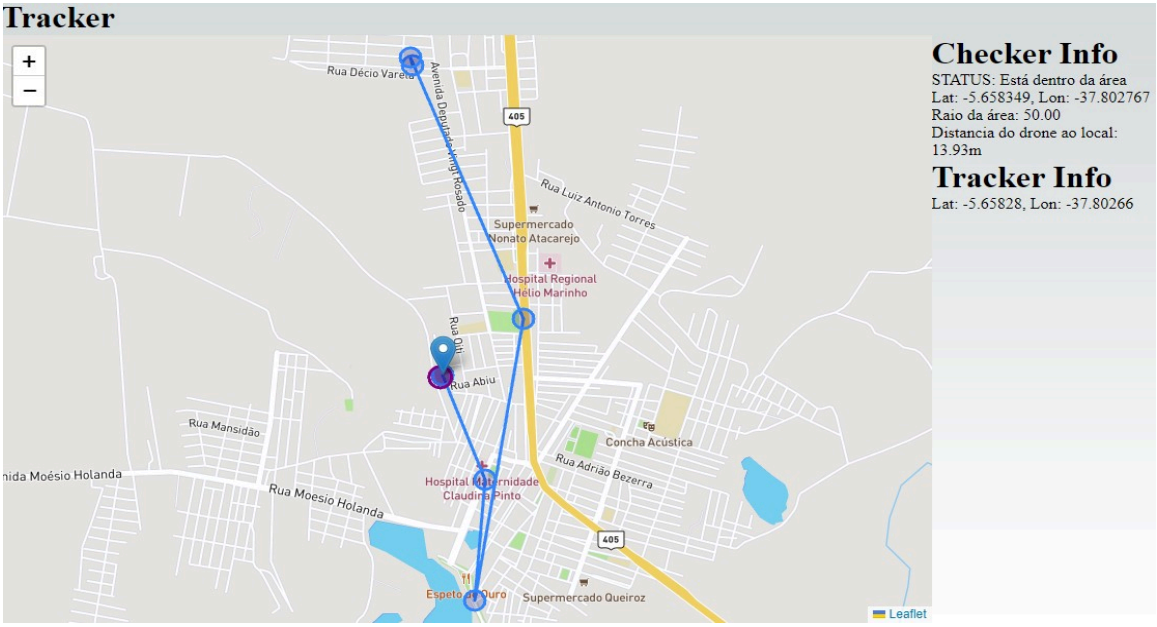


Fonte: O autor.

6.2 Teste para *delay* de 3 minutos

Outro teste realizado, dessa vez com o *delay* configurado para 3 minutos. O percurso foi o mesmo. Comparando com o teste anterior, nota-se que há 2 envios de dados a mais, porém, houve um retardo no início da rota.

Figura 37 – Teste da aplicação para o *delay* de 3 minutos

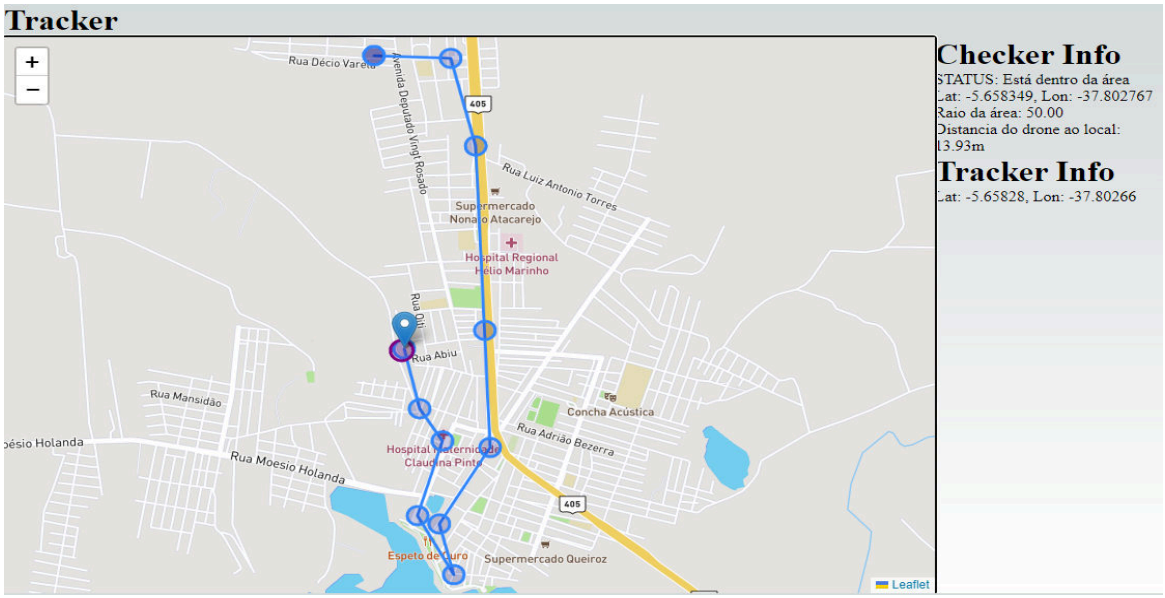


Fonte: O autor.

6.3 Teste para *delay* de 1 minuto

Já para o teste de 1 minuto, nota-se que há bem mais pontos que os demais, o que garante que o intervalo de envio de dados foi menor que os demais. Além disso, devido haver mais dados acerca da localização, nota-se que a rota mostrada se torna mais precisa quando comparada aos demais casos.

Figura 38 – Teste da aplicação para o *delay* de 1 minuto



Fonte: O autor.

6.4 Observações

Diante dos resultados obtidos e do sucesso tanto na implementação do projeto quanto nos testes realizados, algumas observações foram feitas durante o desenvolvimento que devem ser consideradas para possíveis alterações, dependendo da aplicação.

O primeiro ponto observado foi em relação ao parâmetro de *delay* do dispositivo acoplado ao drone. Esse parâmetro é extremamente importante e depende da funcionalidade do projeto ele deve ser definido como maior ou menor.

Após a análise dos testes, verificou-se que, para obter dados mais precisos o *delay* deve ser configurado o mais baixo possível. Em contrapartida, com *delays* mais baixos, há um aumento no uso de dados, pois mais dados serão enviados para o servidor, além disso, o processamento também é afetado, pois são mais instruções devem ser realizadas.

O tamanho máximo de um objeto é de 192 *bytes*, mas em ambiente de desenvolvimento, observou-se que os objetos enviados (seu padrão pode ser observado na Figura 9 da metodologia) tem um tamanho médio de 107 *bytes*. Porém, quando o dispositivo perde sua conexão com a *internet*, ele passa a armazenar esses objetos e somente quando houver a reconexão, deve enviar todos de uma só vez (Figura 10 da metodologia).

Essa situação acarreta em um descontrole de envio de dados, uma vez que passa a depender do número de objetos armazenados no *array* enquanto não havia conexão. Dessa forma, o novo tamanho de dados a ser enviado é representado pela equação abaixo:

$$array_{size} = (n \cdot 107) + 2$$

Onde:

$array_{size}$ é o tamanho total do array a ser enviado;

n corresponde ao número de objetos armazenados enquanto não há conexão.

Na equação há uma adição de 2 *bytes*, que corresponde a dois caracteres simbolizando colchetes, visto que, no formato *JSON* um *array* é representado por colchetes.

Com isso, recomenda-se que quando o *tracker* for percorrer maiores distancias, os *delays* sejam configurados com valores mais altos, isso implicará em uma menor precisão do percurso, mas irá contribuir na sustentabilidade do dispositivo, uma vez que, fará menos envios de dados pois há a diminuição de envios; e também irá diminuir o número de objetos inseridos no *array* quando não houver conexão.

Já em relação a topologia *checker,* observou-se que os parâmetros de latitude e longitude de referência, apesar de serem facilmente alterados, podem ser otimizados e alterados dependendo da sua aplicabilidade. Essa implementação consiste na adição de um módulo GPS acoplado ao seu microcontrolador. Essa mudança ocasionaria a remoção da parametrização do local de referência do *checker*, e tornaria a área de verificação móvel, e não mais estática.

Outro fator de grande impacto a ser comentado sobre o *checker* é que dependendo do serviço de automação, existe uma situação crítica que pode acarretar em grandes problemas pois fatores como a velocidade do drone não foram levados em conta.

Esse problema depende de 3 importantes variáveis, são elas: *delay* do *tracker*, velocidade do drone acoplado ao *tracker* e o raio de verificação do *checker*. Para facilitar a compreensão desse problema, ilustra-se a situação mais crítica onde o drone está com alta velocidade e fornece os dados de sua localização instantes antes de entrar na área do *checker*.

Dessa forma, ele terá que aguardar o tempo do *delay* configurado para só então sinalizar sua próxima localização. Isso acarreta em problemas de segurança e instabilidade na aplicação, uma vez que o drone não terá acesso ao local e deverá esperar o tempo do *delay* para sinalizar sua localização, e só então depois desse período, o *checker* irá verificar que o drone está dentro da área acionando o possível comando para abrir um portão.

Para a resolução desse problema, como não é levado em conta a velocidade do drone, poderia ser implementado a adição de áreas com raios diferentes, atuando como camadas, para o *checker.* Além disso, pode ser desenvolvido a comunicação entre ele e o *tracker*, a fim de alterar o tempo do *delay*, ou seja, tornar o *delay* de envio de dados dinâmico.

Descrevendo com outras palavras, quando o drone entrasse em uma área, receberia o comando de diminuir o *delay* para que o *checker* conseguisse acompanhar sua localização em menores intervalos, dessa forma, as áreas atuariam como camadas de controle do *delay* de envio de dados do dispositivo de rastreamento.

Por fim, a última observação consiste na topologia do *Frontend*, onde seria interessante uma possível adição de um formulário para que o administrador pudesse adicionar diferentes dispositivos, tornando a aplicação mais abrangente e aplicável em larga escala.

7 CONCLUSÕES

Em síntese, este trabalho buscou desenvolver um sistema de rastreamento por geolocalização de drones que utiliza a tecnologia da *internet* das coisas. Os objetivos específicos foram alcançados por meio da implementação de tecnologias de baixo custo, análise de parâmetros e realização de testes gerais e específicos.

Durante o processo de testes, algumas dificuldades foram encontradas, como a obtenção de dados sobre o consumo energético acerca da comunicação entre o microcontrolado e o módulo GPS. Apesar disso, os resultados obtidos contribuem para o avanço da tecnologia de rastreamento no setor de entregas, fornecendo informações importantes para projetos futuros.

Ademais, o uso de tecnologias inovadoras e de ótimo custo-benefício pode ser incentivado em diferentes setores, melhorando o transporte de produtos e promovendo o desenvolvimento tecnológico. Em vista disso, acredita-se que este trabalho possa contribuir para o avanço e aprimoramento da área de rastreamento por geolocalização de drones e, consequentemente, para a melhoria da logística de transporte de produtos.

Após resultados do trabalho, surge como opção o prosseguimento da aplicação, reavaliando e reajustando os pontos destacados no final do capitulo de resultados e discussões. O intuito é tornar a aplicação mais dinâmica, otimizada e confiável para que não apresente nenhuma instabilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

AL-FUQAHA, Ala et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials, v. 17, n. 4, p. 2347-2376, 2015.

ASHTON, K. That 'Internet of Things' Thing. In: RFID Journal, Jun. 2009.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. In: Computer Networks, v. 54, n. 15, pp. 2787-2805, Oct. 2010.

BANZI, Massimo; SHILOH, Michael. Getting Started with Arduino. O'Reilly Media, Inc., 2014.

BARTHOLOMEW, J.; FOX, P.; HOLDEN, D. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux. Sebastopol: O'Reilly Media, 2018.

CROCKFORD, D. JSON: The JavaScript Syntax Extension. Disponível em: https://www.json.org/json-en.html. Acesso em: 13 abr. 2023.

DatasheetHub. GY-NEO6MV2 Flight Control GPS Module Datasheet. [S.l.], [s.d.]. Disponível em: https://datasheethub.com/gy-neo6mv2-flight-control-gps-module/. Acesso em: 13 abr. 2023.

DELOITTE. Sustentabilidade na Cadeia de Suprimentos: uma visão estratégica para os negócios. [S.l.], 2021. Disponível em: https://www2.deloitte.com/content/dam/Deloitte/br/Documents/strategy/Deloitte-CPO-Survey-2021.pdf. Acesso em: 3 abr. 2023.

DOIT.AM. DOIT ESP32 DevKit v1. Disponível em: https://roboeq.ir/files/id/4034/name/ESP32%20MODULE.pdf/. Acesso em: 13 abr. 2023.

DORNELAS, F.; VIEIRA, L. Node.js: Desenvolvimento escalável de aplicações em JavaScript. Rio de Janeiro: Novatec, 2018.

ESPRESSIF SYSTEMS. ESP32 Datasheet. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 13 abr. 2023.

FACEBOOK. React. Disponível em: https://reactjs.org/. Acesso em: 13 abr. 2023.

FLANAGAN, David. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. 7. ed. O'Reilly Media, 2020.

GARCIA, M.; BROWN, R. Full-Stack Next.js. Sebastopol: O'Reilly Media, 2014.

GOOGLE MAPS. Mapa da UFERSA. Disponível em: https://www.google.com/maps/d/u/0/embed?mid=1nQgTpva4iHG0llCPVZhOyQs4-WhLnSg&ehbc=2E312F&ll=-5.771790071718634%2C-37.56938999999999&z=15. Acesso em: 19 abr. 2023.

GRAND VIEW RESEARCH. Supply Chain Analytics Market Size, Share, & Trends Report. San Francisco, 2021. Disponível em: https://www.grandviewresearch.com/industry-analysis/the-global-supply-chain-analytics-market. Acesso em: 3 abr. 2023.

GRANVILLE, Lisandro. MQTT - Protocolo para Internet das Coisas. GTA - Grupo de Teleinformática e Automação, 2019. Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/#. Acesso em: 17 abr. 2023.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of Things (IoT): A vision, architectural elements, and future directions. In: Future Generation Computer Systems, v. 29, n. 7, pp. 1645-1660, Sep. 2013.

HAO, Kun et al. A review on key technologies in internet of things. International Journal of Distributed Sensor Networks, v. 11, n. 6, p. 1-12, 2015.

HARBISON, J. Next.js: Building Server-Side-Rendered React Apps. Apress, 2019.

HUNKELER, U.; TRINKLER, P.; STERMER, A. MQTT-S — A publish/subscribe protocol for wireless sensor networks. In: 2008 Sixth IEEE International Conference on Pervasive Computing and Communications. IEEE, 2008. p. 1-6.

KAMAL, Raj. Microcontroladores: Fundamentos e Aplicações com PIC. 2. ed. São Paulo: Pearson Prentice Hall, 2014.

KOLBAN, N. Kolban's book on the ESP32 & ESP8266. Leanpub, 2018.
KREBS, M.; FABRY, T. TypeScript for Angular Developers: A Complete Guide. Berkeley: Apress, 2021.

LEAFLET. Leaflet - a JavaScript library for interactive maps. Disponível em: https://leafletjs.com/. Acesso em: 13 abr. 2023.

LEAL, W. J.; BORGES, L. C. Next.js: O guia completo. São Paulo: Casa do Código, 2021.

LIU, Jing et al. The research and implementation of a network-enabled interactive educational system based on the internet of things. International Journal of Online Engineering, v. 12, n. 5, p. 14-25, 2016.

LOVINE, M. Mastering TypeScript 3 - Third Edition: Build enterprise-ready, industrial-strength web applications using TypeScript 3 and modern frameworks. Birmingham: Packt Publishing, 2019.

MAZIDI, Muhammad Ali; NAIMI, Sarmad; NAIMI, Sepehr. The AVR microcontroller and embedded systems using Assembly and C for ATMEL. Prentice Hall, 2016.

MCCANDLESS, C. Leaflet.js essentials: Create stunning web maps with Leaflet.js. Packt Publishing, 2014.

MICROSOFT. TypeScript. Disponível em: https://www.typescriptlang.org/. Acesso em: 13 abr. 2023.

MQTT.JS. mqtt.js. Disponível em: https://github.com/mqttjs/MQTT.js. Acesso em: 13 abr. 2023.

NEOVA SOLUTIONS. Haversine vs Vincenty: Which is the Best? Neova Solutions, 2019. Disponível em: https://www.neovasolutions.com/2019/10/04/haversine-vs-vincenty-which-is-the-best/. Acesso em: 17 de abril de 2023.

NETO, Souza. Checker: código-fonte. 2023. Disponível em: https://github.com/souzaneto25/checker. Acesso em: 19 abr. 2023.

NETO, Souza. Tracker: código-fonte. 2023. Disponível em: https://github.com/souzaneto25/tracker. Acesso em: 19 abr. 2023.

NEXT.JS. What is Next.js?. Disponível em: https://nextjs.org/docs/getting-started. Acesso em: 13 abr. 2023.

O'REILLY, T.; LOUKIDES, M. What is Github? Disponível em: https://www.oreilly.com/library/view/what-is-github/9781449347504/. Acesso em: 13 abr. 2023.

O'SULLIVAN, J. MQTT Essentials: A Lightweight IoT Protocol. O'Reilly Media, 2017.

OLIVEIRA, L. R. C. et al. Internet das Coisas e suas tecnologias: Uma visão geral. In: Anais do V Simpósio de Sistemas Embarcados e Computação Ubíqua (Secomp 2018), 2018, Curitiba. Anais do V Simpósio de Sistemas Embarcados e Computação Ubíqua (Secomp 2018). Curitiba: SBC, 2018. p. 135-142.

OLIVEIRA, R. C. Next.js e React: O guia completo para iniciantes. São Paulo: Novatec, 2021.

PÉREZ, J. Mastering the ESP32. Packt Publishing, 2018.

PRADO, M. TypeScript: Desvendando a linguagem. São Paulo: Casa do Código, 2019.

SANTOS, L.; PINTO, G. Desenvolvimento de aplicações web com Node.js e React. In: Anais do Simpósio Brasileiro de Sistemas de Informação, 2020. Disponível em: http://sbsi.org.br/sbsi2020/artigos/281873.pdf. Acesso em: 13 abr. 2023.

SCHWARTZMAN, David. Programação para Arduino: Avançado. Novatec Editora, 2015.

SHIELDS, M. GitHub: What it is, how it works, and why it's a critical part of modern software development. Disponível em: https://www.techrepublic.com/article/github-what-it-is-how-it-works-and-why-its-a-critical-part-of-modern-software-development/. Acesso em: 13 abr. 2023.

SINGH, G.; SHARMA, S. MQTT: a lightweight IoT protocol for sensor networks. In: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, 2020. p. 1-6.

STANFORD-CLARK, A.; NIPPER, A. MQTT - Simply the best protocol for IoT. In: 3rd International Conference on Advanced Information Networking and Applications Workshops (AINAW'99), 1999, Washington. Proceedings of 3rd International Conference on Advanced Information Networking and Applications Workshops (AINAW'99). Washington: IEEE Computer Society, 1999. p. 456-466.

SYSTEMS, E. ESP32 Series Datasheet. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 13 abr. 2023.

THOMAS, C. M., FEATHERSTONE, W. E. Validation of Vincenty's Fórmulas for the Geodesic Using a New Fourth-Order Extension of Kivioja's Formula. Journal of Surveying Enginnering, p 20-26, fevereiro 2005.

TIOBE Index. TIOBE Index for April 2023. Disponível em: https://www.tiobe.com/tiobe-index/. Acesso em: 13 abr. 2023.

U-BLOX. NEO-6 Datasheet [GPS.G6-HW-09005]. Disponível em: https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf. Acesso em: 13 abr. 2023.

VINCENTY, T.: Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Surv. Rev., XXII (176), 88-93.1975.

W3TECHS. Usage statistics and market share of React for websites. Disponível em: https://w3techs.com/technologies/details/js-react/all/all. Acesso em: 13 abr. 2023.

WANG, Qun et al. Development of IoT-based intelligent transportation system. Journal of Network and Computer Applications, v. 60, p. 192-202, 2015.

ZEIT. Vercel - Develop. Preview. Ship. Disponível em: https://vercel.com/. Acesso em: 13 abr. 2023.

ZHOU, Kaidi et al. Cloud Manufacturing: A Computing and Service-Oriented Manufacturing Model. Computer Integrated Manufacturing Systems, v. 16, n. 1, p. 1-7, 2010.

APÊNDICE A – Código-Fonte referente ao TRACKER.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <TinyGPSPlus.h>
#include <SoftwareSerial.h>
#include <ArduinoJson.h>

#define RXD2 16                          // RXD2 do ESP32 conectado ao TX do GPS
#define TXD2 17                          // TXD2 do ESP32 conectado ao RX do GPS
#define GPS_BAUDRATE 9600                // Baud rate do GPS
#define WIFI_SSID "redewifi"             // SSID da rede WiFi
#define WIFI_PASSWORD "senhawifi"        // Senha da rede WiFi
#define MQTT_SERVER "broker.emqx.io"     // Endereço do servidor MQTT
#define MQTT_PORT 1883                   // Porta do servidor MQTT
#define MQTT_TOPIC_PREFIX "data/tracker" // Préfixo do tópico MQTT para publicar as
coordenadas
#define MQTT_USERNAME "emqx"             // Usuario do servidor
#define MQTT_PASSWORD "public"           // Senha do usuario
#define LED 2                            // Led interno do esp32
#define LED_EXT 15                       // Led externo ao esp32
#define DELAY 5000                       // Delay em ms para envio de dados.
Ex:60*1000=5seg

SoftwareSerial ss(RXD2, TXD2);            // Define a serial para comunicação com o
GPS
TinyGPSPlus gps;                          // Objeto para fazer o parsing dos dados do
GPS
WiFiClient wifiClient;                    // Cliente WiFi
PubSubClient client(wifiClient);          // Cliente MQTT
const int json = JSON_OBJECT_SIZE(10);    // Tamanho do objeto
StaticJsonDocument<json> doc;             // Criação do documento (objeto que vai ser
publicado)
size_t freeHeap = ESP.getFreeHeap();      // Quantidade de memória livre disponível
na heap, que é a área de memória dinâmica usada pelo programa durante a execução
DynamicJsonDocument doc2((freeHeap / 2)); // Criação de um documento com tamanho
máximo igual a metade da memoria disponivel no ESP32
JsonArray array = doc2.to<JsonArray>();   // Criação de um array de documentos

// Função responsável por conectar ao Wifi
void setup_wifi()
{
  delay(1000);

  Serial.println();
  Serial.print("Conectando ao wifi: ");
```

```
  Serial.println(WIFI_SSID);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  // Enquanto nao for estabelecida, fica aguardadando
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
    Serial.print(".");
  }

  // Conexao com wifi estabelecida. Aciona o led interno
  Serial.println("");
  Serial.println("WiFi conectado");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  digitalWrite(LED, HIGH);
}

// Delay customizado com verificação do gps
static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}

// Função de inicialização, após conectar ao wifi inicializa o servidor
void setup()
{
  Serial.begin(115200);
  ss.begin(GPS_BAUDRATE);
  pinMode(LED, OUTPUT);
  pinMode(LED_EXT, OUTPUT);
  setup_wifi();
  client.setServer(MQTT_SERVER, MQTT_PORT);
  Serial.print(freeHeap);
}

void loop()
{
  // Se os dados do GPS não estiverem disponíveis e válidos
  if (millis() > 5000 && gps.charsProcessed() < 10)
```

```
  {
    Serial.println("Esperando GPS...");
  }
  else
  {

    String client_id = "esp32-";
    client_id += String(WiFi.macAddress());
    String topic = String(MQTT_TOPIC_PREFIX) + "/" + client_id;
    client.connect(client_id.c_str(), MQTT_USERNAME, MQTT_PASSWORD);

    // se o cliente estiver conectado publica o
    if (client.connected() && gps.location.isValid() && gps.hdop.isValid() &&
gps.satellites.isValid())
    {
      // se ja tiver dados armazenados, envia
      String array_armazenado;
      serializeJson(doc2, array_armazenado);
      if (array_armazenado.length() > 2)
      {
        Serial.println("_______________Old array location_________________");
        Serial.print("array_armazenado length: ");
        Serial.println(array_armazenado);
        client.publish(topic.c_str(), array_armazenado.c_str());
        array = doc2.to<JsonArray>();
        Serial.println("_____________________________________________________");
      }
      Serial.println("..................New location...................");
      String new_location;
      doc.clear();
      doc["device"] = "esp32";
      doc["sensor"] = client_id;
      doc["coordinates"]["lat"] = String(gps.location.lat(), 6);
      doc["coordinates"]["lon"] = String(gps.location.lng(), 6);
      serializeJson(doc, new_location);
      digitalWrite(LED_EXT, HIGH);

      client.publish(topic.c_str(), new_location.c_str());

      delay(1000);
      digitalWrite(LED_EXT, LOW);
      Serial.println("Topic " + topic);
      Serial.println("Sats " + String(gps.satellites.value()));
      Serial.println("HDOP " + String(gps.hdop.hdop()));
      Serial.println(new_location);
      Serial.println("...............................................");
```

```
      client.disconnect();
    }
    else
    {
      Serial.println("..............Client disconnected.................");
      if (gps.location.isValid() && gps.hdop.isValid() && gps.satellites.isValid())
      {
        Serial.println("_______________New location saved_________________");
        JsonObject obj = array.createNestedObject();
        obj["device"] = "esp32";
        obj["sensor"] = client_id;
        obj["coordinates"]["lat"] = String(gps.location.lat(), 6);
        obj["coordinates"]["lon"] = String(gps.location.lng(), 6);
        Serial.println("{device:esp32, sensor:" + client_id + ", lat:" +
String(gps.location.lat(), 6) + ", lon:" + String(gps.location.lng(), 6) + "}");
        String output;
        serializeJson(doc2, output);
        Serial.print("Output length: ");
        Serial.println(output.length());
        Serial.println("_____________________________________________________");
      }
      else
      {
        Serial.println("..................Searching sats..................");
        Serial.println("{device:esp32, sensor:" + client_id + ", lat:" +
String(gps.location.lat(), 6) + ", lon:" + String(gps.location.lng(), 6) + "}");
      }
      Serial.println("..............................................");
    }
  }
  smartDelay(DELAY);
}
```

APÊNDICE B – Código-Fonte referente ao CHECKER.

```
#include <WiFi.h>
#include <cmath>
#include <ArduinoJson.h>

#define RXD2 16                          // RXD2 do ESP32 conectado ao TX do GPS
#define TXD2 17                          // TXD2 do ESP32 conectado ao RX do GPS
#define GPS_BAUDRATE 9600                // Baud rate do GPS
#define WIFI_SSID "redewifi"             // SSID da rede WiFi
#define WIFI_PASSWORD "senhawifi"        // Senha da rede WiFi
#define MQTT_SERVER "broker.emqx.io"     // Endereço do servidor MQTT
#define MQTT_PORT 1883                   // Porta do servidor MQTT
#define MQTT_TOPIC_PREFIX "data/checker" // Préfixo do tópico MQTT para publicar as
coordenadas
#define MQTT_USERNAME "emqx"             // Usuario do servidor
#define MQTT_PASSWORD "public"           // Senha do usuario
#define LED 2                            // Led interno do esp32
#define LED_EXT 15                       // Led externo ao esp32

const double LATITUDE_REFERENCE = -5.77179;                        // Latitude de
referência para calcular o distanciamento do dispositivo
const double LONGITUDE_REFERENCE = -37.56939;                      // Longitude de
referência para calcular o distanciamento do dispositivo
const double MAX_DISTANCE = 100;                                   // Distância
limite usada para disparar SMS de alerta de distanciamento, medida em metros
const String TOPIC_DRONE = "data/tracker/esp32-00:00:00:00:00:00"; // Tópico que
retorna a localização atual do drone

WiFiClient wifiClient;           // Cliente WiFi
PubSubClient client(wifiClient); // Cliente MQTT
StaticJsonDocument<192> doc;     // Criação do documento (objeto que vai ajudar a
calcular a distancia)
StaticJsonDocument<256> doc2;    // Criação do documento (objeto que vai ser
publicado)

// Função responsável por conectar ao Wifi
void setup_wifi()
{
  delay(1000);

  Serial.println();
  Serial.print("Conectando ao wifi: ");
  Serial.println(WIFI_SSID);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
```

```
  // Enquanto nao for estabelecida, fica aguardadando
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
    Serial.print(".");
  }

  // Conexao com wifi estabelecida. Aciona o led interno
  Serial.println("");
  Serial.println("WiFi conectado");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  digitalWrite(LED, HIGH);
}

// Função responsárvel por conectar ao MQTT e realizar o subscribe referente ao
tópico do drone
void connectToMqtt()
{
  while (!client.connected())
  {
    String client_id = "esp32-";
    client_id += String(WiFi.macAddress());
    Serial.println("Conectando ao broker MQTT...");
    if (client.connect(client_id.c_str(), MQTT_USERNAME, MQTT_PASSWORD))
    {
      Serial.println("Conectado ao broker MQTT!");
      if (client.subscribe(TOPIC_DRONE.c_str()))
      {
        Serial.println("Subscribed");
      }
    }
    else
    {
      Serial.print("Falha na conexão MQTT: ");
      Serial.println(client.state());
      delay(5000);
    }
  }
}

// Calcula a distância em que o dispositivo está, levando em consideração a curvatura
da terra (valor retornado em metros) formula de vicenty
double vincenty(double lat1, double lon1, double lat2, double lon2)
{
```

```
  const double a = 6378137;           // semieixo maior da Terra em metros
  const double b = 6356752.314245;    // semieixo menor da Terra em metros
  const double f = 1 / 298.257223563; // achatamento da Terra

  double L = (lon2 - lon1) * M_PI / 180;
  double U1 = atan((1 - f) * tan(lat1 * M_PI / 180));
  double U2 = atan((1 - f) * tan(lat2 * M_PI / 180));
  double sinU1 = sin(U1), cosU1 = cos(U1);
  double sinU2 = sin(U2), cosU2 = cos(U2);

  double lambda = L, lambdaP, sinLambda, cosLambda, sinSigma, cosSigma, sigma,
sinAlpha, cosSqAlpha, cos2SigmaM, C;
  int iterLimit = 100;
  do
  {
    sinLambda = sin(lambda), cosLambda = cos(lambda);
    sinSigma = sqrt((cosU2 * sinLambda) * (cosU2 * sinLambda) +
                    (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda) * (cosU1 * sinU2 -
sinU1 * cosU2 * cosLambda));
    if (sinSigma == 0)
      return 0; // pontos coincidentes
    cosSigma = sinU1 * sinU2 + cosU1 * cosU2 * cosLambda;
    sigma = atan2(sinSigma, cosSigma);
    sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma;
    cosSqAlpha = 1 - sinAlpha * sinAlpha;
    cos2SigmaM = cosSigma - 2 * sinU1 * sinU2 / cosSqAlpha;
    C = f / 16 * cosSqAlpha * (4 + f * (4 - 3 * cosSqAlpha));
    lambdaP = lambda;
    lambda = L + (1 - C) * f * sinAlpha *
                     (sigma + C * sinSigma * (cos2SigmaM + C * cosSigma * (-1 + 2 *
cos2SigmaM * cos2SigmaM)));
  } while (fabs(lambda - lambdaP) > 1e-12 && --iterLimit > 0);

  if (iterLimit == 0)
    return 0; // fórmula não convergiu

  double uSq = cosSqAlpha * (a * a - b * b) / (b * b);
  double A = 1 + uSq / 16384 * (4096 + uSq * (-768 + uSq * (320 - 175 * uSq)));
  double B = uSq / 1024 * (256 + uSq * (-128 + uSq * (74 - 47 * uSq)));
  double deltaSigma = B * sinSigma * (cos2SigmaM + B / 4 * (cosSigma * (-1 + 2 *
cos2SigmaM * cos2SigmaM) - B / 6 * cos2SigmaM * (-3 + 4 * sinSigma * sinSigma) * (-3
+ 4 * cos2SigmaM * cos2SigmaM)));
  double s = b * A * (sigma - deltaSigma);

  return s;
}
```

```
// Verifica se o dispositivo ultrapassou o limite de distância
bool deviceIsTooFar(float lat, float lon, String *distance)
{
  double dist = vincenty(lat, lon, LATITUDE_REFERENCE, LONGITUDE_REFERENCE);

  *distance = String(dist);

  if (dist > MAX_DISTANCE)
    return true;

  return false;
}

// será chamada sempre que houver uma mensagem MQTT recebida, conexão perdida ou
estabelecida, etc.
void callback(char *topic, byte *payload, unsigned int length)
{

  // dados recebidos do tracker
  deserializeJson(doc, payload);
  const char *sensor = doc["sensor"];
  const char *device = doc["device"];
  float latitude = doc["coordinates"]["lat"];
  float longitude = doc["coordinates"]["lon"];

  String distance;

  if (deviceIsTooFar(latitude, longitude, &distance))
  {

    if (client.connected())
    {
      String client_id = "esp32-";
      client_id += String(WiFi.macAddress());
      String topic = String(MQTT_TOPIC_PREFIX) + "/" + client_id;
      Serial.println("..................Device is far...................");
      String new_data;
      doc2.clear();
      doc2["device"] = "esp32";
      doc2["sensor"] = client_id;
      doc2["distance"] = String(distance);
      doc2["isClose"] = false;
      doc2["referenceLocation"]["lat"] = String(LATITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["lon"] = String(LONGITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["max_distance"] = String(MAX_DISTANCE);
```

```
      serializeJson(doc2, new_data);
      client.publish(topic.c_str(), new_data.c_str());
      Serial.println(new_data);
      Serial.println(topic);
      Serial.println(".............................................");
    }

    digitalWrite(LED_EXT, LOW); // apaga o led sinalizando que o dispositivo está
muito longe
    Serial.println("{Lat: " + String(latitude, 6) + "," + " Long: " +
String(longitude, 6) + "," + distance + "m }");
  }
  else
  {

    if (client.connected())
    {
      String client_id = "esp32-";
      client_id += String(WiFi.macAddress());
      String topic = String(MQTT_TOPIC_PREFIX) + "/" + client_id;
      Serial.println("..................Device is close...................");
      String new_data;
      doc2.clear();
      doc2["device"] = "esp32";
      doc2["sensor"] = client_id;
      doc2["distance"] = String(distance);
      doc2["isClose"] = true;
      doc2["referenceLocation"]["lat"] = String(LATITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["lon"] = String(LONGITUDE_REFERENCE, 6);
      doc2["referenceLocation"]["max_distance"] = String(MAX_DISTANCE);
      serializeJson(doc2, new_data);
      client.publish(topic.c_str(), new_data.c_str());
      Serial.println(new_data);
      Serial.println(topic);
      Serial.println(".............................................");
    }

    digitalWrite(LED_EXT, HIGH); // acende led sinalizando que o dispositivo está
muito perto
    Serial.println("{Lat: " + String(latitude, 6) + "," + " Long: " +
String(longitude, 6) + "," + distance + "m }");
  }
}

// Função de inicialização, após conectar ao wifi inicializa o servidor
void setup()
```

```
{
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  pinMode(LED_EXT, OUTPUT);
  setup_wifi();
  client.setServer(MQTT_SERVER, MQTT_PORT);
  client.setCallback(callback);

  connectToMqtt();
}

// Função de loop
void loop()
{
  // Mantenha a conexão MQTT ativa
  if (!client.connected())
  {
    connectToMqtt();
  }
  client.loop();
}
```