



TRABALHO DE GRADUAÇÃO

SISTEMA DE GEOLOCALIZAÇÃO PARA
ESTAÇÕES AMBIENTAIS MÓVEIS

Guilherme Silva Lemos

Brasília, Novembro de 2021



**ENGENHARIA
MECATRÔNICA**
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

SISTEMA DE GEOLOCALIZAÇÃO PARA ESTAÇÕES AMBIENTAIS MÓVEIS

Guilherme Silva Lemos

*Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação*

Banca Examinadora

Prof. Jones Yudi Mori Alves da Silva, _____
ENM/UnB
Orientador

Prof. Renato Coral Sampaio, FGA/UnB _____
Examinador interno

M.Sc. Jorge Andrade Seixas Maia, CrazyTech-
Labs _____
Examinador externo

Brasília, Novembro de 2021

FICHA CATALOGRÁFICA

GUILHERME SILVA LEMOS

Sistema de geolocalização para estações ambientais móveis

[Distrito Federal] 2021.

viii, 87p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Internet das Coisas

2.Estaçao ambiental

3. TTGO T-Beam

4.ThingsBoard

I. Mecatrônica/FT/UnB

II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

LEMOS, GUILHERME SILVA, (2021). Sistema de geolocalização para estações ambientais móveis. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº011, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 87p.

CESSÃO DE DIREITOS

AUTOR: Guilherme Silva Lemos

Sistema de geolocalização para estações ambientais móveis

GRAU: Engenheiro

ANO: 2021

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito do autor.

Guilherme Silva Lemos

Departamento de Eng. Mecânica (ENM) - FT

Universidade de Brasília (UnB)

CEP 70910-970, – Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho a todos que contribuíram para o sucesso da minha jornada acadêmica.

Guilherme Silva Lemos

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais pelo apoio incondicional ao longo da minha trajetória acadêmica. Sou grato também ao professor Jones pela compreensão e empatia ao longo do desenvolvimento deste projeto. Agradeço muito às amizades que criei na Universidade de Brasília, em especial ao Alexander, Antônio, Bruno, Carlos, Felipe, Gabriel B., Gabriel O., Gildo, Gustavo, Lorena, Luísa e Ricardo. Agradeço também à minha namorada Vívian. O carinho e apoio emocional fornecido por vocês foi imprescindível para o sucesso dessa trajetória.

Guilherme Silva Lemos

RESUMO

No atual contexto de mudanças climáticas e conforto urbano, é necessária uma capilarização da rede de monitoramento do clima para descrever melhor o contexto climático e oferecer dados de fomento ao planejamento de políticas públicas de urbanização, transporte, etc. Este projeto busca desenvolver um sistema que permita o monitoramento de variáveis climáticas, como temperatura e umidade, junto a dados de geolocalização, através de uma plataforma móvel de baixo custo em modelo *open source*, possibilitando a replicação e continuação do desenvolvimento. O sistema desenvolvido no presente trabalho é capaz de adquirir dados dos sensores disponíveis e transmiti-los, via tecnologia Lora, a um gateway remoto conectado à Internet. Utilizando o protocolo MQTT, um sistema em nuvem recebe, armazena e possibilita a visualização das séries temporais em um dashboard online.

Palavras Chave: Internet das Coisas, Estação ambiental, TTGO T-Beam, ThingsBoard

ABSTRACT

In the current context of climate change and urban comfort, it is necessary a capillarization of the climate monitoring network to better describe the climatic context and offer data to promote the planning of public policies for urbanization, transportation, etc. This project seeks to develop a system that allows the monitoring of climate variables, such as temperature and humidity, together with geolocation data, through a low-cost mobile platform in *open source* model, enabling replication and further development. The system developed in this work is able to acquire data from available sensors and transmit them, via Lora technology, to a remote gateway connected to the Internet. Using the MQTT protocol, a cloud system receives, stores and enables the visualization of the time series in an online dashboard.

Keywords: Internet of Things, Weather Station, TTGO T-Beam, ThingsBoard

SUMÁRIO

1	Introdução.....	1
1.1	CONTEXTUALIZAÇÃO.....	1
1.1.1	CLIMA URBANO	2
1.1.2	TECNOLOGIAS DE BAIXO CUSTO	4
1.1.3	INTERNET DAS COISAS	12
1.2	OBJETIVOS	13
1.3	ESTRUTURA DO DOCUMENTO	14
2	Fundamentos	15
2.1	SISTEMAS DE GEOLOCALIZAÇÃO.....	15
2.1.1	GPS.....	16
2.1.2	GLONASS.....	17
2.2	PLATAFORMA IoT	18
2.2.1	DETALHES E FUNCIONAMENTO DA PLATAFORMA THINGSBOARD	21
2.3	COMUNICAÇÃO.....	24
2.3.1	COMUNICAÇÃO ENTRE OS NÓS DE MEDIDAÇÃO E O GATEWAY	25
2.3.2	ASPECTOS REGULATÓRIOS.....	26
2.3.3	COMUNICAÇÃO ENTRE O GATEWAY E A PLATAFORMA IoT.....	27
3	Desenvolvimento.....	28
3.1	ESCOLHAS DE HARDWARE	28
3.2	IMPLEMENTAÇÃO DOS ALGORITMOS NA PLACA TTGO T-BEAM	30
3.2.1	UNIDADE DE SENSORIAMENTO.....	31
3.2.2	GATEWAY.....	34
3.3	IMPLEMENTAÇÃO DA PLATAFORMA IoT	36
3.3.1	CONFIGURAÇÃO DO AMAZON EC2	37
3.3.2	INSTALAÇÃO DA THINGSBOARD.....	41
3.3.3	CONFIGURAÇÃO DA THINGSBOARD	44
3.4	PROBLEMAS E DIFICULDADES.....	50
3.4.1	SINAL GPS FRACO.....	50
3.4.2	CI DE GESTÃO DE ENERGIA AXP192	50
3.4.3	INCOMPATIBILIDADE DA TELA OLED COM O MÓDULO AXP192	51
3.4.4	AFERIÇÃO DA TENSÃO DA BATERIA 18650	51

4 Testes e resultados.....	52
4.1 TESTES NA UNIDADE DE SENSORIAMENTO	52
4.2 TESTES COM O GATEWAY.....	53
4.3 TESTES COM A THINGSBOARD	55
5 Conclusões.....	61
5.1 LOCALIZAÇÃO	61
5.2 COMUNICAÇÃO.....	61
5.3 VISUALIZAÇÃO DE DADOS	62
5.4 ALGORITMOS IMPLEMENTADOS.....	62
5.5 PERSPECTIVAS FUTURAS.....	62
5.5.1 MELHORIAS NA LOCALIZAÇÃO	62
5.5.2 MELHORIAS NA COMUNICAÇÃO ENTRE AS UNIDADES DE SENSORIAMENTO E o GATEWAY	63
5.5.3 MELHORIAS NA VISUALIZAÇÃO DE DADOS	63
5.5.4 OTIMIZAÇÃO DO PROJETO	63
5.5.5 INTEGRAÇÃO COM O PROJETO MOCHILA BIOCLIMÁTICA.....	63
REFERÊNCIAS BIBLIOGRÁFICAS	65
Anexos.....	68
I Códigos utilizados	69
I.1 ALGORITMO DA UNIDADE DE SENSORIAMENTO.....	69
I.1.1 UTILITIES.H - CONJUNTO DE PARÂMETROS.....	69
I.1.2 BOARDS.H - INICIALIZAÇÃO DA PLACA	70
I.1.3 ALGORITMO PRINCIPAL DA UNIDADE DE SENSORIAMENTO	72
I.2 ALGORITMO DO GATEWAY.....	77
I.2.1 UTILITIES.H - CONJUNTO DE PARÂMETROS.....	77
I.2.2 BOARDS.H - INICIALIZAÇÃO DA PLACA	78
I.2.3 ALGORITMO PRINCIPAL DO GATEWAY	80

LISTA DE FIGURAS

1.1	Diferença de temperatura em uma área urbana que sofre com a Ilha de Calor Urbana	3
1.2	Modelo WS-2902C. Fabricado pela <i>Ambient Weather</i>	5
1.3	Modelo Vantage Vue. Fabricado pela <i>Davis Instruments</i>	7
1.4	Protótipo meteorolog	8
1.5	Dados comparativos de umidade relativa do ar - Projeto Meteorolog e CEPSRM	9
1.6	Estação meteorológica de baixo custo (EMBC) construída por Moura na Universidade Federal de Uberlândia	10
1.7	Comparativo da direção do vento registrado na EMBC e na ECA (INMET) Fonte: Moura (2018)	10
1.8	Comparativo da velocidade do vento registrado na EMBC e na ECA (INMET) Fonte: Moura (2018).....	11
1.9	Projeto Mochila Bioclimática construída na Universidade de Brasília (UnB)	11
1.10	Blocos e princípios básicos sinérgicos da IoT	12
1.11	Esquema de unidades e comunicações do projeto.	13
2.1	Gráfico Azimute x Elevação comparando as órbitas dos satélites entre os sistemas...	16
2.2	Configuração da constelação de satélites do GPS em 3 de Janeiro de 2008. Cada círculo preto representa um satélite e cada letra (de A a F) representa um plano orbital.....	17
2.3	O serviço disponibilizado pela Amazon tem capacidade de usar dados de localização.	18
2.4	Exemplo demonstrativo de um rastreador GPS junto ao microcontrolador ESP32 na plataforma AskSensors.	19
2.5	Exemplo demonstrativo de um rastreador GPS junto à placa LinkIt ONE na plataforma ThingsBoard.	20
2.6	Esquemático da plataforma ThingsBoard e fluxos de dados disponíveis.	22
2.7	Arquitetura monolítica do ThingsBoard.	23
2.8	Processamento de mensagens de telemetria através do serviço de Rule Engine.	24
2.9	Comparação entre diferentes de tecnologias de comunicação sem fio.....	25
3.1	Placa LILYGO TTGO T-Beam Versão 1.0.	28
3.2	Display OLED 128x64 0.96 polegadas.....	30
3.3	Baterias de íons de lítio no modelo 18650.	30
3.4	Mapeamento dos pinos na placa TTGO T-Beam.....	31
3.5	Funcionamento simplificado do código nas unidades de sensoriamento.....	33

3.6	Montagem da unidade de sensoriamento.	34
3.7	Funcionamento simplificado do Gateway.	35
3.8	Montagem do Gateway.	36
3.9	Criação do par de chaves na AWS.	37
3.10	Seleção do tipo de instância no Amazon EC2.	38
3.11	Seleção do armazenamento.	39
3.12	Regras de tráfego de rede.	40
3.13	Instância criada.	40
3.14	Instância Ubuntu Server em execução no Amazon EC2.	41
3.15	Tela de login após a instalação da ThingsBoard.	43
3.16	Dashboard demonstrativo já incluído na edição de comunidade.	44
3.17	Criação de um novo dispositivo.	45
3.18	Acesso à criação de um novo Alias (marcado pelo círculo vermelho).	46
3.19	Configuração do Alias a ser utilizado.	47
3.20	Configuração da fonte de dados no widget.	47
3.21	Acesso aos estados do dashboard.	48
3.22	Estado para visualização detalhada das métricas de temperatura e umidade.	49
3.23	Configuração da ação para acessar o estado.	49
3.24	Antena GPS do módulo avulso (esquerda) e antena GPS da TTGO T-Beam (direita).	50
4.1	Dados de temperatura, umidade e localização obtidos pela unidade de sensoriamento.	53
4.2	Visualização dos dados enviados pela unidade de sensoriamento no Gateway.	54
4.3	Verificação da comunicação entre o Gateway e a plataforma ThingsBoard.	55
4.4	Unidade de sensoriamento percorrendo o Campus Darcy Ribeiro.	56
4.5	Gráficos de temperatura e umidade ao longo do tempo.	57
4.6	Unidade de sensoriamento percorrendo a Entrequadra Norte 313/314.	58
4.7	Teste com duas unidades de sensoriamento.	59
4.8	Teste com duas unidades de sensoriamento em movimento.	60

LISTA DE TABELAS

1.1	Parâmetros de calibração da estação WS2902C	6
1.2	Acurácia e resolução das medidas	6
1.3	Acurácia, resolução e intervalo de atualização das medidas aferidas na estação Davis Vantage Vue[1].....	7
2.1	Comparação entre os sistemas GPS e GLONASS[2].....	16
2.2	Comparação entre diferentes tecnologias de comunicação sem fio[3]	26

LISTA DE SÍMBOLOS

Siglas

IBGE	Instituto Brasileiro de Geografia e Estatística
ICU	Ilha de Calor Urbana
OMM	Organização Mundial de Meteorologia
LCD	<i>Liquid-crystal display</i>
UFRGS	Universidade Federal do Rio Grande do Sul
CEPSRM	Centro Estadual de Pesquisas em Sensoriamento Remoto e Meteorologia
EMBC	Estação meteorológica de baixo custo
ECA	Estação Climatológica Auxiliar
INMET	Instituto Nacional de Meteorologia
IoT	<i>Internet of Things</i>
GNSS	<i>Global Navigation Satellite System</i>
GPS	<i>Global Positioning System</i>
GLONASS	<i>GLObal NAVigation Satellite System</i>
AWS	<i>Amazon Web Services</i>
MQTT	<i>MQ Telemetry Transport</i>
HTTPS	<i>Hyper Text Transfer Protocol Secure</i>
IDE	<i>Integrated Development Environment</i>
GCP	<i>Google Cloud Platform</i>
RFID	<i>Radio and frequency identification</i>
LoRa	<i>Long Range</i>
LPWA	<i>Low Power Wide Area</i>
ANATEL	Agência Nacional de Telecomunicações
CoAP	<i>Constrained Application Protocol</i>
LwM2M	<i>Lightweight Machine to Machine</i>
SDK	<i>Software Development Kit</i>
GPIO	<i>General Purpose Input and Output</i>
PMIC	<i>Power Management Integrated Circuit</i>
USB	<i>Universal Serial Bus</i>
OLED	<i>Organic Light-Emitting Diode</i>

JSON	<i>JavaScript Object Notation</i>
FreeRTOS	<i>Free Real Time Operating System</i>
EC2	<i>Elastic Compute Cloud</i>
SSH	<i>Secure Shell</i>
JDK	<i>Java Development Kit</i>
I ² C	<i>Inter-Integrated Circuit</i>
TTFF	<i>Time to first fix</i>
CRC	<i>Cyclic Redundancy Check</i>

Capítulo 1

Introdução

Este capítulo traz a motivação e o contexto por trás do projeto. Há uma contextualização do clima urbano e sua importância para o nosso dia-a-dia. Em seguida, é introduzido o conceito de tecnologias de baixo custo e sua recente popularização. Por fim, é detalhado o conceito de Internet das Coisas e como o tópico se intersecciona com o monitoramento climático.

1.1 Contextualização

Com o crescimento urbano e a formação de grandes metrópoles nas últimas décadas, cada vez é mais necessário o estudo e o monitoramento do clima em escalas menores e mais específicas, como os microclimas. Estudos realizados nos centros urbanos têm mostrado que as cidades possuem um clima próprio, resultado de interferências humanas, como pavimentação de ruas, remoção da vegetação, alta concentração de pessoas e a emissão de partículas tóxicas ao meio ambiente. Esse clima próprio, denominado clima urbano, geralmente é afetado com o aumento da temperatura e a diminuição da umidade relativa do ar quando comparado com regiões rurais limítrofes com o perímetro urbano. [4, 5]

O fornecimento de dados distribuídos e de baixo custo facilita a capilarização do monitoramento climático, trazendo dados a níveis mais microscópicos, facilitando tomada de decisões sobre políticas públicas e inovações climáticas. O interesse nesses dados é notável já que são necessários para pesquisas sobre o bem estar da sociedade e consequentemente a preservação do meio ambiente. [6]

A tendência mundial de optar pelo uso de eletrônicos para captação, transmissão e armazenamento de dados meteorológicos permite que a capilarização ocorra de forma ainda mais acelerada. A tecnologia por trás desses eletrônicos tornou-se viável devido à popularização de microcontroladores e sensores de temperatura e umidade, por exemplo. [6]

Dessa forma, o escopo do projeto é a construção de uma infraestrutura de baixo custo e *open source* que permitirá o envio de métricas ambientais (temperatura e umidade, por exemplo) junto

a dados de localização. Além disso, é o intuito do projeto que ele seja replicável e expansível sem grandes dificuldades. O projeto deverá mandar os dados de localização e os parâmetros monitorados para um banco de dados remoto. Sua alimentação será feita por baterias recarregáveis, o que permitirá sua mobilidade.

1.1.1 Clima Urbano

A criação de cidades e crescente ampliação de áreas urbanas (zonas metropolitanas) têm contribuído para o crescimento de impactos ambientais. Mudanças físicas e biológicas no ambiente modificam a paisagem e comprometem ecossistemas ao longo do tempo. Essas mudanças ocorrem por inúmeros motivos, sejam naturais ou a partir de intervenções antropológicas. O desenvolvimento tecnológico e global tem contribuído para que essas mudanças se intensifiquem, especialmente no ambiente urbano. [7]

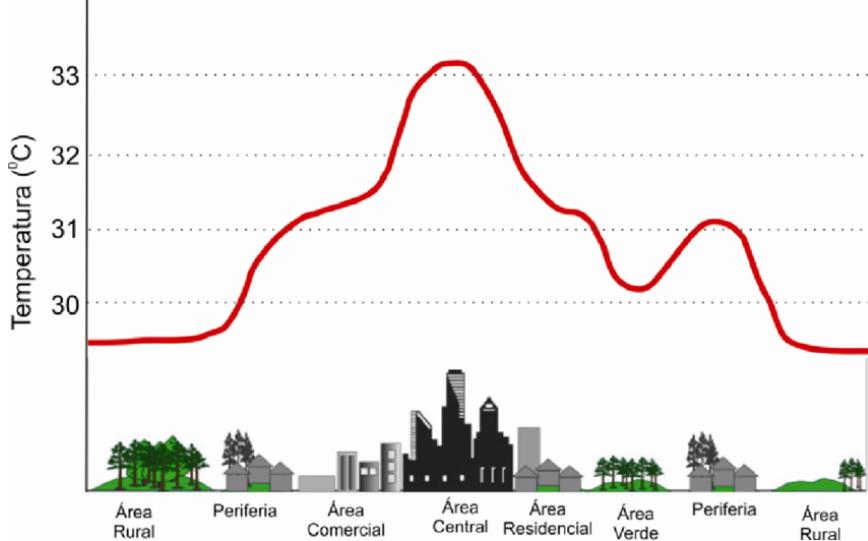
Dados apresentados pelo Instituto Brasileiro de Geografia e Estatística (IBGE) (2015) mostram que no Brasil cerca de 85% da população reside em áreas urbanas. A partir de meados do século XX, a acelerada urbanização e crescimento das cidades promoveram mudanças climáticas e ambientais no planeta, de forma mais significante do que qualquer outra atividade humana. Essa transformação de um país rural para um país urbano ocorreu segundo um processo desigual e privilegiado, com a marginalização de classes sociais da população, por não terem condições de aquisição, em zonas que deveriam ser protegidas para preservação de nascentes, encostas e reservas ambientais. [7]

O morador da cidade deseja, acima de tudo, viver em um ambiente que apresente melhores condições de vida, independente de sua classe social. Ar puro, água potável, níveis controlados de ruídos e outras questões são pontos básicos para se ter uma vida de qualidade. Contudo, percebe-se atualmente que é necessária uma mudança na forma de expansão e criação de centros urbanos, já que a forma tradicional que foi e ainda é feita trouxe grandes impactos ambientais no ecossistema urbano.

A diferença das condições climáticas entre um centro urbano e uma zona rural é evidente e facilmente perceptível. A temperatura no centro urbano é mais alta, a umidade é mais baixa, há uma concentração maior de partículas prejudiciais à saúde em suspensão no ar. É possível imaginar através de analogias os motivos porque isso acontece: Edifícios bloqueiam/dificultam a circulação de ar, ruas pavimentadas diminuem a absorção de água pelo solo e aumentam o albedo (quantidade de irradiação que é refletida pelo solo) e o grande número de pessoas aumenta o fluxo de veículos que emitem poluentes e calor. Em contra partida também é possível pensar em motivos pelo qual a zona rural geralmente possui um clima mais ameno em relação ao centro urbano, como grandes áreas de faixa verde para absorver o calor irradiado.

Nesse contexto, surge o conceito de Ilha de Calor Urbana (ICU). Essas ilhas são fenômenos associados ao clima urbano, na qual uma determinada área urbana pode apresentar condições microclimáticas diferentes em relação às áreas rurais adjacentes. [5] O fenômeno pode ser observado na figura 1.1.

Figura 1.1: Diferença de temperatura em uma área urbana que sofre com a Ilha de Calor Urbana.



Fonte: Azevedo et al. [8].

A importância de estudar e coletar dados sobre as ICUs está no entendimento dos efeitos decorrentes desse fenômeno. Os efeitos negativos afetam diretamente o cotidiano nas cidades, trazendo uma baixa qualidade do ar e um aumento na temperatura, fatores que são responsáveis por um número considerável de óbitos no mundo. Outro fator importante que é impactado diretamente pelas ICUs é a eficiência energética da cidade, já que há um aumento no consumo de energia elétrica para amenizar a situação climática com equipamentos de ar condicionado ou ventiladores, por exemplo.

O problema do fenômeno das ICUs é intensificado devido a outro fenômeno natural, a inversão térmica. No caso geral, ao longo da noite, o solo perde muito calor por meio de irradiação para a atmosfera imediatamente acima dele, graças a diferença de temperatura do solo quente aquecido ao longo do dia e a atmosfera fria, com menor capacidade térmica. Essa troca de calor gera uma movimentação nas massas de ar, diminuindo a sensação térmica e formando ventos. Contudo, na presença de uma ICU, o ar tende a manter uma temperatura similar à temperatura do solo, reduzindo as trocas de calor e consequentemente, a formação de ventos. Isso contribui para uma estagnação da massa de ar nos centros urbanos, concentrando ainda mais poluentes na atmosfera da cidade. [9]

Com o intuito de identificar as ICUs e descrever melhor a condição climática, é necessário o monitoramento das variáveis climáticas essenciais, chamadas de *Essential Climate Variables* pela Organização Mundial de Meteorologia (OMM). A OMM descreve uma variável climática essencial como uma variável física, química ou biológica, ou ainda um grupo de variáveis interligadas que contribuem de forma crítica para a caracterização do clima terrestre [10]. Algumas variáveis que a OMM considera como essenciais são a pressão atmosférica, quantidade de precipitação, índice de radiação solar, velocidade e direção do vento, umidade e composição química atmosférica. Entretanto, existem outros parâmetros auxiliares que descrevem melhor o conforto e a situação ambiental da cidade, como o nível de ruído e a luminosidade. Dentre os parâmetros citados, os

que serão monitorados nesse projeto são:

- Temperatura;
- Umidade;
- Latitude;
- Longitude;
- Data e hora do envio dos dados (*timestamp*).

1.1.2 Tecnologias de Baixo Custo

A popularização de plataformas de desenvolvimento em torno de microcontroladores e microprocessadores tem ocorrido através de uma recente tendência de mercado, o baixo custo e o fácil acesso a essas tecnologias. Devido à essa grande popularização e à vasta diversidade de projetos e ferramentas que podem ser construídas, o monitoramento ambiental urbano pode ser feita de forma mais distribuída e independente. A possibilidade de gerar dados climáticos a partir de plataformas de baixo custo pode facilitar a descrição do clima atual e futuro. A partir dessa acessibilidade, foi possível descentralizar o conhecimento e os dados sobre o clima. Não somente grandes universidades e centros de pesquisa têm capacidade para descrever o clima atual, indivíduos podem desenvolver suas próprias soluções que atualmente conseguem ter uma precisão satisfatória e sem o custo de equipamentos profissionais. [11]

Desde meados da primeira década do século XXI tem surgido um conceito de sensoriamento coletivo e participativo, onde cada indivíduo tem atuado como atuador direto na malha de monitoramento da realidade, seja utilizando a tecnologia para realizar a transmissão ou para realizar a própria observação da informação.[11, 12] Essa participação coletiva na modalidade de sensoriamento pode ocorrer por diferentes formas, sejam por percepções pessoais e subjetivas , por exemplo, um formulário de sensação térmica, ou por captações objetivas através de sensores adquiridos e configurados pelos próprios indivíduos. Aproveita-se a partir dessa situação a mobilidade e a distribuição dos indivíduos para contornar a necessidade de grandes investimentos na malha de coleta, quando o objetivo é expandir a cobertura espacial, mesmo que o seu objetivo não seja o monitoramento ambiental/meteorológico.

Com a popularização de materiais e equipamentos no mundo da eletrônica e mais especificamente na telemetria, vem o questionamento sobre a confiabilidade dos dados gerados por esses componentes de baixo custo. Afinal não há vantagem alguma nessa disseminação de tecnologia querendo investigar um determinado fenômeno se os dados gerados não possuem significância real. [11]

1.1.2.1 Opções Comerciais de Estações

No contexto de qualidade de dados, é importante observar as opções comerciais já existentes no mercado. Apesar de muitas vezes não terem um preço tão alto quanto o esperado de um

equipamento profissional, a disponibilidade e facilidade de acesso ao produto em países como o Brasil é limitada. Outro ponto importante a ser destacado nessas soluções de mercado é a manutenção a longa data, já que geralmente após algum tempo de mercado, o fabricante deseja atualizar sua revisão de hardware, e consequentemente, sua revisão de software. Muitas vezes os hardwares antigos não recebem atualizações equivalentes de software, deixando-os defasados e com bugs que não serão corrigidos. Existem iniciativas de softwares *open source*, como por exemplo, o WeeWX[13], que visam dar atualizações e até mesmo novos recursos para esses hardwares.

Um exemplo de estação climática comercial é o modelo WS-2902C fabricado pela *Ambient Weather*, o qual está representada na figura 1.2. A estação permite monitorar a velocidade e a direção do vento, índices pluviométricos, radiações solares, pressão atmosférica, umidade relativa do ar, ponto de orvalho, entre outros parâmetros. A estação é acompanhada de um display LCD para demonstrar os dados de forma eficiente e assertiva. Não é possível trocar as aferições para o sistema métrico de unidades. O fabricante especifica que o período de atualização dos parâmetros é de 16 segundos.

Figura 1.2: Modelo WS-2902C. Fabricado pela *Ambient Weather*.



Fonte: Ambient Weather [14].

O manual do equipamento [14] especifica que a estação permite a calibração individual dos parâmetros a fim de se ajustar os sensores para terem uma precisão melhor. Os intervalos de calibração são limitados. Além disso, o fabricante sugere a fonte de calibração, ou seja, o sensor de referência que normalmente é utilizado para calibrar um parâmetro específico. As informações estão dispostas na tabela 1.1:

Tabela 1.1: Parâmetros de calibração da estação WS2902C[14].

Parâmetro	Intervalo de ajuste	Tipo	Instrumento de calibração
Temperatura	$\pm 9^{\circ}\text{F}$	Offset	Termômetro de mercúrio
Umidade	$\pm 9\%$	Offset	Psicômetro
Pressão absoluta	$\pm 10 \text{ hpa}$	Offset	Barômetro de laboratório calibrado
Pressão relativa	-	Offset	Estações meteorológicas de referência
Direção do vento	$\pm 180^{\circ}$	Offset	GPS, bússola de precisão
Velocidade do vento	0.5 a 1.5	Ganho	Anemômetro calibrado
Precipitação	0.5 a 1.5	Ganho	Pluviômetro analógico

Outros dados importantes presentes no manual são relativos à acurácia de cada parâmetro. Esse é o ponto principal quando se discute sobre a confiabilidade de um sensor, já que está associado ao erro aleatório e portanto, não é possível compensar esse tipo de erro. Além disso, a resolução de cada sensor também é uma informação importante, pois vai identificar a menor unidade possível a ser medida naquele parâmetro.

A tabela 1.2 descreve a acurácia e a resolução das medidas:

Tabela 1.2: Acurácia e resolução das medidas[14].

Parâmetro	Acurácia	Resolução
Temperatura	2°F	0.1°F
Umidade	5%	1%
Pressão atmosférica	0.08 inHg	0.01 inHg
Direção do vento	10°	1°
Velocidade do vento	2.2 mph ou 10% (o maior entre os dois)	1.4mph
Luminosidade	15%	1 Lux
Precipitação	5%	0,01 in

O ponto final a ser descrito sobre essa estação comercial é o seu custo. Como não há revenda oficial desse modelo no Brasil, o preço de referência foi coletado na página oficial do fabricante. Em novembro de 2021, o modelo custava US\$ 169,99, no site do fabricante. No mesmo site, também é possível obter uma estimativa de preço em reais, junto a taxas de importação. O preço em reais era de aproximadamente R\$ 1915¹.

Outro modelo existente no mercado é o *Vantage Vue Wireless Weather Station* da fabricante *Davis Instruments*, da figura 1.3. Uma observação imediata é o suporte do produto tanto para o sistema imperial quanto o sistema métrico de unidades. De forma similar ao modelo da *Ambient Weather*, este modelo vem acompanhado de uma tela LCD com as medidas principais dispostas de forma fácil e objetiva de serem observadas. Todos os parâmetros observados pela estação anterior

¹Ambient Weather WS-2902C <https://ambientweather.com/amws2902.html>

estão também cobertos por esse modelo. Algumas informações extras são mostradas no display, como os horários do por e nascer do sol.

Figura 1.3: Modelo Vantage Vue. Fabricado pela *Davis Instruments*.



Fonte: Davis Instruments [1].

O manual especifica que a central (sem o display) tem um consumo de $30mA$ (pico) / $0.2mA$ a $3.3VDC$. Com a bateria recomendada, a estação pode funcionar por 8 meses sem a necessidade de luz solar. Com a presença da luz solar, a estação pode-se recarregar através de seu painel fotovoltaico, com capacidade de gerar até $0.5W$. Seu funcionamento pode durar mais de 2 anos nesse caso.[1]

Também é especificado que essa estação vem pré-calibrada, ou seja, não há a necessidade de usar instrumentos de referências para garantir a precisão das medidas fornecidas. A página do modelo também especifica que tem um intervalo de atualização das medidas a cada $2.5s$, porém após uma inspeção do manual, foi possível observar que o valor de $2.5s$ se refere apenas à medida que tem o maior intervalo de atualização (direção e velocidade do vento), mas os outros valores são atualizados com intervalos diferentes [1]. A tabela a seguir traz a acurácia, resolução e intervalo de atualização de cada medida (não inclui medidas calculadas utilizando dados coletados por sensores):

Tabela 1.3: Acurácia, resolução e intervalo de atualização das medidas aferidas na estação Davis Vantage Vue[1].

Parâmetro	Acurácia	Resolução	Taxa de Atualização
Temperatura	0,5°C acima de -7°C, 1°C abaixo de -7°C	0,1°C	1 minuto
Umidade	2%	1%	1 minuto
Pressão atmosférica	0,8 mmHg	0,1 mmHg	1 minuto
Direção do vento	3°	1°	2,5 a 3 segundos
Velocidade do vento	1 m/s ou 5% (o maior entre os dois)	0,5m/s	2,5 a 3 segundos
Precipitação	5%	0,2mm	20 a 24 segundos
Horário	8 segundos/mês	1 minuto	1 minuto

Percebe-se através da leitura dos manuais e das páginas que esse modelo tem uma abordagem mais profissional e menos de baixo custo. O produto não necessita de calibrações e tem tolerâncias menores quando comparado ao modelo da *Ambient Weather*. Essas qualidades são refletidas diretamente em seu preço, a US\$ 425 no site oficial do fabricante¹ (aproximadamente R\$ 2356 sem custos de transporte e importação). Como no caso anterior, a estação não está disponível oficialmente no Brasil. Ademais, a oferta em sites varejistas é escassa, dificultando uma especificação em território brasileiro.

A partir dos dois modelos comerciais vistos nessa seção, pode-se ter uma ideia de parâmetros de projetos comumente utilizados nesse tipo de aplicação, e principalmente relacioná-los com o custo do produto. Também é importante notar que apesar da possibilidade de uma construção móvel, em torno de um tripé portátil, por exemplo, nenhum desses modelos comerciais visa o monitoramento de parâmetros ao longo de um trajeto.

1.1.2.2 Estações meteorológicas de baixo custo construídas anteriormente

Outras soluções de estações meteorológicas de baixo custo foram propostas anteriormente. Soluções desenvolvidas por Silva et al (2015) [15], por Moura (2018) [16] e por Pazos et al (2020) [17] trouxeram como microcontrolador e processador das informações recebidas pelos sensores a plataforma *Arduino*. Devido ao fato de ser uma plataforma *open source*, diversos recursos e bibliotecas estão disponíveis na internet. Isso facilita a interação da plataforma com sensores de baixo custo disponíveis no mercado, já que não será necessário, na maioria dos casos, desenvolver um driver que possibilita a interpretação correta dos dados a partir do microcontrolador.

Figura 1.4: Protótipo meteorolog.

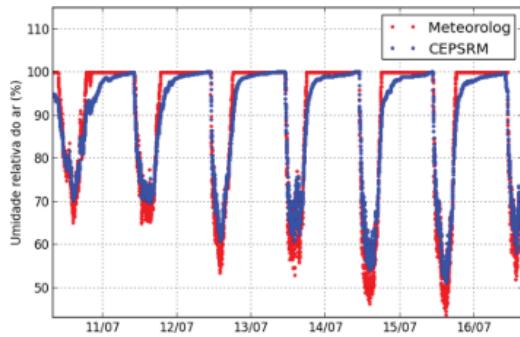


Fonte: Silva et al.[15].

¹Davis Vantage Vue Weather Station
<https://www.davisinstruments.com/collections/vantage-vue/products/vantage-vue-wireless-weather-station>

A estação da figura 1.4 coleta apenas dados referentes a quatro sensores: Umidade, temperatura, pressão atmosférica e luminosidade, sendo essa última através de um resistor sensível a luz, o que significa que a escala desse dado foi arbitrária. Os dados estatísticos apresentados pelos autores indicam que os sensores de temperatura e pressão atmosférica apresentaram erros de calibração, sendo fácil seu reparo através de ajustes com base em dados de referência. Já o sensor de umidade apresentou um problema relacionado ao tempo de resposta em situações de umidades próximas a 100%, mais intrínseco à construção do sensor.[15] Os dados referentes ao sensor de umidade da estação Meteorolog estão dispostos abaixo juntamente aos dados de umidade da estação CEPSRM, a qual foi calibrada e utilizada pela comunidade científica na UFRGS:

Figura 1.5: Dados comparativos de umidade relativa do ar - Projeto Meteorolog e CEPSRM.



Fonte: Silva et al.[15].

A discrepância dos dados de umidade foi corroborada com um alto desvio padrão da diferença entre a medida de referência e a da estação de baixo custo. Segundo os autores, a diferença no tempo de resposta vem principalmente a partir da proteção dos sensores de intempéries. O sensor utilizado na estação de referência possui uma proteção porosa que permite uma condensação mais rápida da umidade relativa do ar, acelerando o seu tempo de resposta.[15]

Outra abordagem de estação meteorológica de baixo custo foi a produzida por Moura na Universidade Federal de Uberlândia (UFU) [16]. Essa estação, chamada de EMBC (Estação meteorológica de baixo custo) conseguiu captar os dados relacionados a pressão atmosférica, temperatura, umidade, velocidade e direção do vento e quantidade de chuva. Além disso, a EMBC mantém os dados armazenados localmente e pode-se comunicar através da internet. Outro detalhe importante dessa implementação é a sua fonte de alimentação. O projeto utiliza um painel solar que gera 10 Watts por hora, além de uma bateria selada de 6V 24Ah para armazenar a energia gerada pelo painel e alimentar a estação ao longo do dia. Apesar da EMBC possuir comunicação com a internet, os dados não são enviados para um banco de dados remoto, sendo assim são apenas enviados para um *endpoint* instantaneamente [16]. A figura 1.6 representa a estação construída:

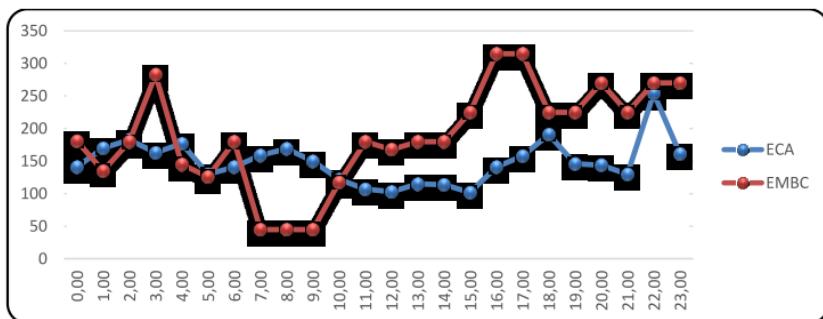
Figura 1.6: Estação meteorológica de baixo custo (EMBC) construída por Moura na Universidade Federal de Uberlândia.



Fonte: Moura [16].

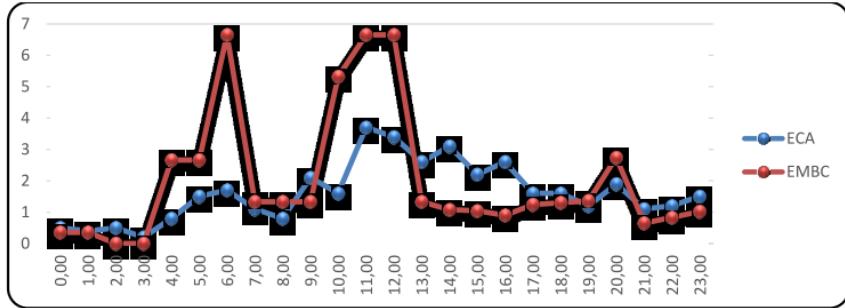
Para verificar a precisão e acurácia dos dados, a estação foi instalada no mesmo local da ECA (Estação Climatológica Auxiliar) do INMET (Instituto Nacional de Meteorologia). Os dados de temperatura e umidade obtidos na EMBC obtiveram cerca de 92% de acurácia após a calibração dos sensores através do cálculo de ponto de orvalho, tendo assim uma precisão aceitável para um dispositivo de baixo custo. Os dados de precipitação, apesar da baixa amostragem de ocorrências de precipitações, também seguiram uma acurácia aproximada de 92%. É importante destacar que o pluviômetro não tem calibração linear a partir dos dados, por ser um sensor analógico. O sensor de pressão atmosférica também apresentou uma fidelidade grande aos dados de referência, principalmente após o auto ajuste do sensor. A pressão atmosférica máxima alcançou 100% de acurácia, enquanto a mínima apresentou 99,9% de acurácia. Já os dados relacionados aos ventos tiveram grande discrepância apresentada. O autor relata sobre a diferença dos comportamentos dos ventos em diferentes altitudes, já que a EMBC foi instalada próximo ao solo (sendo atingida por ventos de superfície) e o anemômetro da ECA-INMET está instalada a uma altura aproximada de 8 metros, apresentando ventos que não necessariamente se comportam da mesma forma ao nível do solo.[16] Essa discrepância está demonstrada através das Figuras 1.7 e 1.8:

Figura 1.7: Comparativo da direção do vento registrado na EMBC e na ECA (INMET).



Fonte: Moura [16].

Figura 1.8: Comparativo da velocidade do vento registrado na EMBC e na ECA (INMET).



Fonte: Moura [16].

Há de se destacar que ao contrário da estação Meteorolog (Figura: 1.4), na EMBC os sensores de temperatura, umidade e pressão atmosférica foram enclausurados em um abrigo cerâmico com aletas que permitem a passagem de ar e os protegem de intempéries. Isso corrobora com a discrepância nos dados de umidade encontrada na Meteorolog 1.5 e que não foi tão relevante na EMBC.

A última abordagem a ser relatada neste trabalho foi feita na Universidade de Brasília (UnB) por Romero et al (2020)[17]. Essa abordagem teve como objetivo a construção de uma estação meteorológica móvel e portátil, chamada de Mochila bioclimática pelos autores do projeto. Como o projeto foi mais focado no monitoramento microclimático em trajetos do campus da UnB, seu escopo foi reduzido para monitorar apenas a pressão atmosférica, temperatura e umidade. A alimentação da estação foi feita através de uma bateria selada de 12V a 2,3Ah que era recarregada manualmente quando necessário[17]. Na figura 1.9 está demonstrado o projeto em uso:

Figura 1.9: Projeto Mochila Bioclimática construída na Universidade de Brasília (UnB).



Fonte: Romero et al. [17].

Apesar do projeto apresentar mobilidade, não era possível associar os dados a uma coordenada exata. Dessa forma, a utilização do projeto durante longos trajetos prejudica a associação dos dados ambientais a um local específico. Além disso, os dados ficavam gravados apenas localmente, sem o envio para um banco de dados remoto, por exemplo.

A importância de relatos anteriores dos projetos acima está na dificuldade que foi encontrada em cada ocasião, ajudando na previsão de problemas que podem ser encontrados ao longo da execução deste projeto.

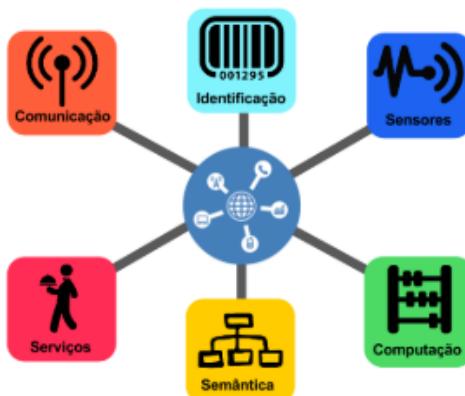
O projeto deste trabalho visa em melhorar e estender a funcionalidade do projeto Mochila Bioclimática[17], principalmente nos quesitos de comunicação, localização e expansão do projeto, sem trazer grandes custos e de forma que seja facilmente replicável.

1.1.3 Internet das Coisas

A Internet das Coisas (do termo em inglês *Internet of Things (IoT)*) surgiu do progresso de grandes áreas da engenharia como sensoriamento e telemetria, microeletrônica, sistemas embarcados e sistemas de comunicação. Seu potencial de uso é vasto e vem sendo amplamente investido, tanto no âmbito acadêmico quanto no meio industrial. [18]

A internet das coisas têm possibilitado a extensão da internet tradicional, conectando objetos que possuem capacidade computacional e que podem se comunicar. Esses objetos no passado não eram comumente conectados a rede[18]. Um exemplo comum no nosso dia-a-dia é o crescente uso de relógios inteligentes que podem monitorar sinais vitais, registrando-os localmente e comunicando com um servidor remoto que processa os dados de forma que fique atrativo para quem usa o produto. A conexão com a rede mundial de computadores possibilita não só o monitoramento de dados, mas também o controle desses dados. Na figura 1.10, é possível observar os diferentes tópicos que são abordados em conjunto em IoT.

Figura 1.10: Blocos e princípios básicos sinérgicos da IoT.



Fonte: Santos et al.[18].

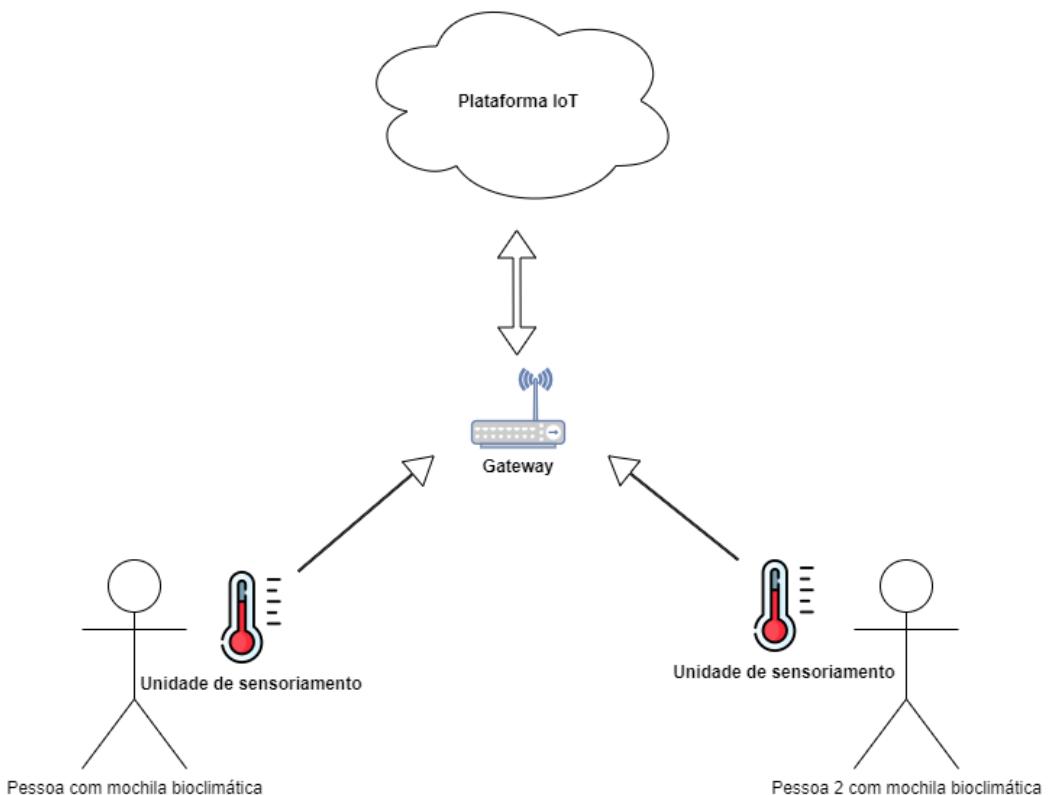
Ainda no contexto de Internet das Coisas, além da coleta e processamento de dados, é de

extrema importância a visualização desses dados de forma que eles fiquem interpretáveis e comprehensíveis, inclusive para pessoas que não necessariamente estão no meio de Internet das Coisas possam compreender os dados processados de forma a facilitar decisões estratégicas e táticas.

O projeto da mochila bioclimática e este trabalho vão de encontro ao tema de Internet das coisas já que foi justamente a popularização e disseminação da IoT que possibilitou a construção de um projeto complexo a custos baixos. Tradicionalmente estações de monitoramento climático são exclusivas de universidades, centros meteorológicos e grandes empresas, mas com a popularização da IoT, os dados foram descentralizados e democratizados a todos, além de todo o conhecimento por trás da construção e desenvolvimento.

Na figura 1.11 é possível ver o diagrama esquemático do projeto, facilitando o seu contexto dentro do tópico de internet das coisas.

Figura 1.11: Esquema de unidades e comunicações do projeto.



Fonte: Figura do Autor.

1.2 Objetivos

O objetivo do projeto é a construção de uma plataforma que possibilite a comunicação, localização, expansão e visualização dos dados coletados pelo projeto Mochila Bioclimática[17].

Etapas do projeto:

- Escolha de plataforma e módulos a serem utilizados no projeto;
- Teste de funcionamento do módulo de geolocalização;
- Teste da comunicação entre módulos;
- Envio e coleta de métricas ambientais junto a dados de geolocalização;
- Envio dos dados coletados para um *endpoint* remoto;
- Visualização dos dados.

Não está dentro do escopo do projeto a calibração dos sensores utilizados no projeto Mochila Bioclimática, devido à falta de acesso aos componentes utilizados no projeto. Ainda assim, o projeto visa permitir a calibração por meio da plataforma IoT a ser utilizada.

1.3 Estrutura do Documento

A estrutura do texto subsequente está disposta da seguinte forma:

- **Capítulo 2:** Fundamentos - São discorridas possíveis tecnologias que podem ser utilizadas para que os objetivos do projeto possam ser alcançados;
- **Capítulo 3:** Desenvolvimento - É discutida a forma como as tecnologias escolhidas foram implementadas ao longo do trabalho. Também são listadas as dificuldades encontradas e como elas podem ser resolvidas;
- **Capítulo 4:** Testes e resultados - São analisados os dados gerados pela implementação. Ademais, a metodologia dos testes é descrita, além dos seus locais de execução;
- **Capítulo 5:** Conclusões - Os resultados obtidos com os testes são criticados e sugestões futuras de melhoria e continuidade do projeto são feitas;

Capítulo 2

Fundamentos

Este capítulo explora as principais possibilidades e opções para alcançar os objetivos do trabalho. São analisadas diversas soluções possíveis para cada problema e elas são comparadas com referências bibliográficas e demais implementações.

2.1 Sistemas de geolocalização

Tem sido cada vez maior o interesse em processar o posicionamento de objetos e veículos com base em um referencial conhecido e com alta acurácia (ao nível centimétrico). Nesse contexto, o sistema de navegação global por satélite - GNSS (*Global Navigation Satellite System*) é composto por soluções diversas que atuam isoladamente e recentemente vêm sendo trabalhadas em conjunto para aumento da precisão da localização. Alguns sistemas são mais tradicionais e consolidados como o sistema americano GPS (*Global Positioning System*) e o sistema russo GLONASS (*Global'naya Navigatsionnaya Sputnikovaya Sistema*), enquanto surgem outras iniciativas mais recentes como o GALILEO, desenvolvido pela União Europeia e o BeiDou/Compass, desenvolvido pela China. Esses sistemas dependem de conexão com pelo menos quatro satélites para estimar uma posição global.[19].

Atualmente, dois desses sistemas estão operando em sua totalidade e são de uso livre para a população mundial, o GPS e o GLONASS. O sistema GALILEO e o BeiDou ainda não estão completamente operacionais e não estão amplamente disponíveis como as opções anteriores. Tendo em vista que o projeto atual visa ser de acesso fácil, amplo e barato, foram desconsiderados os dois últimos sistemas.

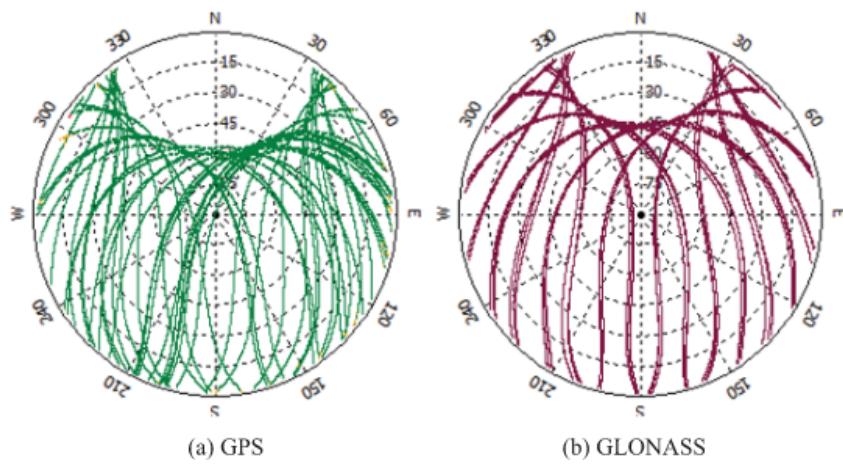
Na tabela a seguir, pode-se comparar algumas características entre o GPS e o GLONASS:

Tabela 2.1: Comparação entre os sistemas GPS e GLONASS[2].

	GPS	GLONASS
Nº de satélites	31	24
Nº de satélites nominais	24	24
Nº de planos orbitais	6	3
Plano de inclinação	55°	65°
Altitude (km)	20.180	19.100

Na figura 2.1 é possível comparar as trajetórias orbitais dos satélites em cada um dos sistemas:

Figura 2.1: Gráfico Azimute x Elevação comparando as órbitas dos satélites entre os sistemas.



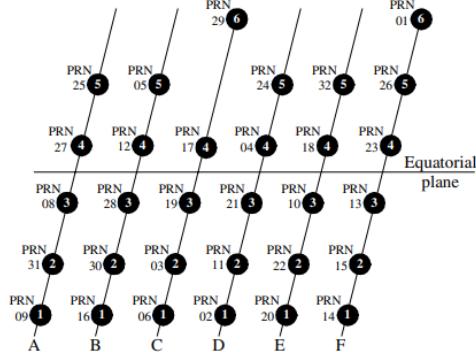
Fonte: Li et al.[20].

Pela figura 2.1 é possível observar que o GLONASS possui uma cobertura maior em latitudes mais afastadas da linha do equador, porém possui uma cobertura menos frequente em relação ao GPS.

2.1.1 GPS

O GPS - *Global Positioning System*, desenvolvido e projetado pelo Departamento de Defesa dos Estados Unidos, vem sido utilizado há mais de 20 anos e atualmente está em fase de operação e modernização. Oficialmente o projeto nasceu em 1973 a partir da decisão da criação de um sistema de satélites baseados em diversos projetos militares já existentes da época.[21] Na configuração padrão, a constelação de satélites do GPS consiste em 24 satélites ativos, com mais 3 disponíveis em casos de falha. Essa constelação está distribuída em 6 planos orbitais, inclinados a aproximadamente 55° em relação a linha do equador.[21]. Na figura 2.2 é possível observar a forma como esses planos estão distribuídos em relação à linha do equador:

Figura 2.2: Configuração da constelação de satélites do GPS em 3 de Janeiro de 2008. Cada círculo preto representa um satélite e cada letra (de A a F) representa um plano orbital.



Fonte: Cojocaru et al.[21].

Essa é a configuração mínima para garantir uma cobertura global, tendo pelo menos 4 satélites visíveis acima da linha do horizonte a partir de qualquer ponto a céu aberto na Terra.

As órbitas dos satélites GPS podem ser descritas como elipses com uma excentricidade baixa. O semieixo maior da elipse orbital é de aproximadamente 26.560 km, implicando em um período orbital (tempo necessário para completar uma volta em torno do globo terrestre) de 12 horas e uma altitude de aproximadamente 20.180 km.

2.1.2 GLONASS

O GLONASS (*GLObal NAVigation Satellite System*) é o sistema que foi inicialmente desenvolvido pela antiga União Soviética e que atualmente é responsabilidade do governo russo. Tendo um passado similar ao do GPS, ele também teve propósitos iniciais militares que foram posteriormente expandidos para o uso civil.

Apesar do início do projeto ter sido em 1972 e do sistema estar operacional já em 1995, a constelação dos satélites GLONASS passou por um longo período de degradação devido à falta de investimentos e baixa vida útil dos primeiros satélites colocados em órbita. Essa baixa disponibilidade do serviço durante esse longo período afetou o uso e pesquisa do sistema, o qual passou por períodos em que apenas seis a oito satélites estavam operando nominalmente. A partir de 2001, foi operado uma reforma do sistema para retomar a sua operação global novamente após o hiato de funcionamento.[22].

O GLONASS é constituído de 24 satélites operacionais e 3 de reserva, distribuídos em três planos orbitais com separações de 120° e com inclinação aproximada de $64,8^\circ$ em relação ao equador, o que permite uma cobertura maior em altas latitudes. A altitude média dos satélites é de 19.100km e o período orbital de aproximadamente 11h15m. Com a normalidade do sistema, 6 a 8 satélites devem estar visíveis a qualquer momento a partir de qualquer ponto da Terra.

No geral, tanto o GPS quanto o GLONASS tem plena capacidade de atender aos pré requisitos do projeto atual. Porém, dado o maior desenvolvimento do GPS devido à sua ampla disponibili-

lidade desde o seu lançamento[22], a variedade de módulos de baixo custo é significativamente mais alta para o GPS, além do amplo suporte oferecido pela comunidade de código aberto. Sendo assim, o GPS foi o escolhido para fornecer o serviço de localização do projeto.

2.2 Plataforma IoT

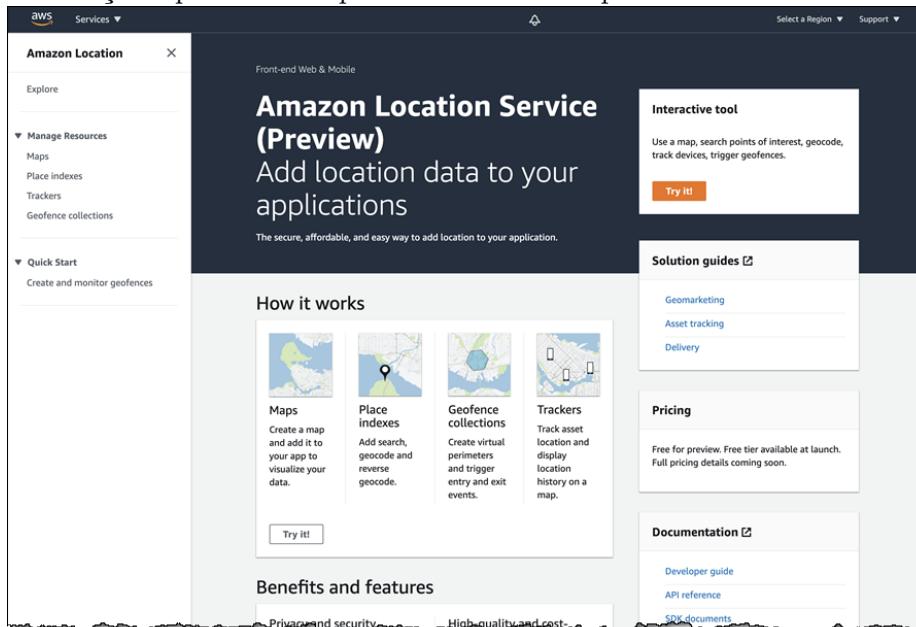
Com o crescimento exponencial da internet das coisas, naturalmente vem sido desenvolvidas soluções e softwares que facilitem a criação de soluções nesse contexto. As plataformas IoT funcionam como um *endpoint* (um receptor) dos dados coletados pelos dispositivos de sensoriamento que são reencaminhados pelo Gateway.

O papel principal dessas plataformas é fornecer uma interface que facilite o processamento, visualização e armazenamento dos dados enviados.

Existem diversas soluções comerciais mais consolidadas nesse ambiente de plataformas IoT, como a AWS IoT¹ (demonstrada na figura 2.3), fornecida pela Amazon, a Azure IoT², fornecida pela Microsoft, e por fim a IBM Wattson³.

É necessário que a plataforma IoT escolhida seja capaz de reproduzir e construir gráficos utilizando mapas, com o propósito de acompanhar a localização de cada ponto de coleta de dados ao longo do transecto percorrido.

Figura 2.3: O serviço disponibilizado pela Amazon tem capacidade de usar dados de localização.



Fonte: Amazon Web Services.

Apesar da disponibilidade dessas plataformas desenvolvidas por empresas mais consolidadas

¹ Amazon Web Services IoT <https://aws.amazon.com/pt/iot/>

² Microsoft Azure IoT <https://azure.microsoft.com/pt-br/overview/iot/>

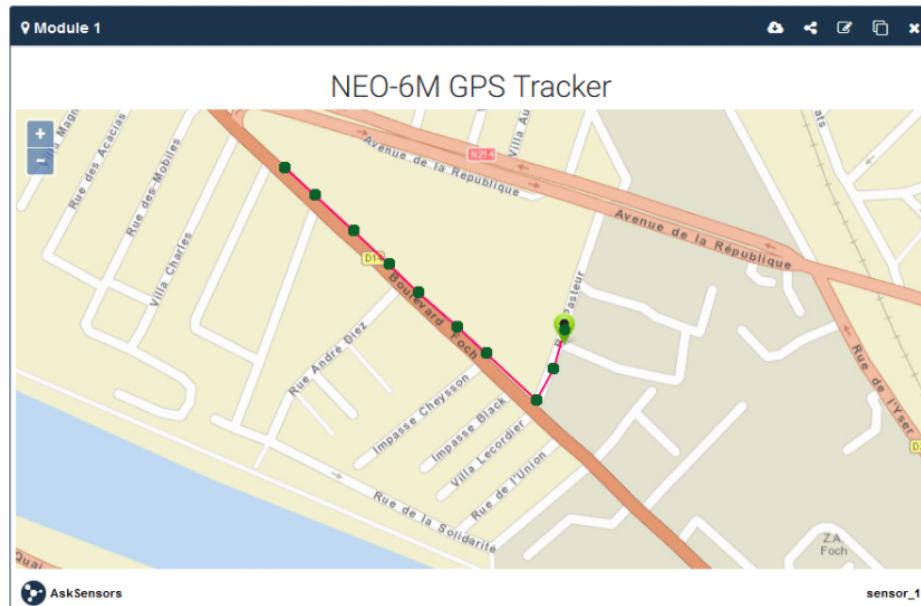
³ IBM Wattson IoT <https://internetofthings.ibmcloud.com/>

no mercado de tecnologia, outras plataformas estão disponíveis com a proposta de serem mais simples e intuitivas, sem o aprofundamento em cada plataforma de *cloud computing* que são oferecidas por essas empresas, trazendo o objetivo de acelerar o desenvolvimento de aplicações IoT, principalmente em projetos como o atual que visam provar um conceito.

Por isso, serão discutidas as plataformas AskSensors[23] e ThingsBoard[24]. Ambas plataformas ajudam a atingir o objetivo do projeto, podendo fornecer dashboards interativos capazes de processar e demonstrar dados de geolocalização. Ambas as plataformas suportam uma grande variedade de microcontroladores, como Arduino, ESP8266, ESP32, Raspberry Pi, além de suportarem o envio desses dados através de diversos protocolos de comunicação, como MQTT (*MQ Telemetry Transport*) e HTTPS (*Hyper Text Transfer Protocol Secure*). Essa vasta variedade de opções em dispositivos e protocolos facilita a busca por um microcontrolador/*hardware* compatível. Ambas plataformas também disponibilizam bibliotecas compatíveis com o Arduino IDE, ambiente de desenvolvimento de código livre que também é suportado por inúmeras placas e microcontroladores disponíveis no mercado.

As duas plataformas inclusive disponibilizam exemplos demonstrativos que visam facilitar e guiar o início do projeto nas ferramentas. Abaixo, na figura 2.4, pode-se observar a representação de um trajeto capturado por um módulo GPS conectado a um microcontrolador ESP32 na plataforma AskSensors:

Figura 2.4: Exemplo demonstrativo de um rastreador GPS junto ao microcontrolador ESP32 na plataforma AskSensors.

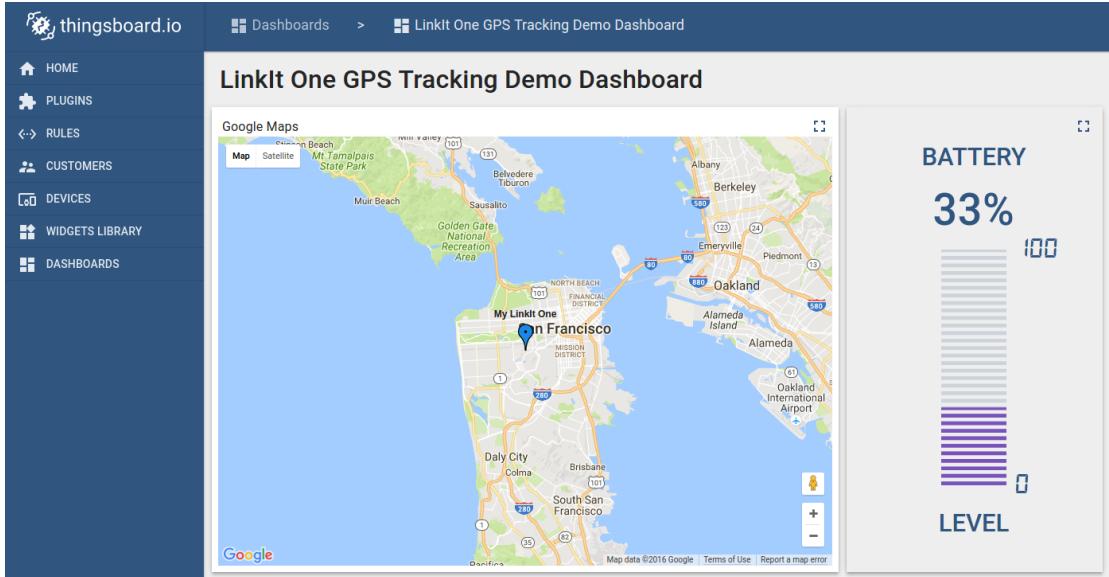


Fonte: AskSensors[25].

Um exemplo com o uso da tecnologia GPS também está disponível na plataforma ThingsBoard, porém com o uso da placa *LinkIt One*¹ e está demonstrado na figura 2.5.

¹*LinkIt One* https://wiki.seeedstudio.com/LinkIt_ONE/

Figura 2.5: Exemplo demonstrativo de um rastreador GPS junto à placa LinkIt ONE na plataforma ThingsBoard.



Fonte: Thingsboard[26].

Ambos exemplos são acompanhados de códigos e bibliotecas que podem ser compilados e executados através da plataforma Arduino IDE[25, 26].

Como as duas plataformas tem uma vasta compatibilidade (tanto de hardware quanto de protocolos de comunicação), é necessário avaliar o custo de manutenção de cada plataforma e também a disponibilidade de planos gratuitos.

A plataforma AskSensors disponibiliza um plano gratuito com duração de três meses. As limitações estão listadas a seguir:

- Apenas dois dispositivos podem ser conectados;
- Não é possível criar dashboards customizáveis;
- A taxa de atualização dos gráficos é de 10s;
- Não há importação ou exportação manual dos dados;
- Não há o suporte para rastreamentos GPS;
- Limite de 30 MB de armazenamento em nuvem.

Como a AskSensors não é de código aberto, não há uma versão da aplicação que possa ser instalada em uma instância local. Ou seja, qualquer opção escolhida dentro dessa plataforma será armazenada pelos servidores próprios da AskSensors.

Nesse ponto, a plataforma ThingsBoard se destaca bastante, justamente por ela ser de código aberto (desenvolvimento feito na linguagem Java) e oferecer uma edição para a comunidade,

totalmente gratuita. Algumas funcionalidades mais avançadas estão indisponíveis na edição de comunidade, como a exportação de dados, geração automática de relatórios e agendamento de tarefas. Porém como a ausência dessas funcionalidades não afeta o projeto, ao contrário das limitações impostas pelo plano gratuito da AskSensors, a plataforma ThingsBoard foi escolhida, utilizando a versão gratuita de comunidade.

É importante destacar que essa versão gratuita precisa ser instalada em uma máquina local ou em uma instância na nuvem, já que tem-se acesso apenas à aplicação e não aos serviços de computação remota oferecidos diretamente pela plataforma.

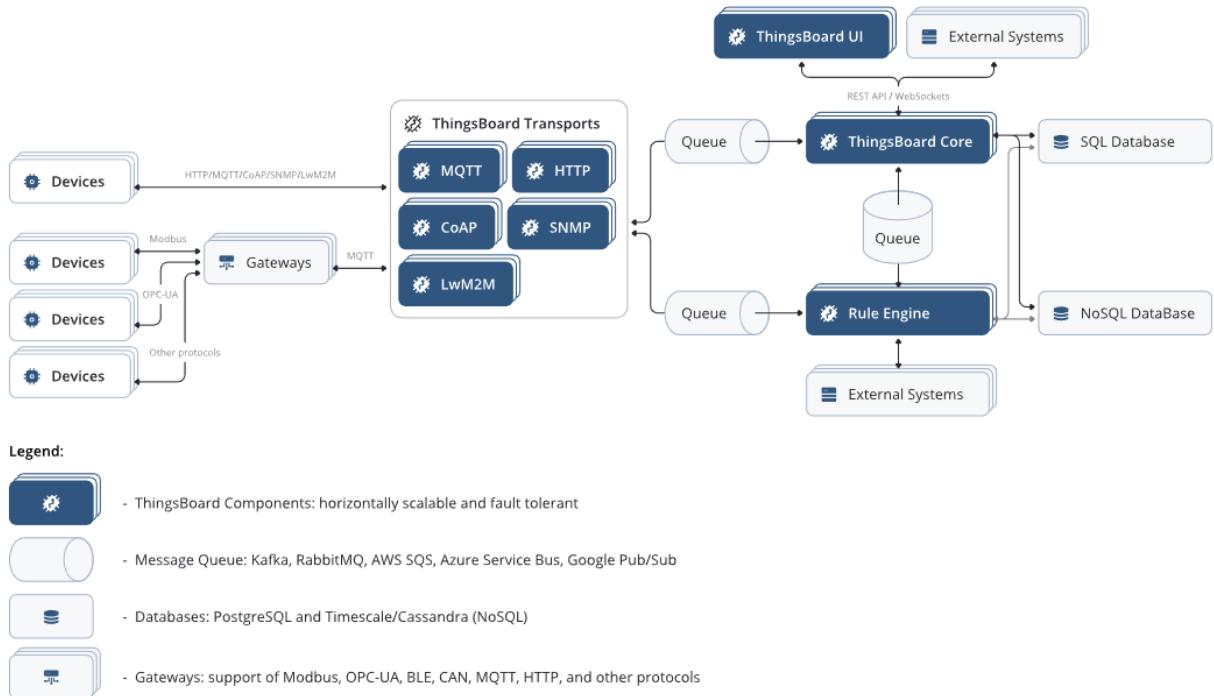
2.2.1 Detalhes e funcionamento da plataforma ThingsBoard

De acordo com o portal da plataforma[27], a ThingsBoard foi desenhada e desenvolvida visando os seguintes princípios:

- **Escalável:** a plataforma é escalável horizontalmente, possibilitando a expansão do projeto de forma nativa;
- **Tolerante a falhas:** É possível acontecer falhas em alguma interface da plataforma sem prejudicar o funcionamento das demais interfaces e leituras de dados, por exemplo;
- **Robusto e eficiente:** Um único servidor da aplicação é capaz de processar dados vindos de até centenas de milhares de dispositivos, dependendo do poder de processamento da máquina/instância a ser utilizado para a instalação da plataforma;
- **Durável:** A plataforma é capaz de armazenar dados cruciais em filas, tornando as mensagens duráveis enquanto percorrem a arquitetura da aplicação;
- **Customizável:** A adição de novas funcionalidades é simples com *widgets* (Unidades que compõem um dashboard e demonstram os dados apresentados, por exemplo um gráfico de pizza ou até mesmo um mapa) customizáveis e através de processamento de dados através do *rule engine nodes*.

O funcionamento da plataforma como um todo está esquematizado na figura 2.6.

Figura 2.6: Esquemático da plataforma ThingsBoard e fluxos de dados disponíveis.



Fonte: ThingsBoard [24].

2.2.1.1 Possibilidades de instalação

A ThingsBoard suporta a instalação em diversos sistemas operacionais e plataformas localmente:

- Ubuntu;
- CentOS;
- Windows;
- Raspberry Pi 3;
- Instalação em contêiner (Docker).

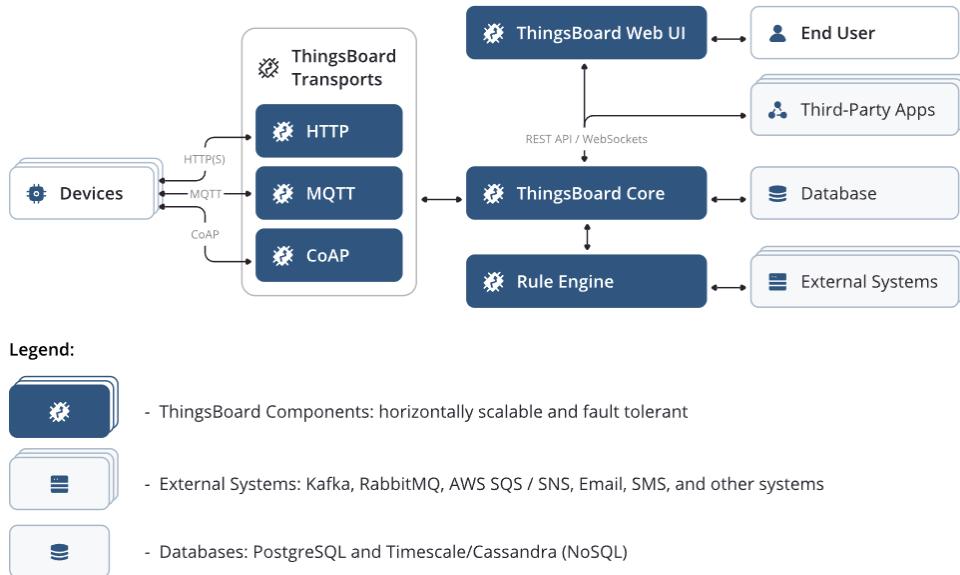
Além dessas opções, é possível instalar a plataforma em diversas instâncias na nuvem. Porém, atualmente os tutoriais só estão disponíveis para os serviços:

- AWS (Amazon Web Services);
- Digital Ocean;
- GCP (Google Cloud Platform);

Em breve, estarão disponíveis guias para o serviço da Microsoft, o Azure, além dos serviços da IBM (IBM Cloud) e da Alibaba (Alibaba Cloud).

A arquitetura escolhida para a instalação do ThingsBoard foi a arquitetura monolítica (representada na figura 2.7), ou seja, toda a aplicação, tanto a interface web com o usuário quanto o código de acesso aos dados estão contidos em um único pacote. O ThingsBoard permite a instalação através da arquitetura por microsserviços, porém ela é indicada para um cenário que milhões de dispositivos estejam conectados com a plataforma.

Figura 2.7: Arquitetura monolítica do ThingsBoard.



Fonte: ThingsBoard Deploy [28].

Duas opções de serviços de banco de dados estão incluídos nessa versão monolítica da aplicação. É possível escolher que os dados recebidos sejam armazenados em um banco PostgreSQL ou em um banco que utiliza a tecnologia NoSQL (Cassandra ou Timescale). A orientação por parte do desenvolvedor do ThingsBoard é que o PostgreSQL seja utilizado na arquitetura escolhida[28]. Essa configuração permite o processamento de cerca de 5 mil pontos de telemetria por segundo, o que é mais do que suficiente para o projeto atual.

Também é possível conectar o ThingsBoard a serviços externos de banco de dados, mas é um cenário recomendado para um grande volume de dados. O fato do serviço de armazenamento de dados estar separado da aplicação principal reduz riscos em falhas totais no sistema, remetendo ao ponto de durabilidade dos dados da ThingsBoard.

2.2.1.2 ThingsBoard *Rule Engine*

Um dos serviços essenciais para o funcionamento da plataforma é o *Rule Engine*. Esse serviço é um sistema altamente customizável e configurável para processamento de eventos complexos. Com a Rule Engine é possível filtrar, transformar e enriquecer mensagens de telemetria enviadas

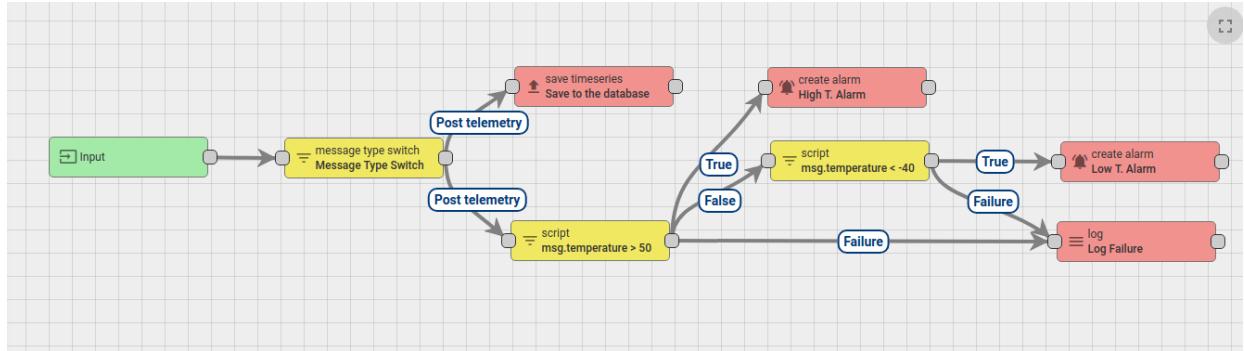
pelos dispositivos. Também é possível disparar diversas ações, como notificações ou comunicações com serviços externos[29].

Cada mensagem transferida dentro da Rule Engine é uma estrutura de dados serializável e imutável que pode representar vários tipos de mensagens presentes no sistema, como mensagens de telemetria, atualizações de atributos (características estáticas de dispositivos) ou ações de comando para os dispositivos.

Dentro da Rule Engine existem os Rule Nodes (nós). Os nós são funções escritas em *JavaScript* que podem filtrar, transformar ou executar alguma ação a partir da mensagem recebida por determinado nó.

Por fim, a Rule Chain (cadeia de eventos) determina a relação entre os nós e como as saídas de cada nó devem ser repassadas aos próximos nós. O exemplo da figura 2.8 traz um Rule Chain que visa salvar todos os dados de telemetria na base de dados, disparar o alarme de alta temperatura caso a temperatura seja maior que 50° C, disparar o alarme de baixa temperatura caso seja menor que -40° C e guardar registros de falhas no processamento (*log failure*)[29].

Figura 2.8: Processamento de mensagens de telemetria através do serviço de Rule Engine.



Fonte: ThingsBoard Rule Engine 2.8.

Utilizando o mesmo mecanismo, é possível criar filtros para exclusão de dados que não fazem sentido (quando ocorrem problemas nas conexões físicas de um sensor, por exemplo) e até mesmo implementar calibrações de sensores.

2.3 Comunicação

De acordo com a figura 1.11, há duas pontes de comunicação dentro do escopo do projeto. A comunicação entre os nós de medição e o Gateway, e a comunicação entre o Gateway e o servidor remoto. Elas serão abordadas de forma independente dentro dessa seção.

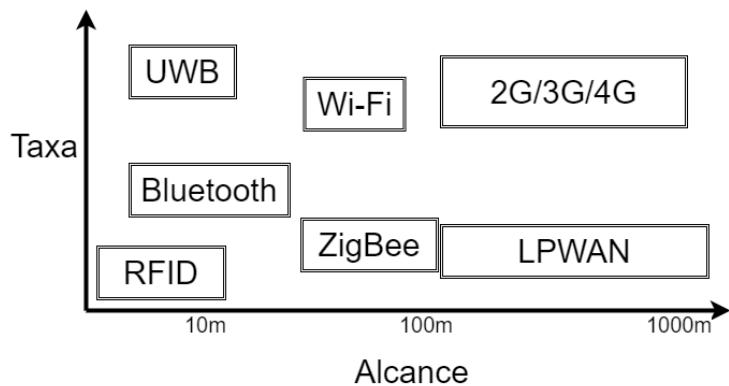
2.3.1 Comunicação entre os nós de medição e o Gateway

Com o desenvolvimento da tecnologia da informação, muitas tecnologias sem fio foram propostas para uma grande variedade de aplicações. No geral, essas tecnologias podem ser divididas em três categorias[3]:

- Tecnologias sem fio de curta distância (abaixo de 10m);
- Tecnologias sem fio de média distância (de 10m até 100m);
- Tecnologias sem fio de longa distância (acima de 100m).

Alguns exemplos de tecnologias de curta distância são o RFID (*Radio-frequency identification*) e o Bluetooth. Para média distância, os exemplos mais notáveis são a WiFi, comum e presente na maioria dos celulares e computadores nos tempos atuais, e o ZigBee. Por fim, para longas distâncias, as redes de celulares (2G/3G/4G) estão presentes, além da tecnologia LoRa (*Long Range*). Na figura 2.9 é possível ter uma comparação entre esses exemplos e outras tecnologias, no quesito de alcance e largura de banda:

Figura 2.9: Comparação entre diferentes de tecnologias de comunicação sem fio.



Fonte: Adaptado de Feng et al. [3].

Na tabela 2.2 é possível ter um comparativo entre algumas das tecnologias representadas na figura 2.9.

Tabela 2.2: Comparação entre diferentes tecnologias de comunicação sem fio[3].

Parâmetro/Tecnologia	ZigBee	Wi-Fi	LoRa
Padrão	IEEE 802.15.4	IEEE 802.11	IEEE 802.15.4G
Frequência	868/915 MHz e 2.4GHz (sem licença)	2.4GHz (sem licença)	868/915 MHz (sem licença)
Taxa de dados	20, 40 e 250 kbps	Até 150 Mbps	50 kbps
Consumo (Transmissão)	37 mW	835 mW	100 mW
Alcance	100m	100m	2 a 5 km (centros urbanos)

Dado o contexto e as necessidades do projeto, apenas as opções de longa distância foram consideradas, já que o projeto acaba lidando com transectos longos e a largura de banda não precisa ser alta. O conceito de tecnologias LPWA (*Low power wide area*) vêm de encontro à essa necessidade. Em oposição às tecnologias tradicionais de longa distância (2G/3G/4G), as tecnologias LPWA focam em um baixo consumo de energia, com baixas taxas de transferência de dados, além da ampla cobertura, tanto no sentido de número de conexões quanto na área de cobertura em si.[3]

Analisando os requisitos para o funcionamento das tecnologias tradicionais de longa distância (2G/3G/4G), como a necessidade de um pagamento recorrente para a assinatura de planos mensais, além da dificuldade significativa de melhorar a cobertura em áreas que não são atendidas atualmente por esses serviços, optou-se por utilizar a tecnologia LoRa no projeto, já que ela traz a cobertura ampla, sem a necessidade de alto investimento em infraestrutura e sem grande consumo energético.

2.3.2 Aspectos Regulatórios

A Resolução 671/2016 da Anatel (Agência Nacional de Telecomunicações), estabelece os parâmetros gerais de administração, condições de uso, autorização e controle de radiofrequências, em território nacional, do espectro de radiofrequências[30].

Alguns princípios nortearam a criação e desenvolvimento desse regulamento, como a constatação que esse espectro é um recurso limitado e constituído como um bem público, administrado pela Anatel. Além disso, princípios de uso como a utilização eficiente, adequada, racional, econômica também foram levados em consideração.

O regulamento define também, em seu Artigo 1º, parágrafo 2º que aplicações com fins industriais, científicas, médicas e militares não dependem da autorização da Anatel.

O trabalho atual se encaixa nesse caso, por possuir fins científicos, e por isso, não é necessário uma autorização da Anatel para a execução deste projeto. É importante notar, porém, que como o dispositivo a ser utilizado (definido na seção 3.1) não possui homologação da Anatel, o projeto não deverá ser replicado com fins comerciais sem a devida autorização e homologação do uso do dispositivo por parte da Anatel.

2.3.3 Comunicação entre o Gateway e a plataforma IoT

A comunicação entre o Gateway (nó do sistema que coletará os dados via LoRa) e a plataforma IoT depende da compatibilidade de protocolos de comunicação aceitos pela opção escolhida na Seção 2.2. De acordo com a documentação da ThingsBoard[27], a plataforma suporta nativamente quatro protocolos de comunicação, sendo eles:

- Protocolo HTTPS (*Hyper Text Transfer Protocol Secure*);
- Protocolo MQTT (*Message Queuing Telemetry Transport*);
- Protocolo CoAP (*Constrained Application Protocol*);
- Protocolo LwM2M (*Lightweight Machine to Machine*).

Além dos protocolos acima, a plataforma suporta implementações customizadas de protocolos de transporte. Um fator que é decisivo na escolha do protocolo é o suporte do SDK (*Software Development Kit*) oferecido pelo desenvolvedor. Esse SDK é utilizado no exemplo descrito pela figura 2.5. Em resumo, esse SDK é uma biblioteca desenvolvida pelos mesmos desenvolvedores da plataforma. Ela é compatível com o Arduino IDE, o que facilita sua implementação para diversos microcontroladores e placas[31].

Atualmente, o SDK suporta apenas o protocolo MQTT para se comunicar com a plataforma, necessitando apenas que o dispositivo esteja numa rede na qual a plataforma também consiga acessar, seja através da internet ou em rede local. Além disso, o SDK fornece exemplos claros de envio de telemetria, retorno de ações para o dispositivo e atualização de atributos. Por fim, ele também gera os tópicos utilizados no protocolo MQTT de forma automática, então não há a necessidade de criar e gerenciar tópicos manualmente.

Em resumo, após as discussões realizadas ao longo desse capítulo, foi decidida a utilização da tecnologia GPS para atender o objetivo de localização; a plataforma IoT ThingsBoard para a visualização (e possível expansão da coleta) dos dados; e o protocolo de comunicação LoRa para a comunicação entre os pontos de medição e o Gateway do projeto.

Capítulo 3

Desenvolvimento

Neste capítulo será abordado o processo de escolha do hardware utilizado, a instalação da plataforma ThingsBoard, além da implementação dos algoritmos. São descritas também as funcionalidades disponíveis após a configuração do projeto.

3.1 Escolhas de Hardware

A partir das decisões de tecnologias a serem utilizadas abordadas no capítulo 2 a procura por um hardware compatível foi iniciada. Ou seja, é necessário que o módulo de localização escolhido opere com a tecnologia GPS e que o módulo de comunicação seja compatível com a tecnologia LoRa. Além disso, é necessário encontrar um microcontrolador que seja compatível com a IDE do Arduino.

Com o propósito de diminuir os possíveis pontos de falha e simplificar a implementação no projeto, a pesquisa inicialmente foi feita por uma placa que atingisse todos os objetivos sem a necessidade de adquirir módulos externos. Nesse contexto, foi encontrada a plataforma LILYGO TTGO T-Beam da figura 3.1.

Figura 3.1: Placa LILYGO TTGO T-Beam Versão 1.0.



Fonte: Loja Filipeflop.

A placa possui as seguintes especificações:

- Microcontrolador ESP32 Dual-core, com clock de até 240 MHz e memória interna de 520 kB (SRAM);
- Módulo Wifi 802.11 b/g/n de 2.4 GHz integrado;
- Bluetooth 4.2 integrado;
- 14 pinos de GPIO (*General Purpose Input/Output*);
- Módulo GPS u-blox NEO-6M com antena cerâmica;
- Módulo LoRa Semtech SX1276 com antena de 915 MHz;
- Suporte para bateria de íons de lítio 18650;
- PMIC (*Power Management Integrated Circuit*) AXP192 capaz de controlar a tensão fornecida para os diferentes módulos da placa, além de controlar o carregamento da bateria 18650;
- Interface de programação via porta Micro USB.

Como a placa utiliza o microcontrolador ESP32, da Espressif, o suporte ao Arduino IDE é garantido pelo fabricante. Além disso, a placa conta com um repositório¹ com exemplos de utilização de diversos módulos da placa.

Como a placa atende a todos os requisitos estabelecidos, ela foi escolhida para o desenvolvimento do projeto. O custo por placa foi de R\$ 400 em agosto de 2021 através do portal Filipeflop². Em novembro, a placa foi encontrado no mesmo portal ao custo de R\$ 450³.

Para fins de geração de dados de testes, foi utilizado o sensor DHT11. Como não está no escopo do projeto a escolha do sensor de temperatura e umidade, esse sensor foi utilizado apenas com o propósito de simular o sensor utilizado no projeto Mochila Bioclimática[17].

Além disso, foi utilizado um módulo Display OLED de 0.96 polegadas de tamanho (diagonal) e com uma resolução de 128x64 pixels, demonstrado na figura 3.2. Essa tela foi utilizada para facilitar a visualização dos dados ao longo do desenvolvimento dos algoritmos, removendo a necessidade da placa precisar de um computador por perto para depurar a implementação do código.

¹Repositório com exemplos de uso desenvolvidos pela LILYGO <https://github.com/LilyGO/TGOT-Beam>

²Filipeflop <https://www.filipeflop.com/>

³TGOT Beam - Filipeflop

<https://www.filipeflop.com/produto/modulo-wifi-esp32-com-suporte-de-bateria-gps-e-lora-915mhz/>

Figura 3.2: Display OLED 128x64 0.96 polegadas.



Fonte: Loja Filipeflop.

Por fim, os últimos componentes físicos utilizados no projeto foram as baterias de íons de lítio 18650 (Figura 3.3). Além de fornecerem versatilidade, foram essenciais para os testes em campo, devido ao fraco sinal GPS dentro de construções. Cada bateria tem uma capacidade 2400 mAh e sua tensão nominal é de 3.7V.

Figura 3.3: Baterias de íons de lítio no modelo 18650.



Fonte: Loja Filipeflop.

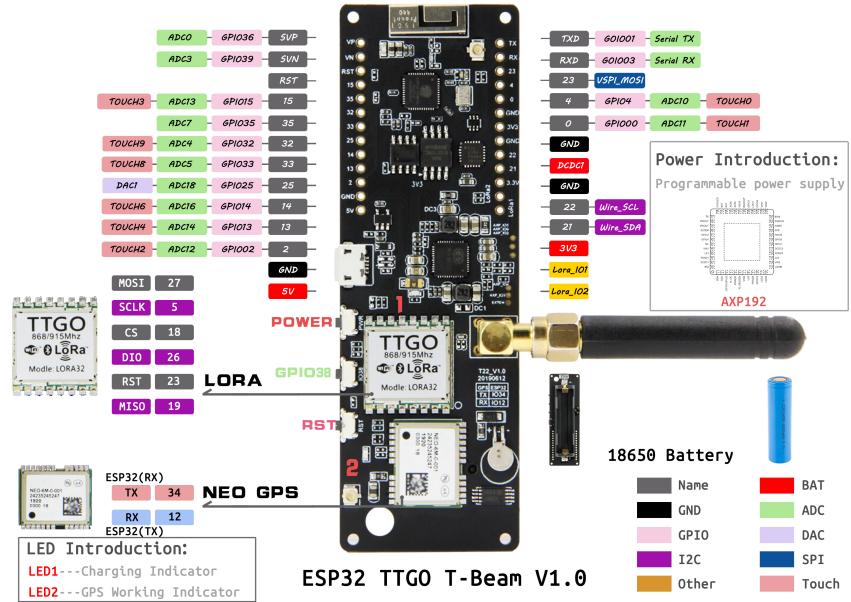
3.2 Implementação dos algoritmos na placa TTGO T-Beam

Para começar o desenvolvimento do projeto, foram utilizados de base os códigos de exemplo fornecidos pelo fabricante LILYGO comentados na seção 3.1. Os códigos foram desenvolvidos na linguagem C++.

Na subseção 2.3.3 foi destacada a existência de um SDK desenvolvido pela própria Thingsboard com o intuito de facilitar a integração de projetos feitos na Arduino IDE com a plataforma.

A placa TTGO T-Beam possui um mapeamento de pinos específico para cada módulo interno. Todos os pinos estão descritos na figura 3.4:

Figura 3.4: Mapeamento dos pinos na placa TTGO T-Beam.



Fonte: Github LILYGO T-Beam[32].

Para simplificar o texto, os códigos serão interpretados através de fluxos. Em cada unidade do projeto (Sensoriamento ou Gateway) são utilizados três arquivos, sendo o código principal, um arquivo de cabeçalho com a inicialização da placa, fornecida pela fabricante, e um arquivo de cabeçalho com uma lista de parâmetros. Os códigos completos estão disponíveis no anexo do documento e no repositório público do projeto¹.

O Arduino IDE deve ser configurado para conseguir compilar códigos para a TTGO T-Beam. Para isso, é necessário abrir a IDE e ir em "Arquivos -> Preferências ". Em "URLs Adicionais para Gerenciadores de Placas" adicione o link https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json. Feito isso, pode-se voltar ao Arduino IDE e agora acessar "Ferramentas -> Placa -> Gerenciador de Placas...". Dentro do gerenciador de tarefas, deve-se pesquisar por esp32 e instalar o pacote. Para verificar a instalação e configurar a IDE para compilar os códigos para a TTGO T-Beam, pode-se retornar ao menu "Ferramentas -> Placa" e selecionar a opção T-Beam. Feito isso, os códigos poderão ser compilados e enviados normalmente.

3.2.1 Unidade de sensoriamento

A unidade de sensoriamento é responsável pela coleta de informações, confecção da mensagem JSON (*JavaScript Object Notation*)² e o envio dessa mensagem via LoRa. Esse ciclo é repetido periodicamente, sendo possível aumentar ou diminuir esse período, com o propósito de controlar a frequência das aferições e do disparo de mensagens. Além disso, também é possível alterar o

¹Repositório do projeto no Github

<https://github.com/embedded-computing/urban-climate-monitoring>

²JSON <https://www.json.org/json-en.html>

número identificador da placa (`board_id`), o qual identificará a placa de origem no Gateway.

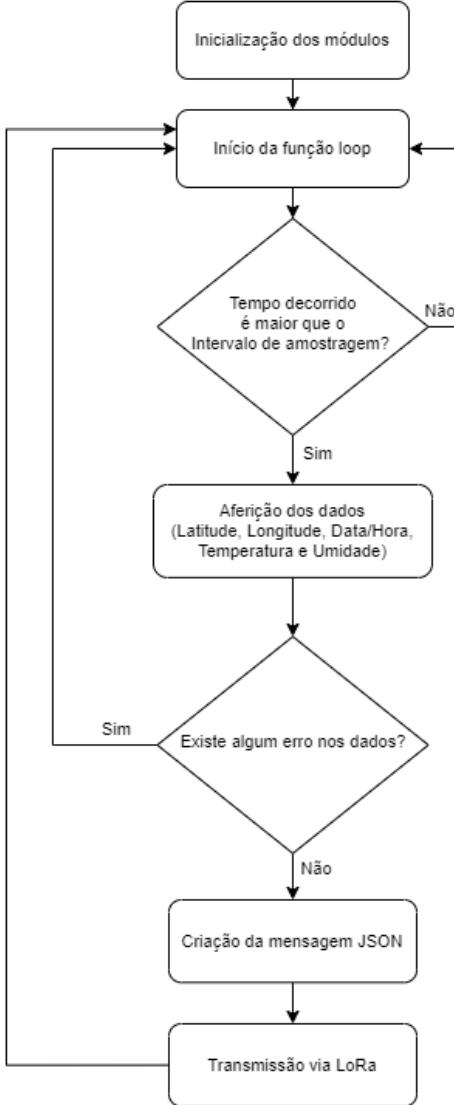
O JSON construído pelo algoritmo tem a seguinte estrutura:

- **board_id**: Identificador da placa. Tipo inteiro;
- **packet_id**: Identificador sequencial do pacote. Tipo inteiro;
- **ts**: *Timestamp*, ou seja, momento da aferição dos dados. Esse número segue o formato *Unix*¹. Tipo string;
- **temperature**: Aferição de temperatura. Tipo inteiro;
- **humidity**: Aferição de umidade. Tipo inteiro;
- **latitude**: Aferição de latitude. Tipo ponto flutuante;
- **longitude**: Aferição de longitude. Tipo ponto flutuante;

O fluxo da figura 3.5 representa o funcionamento do código implementado na unidade de sensoriamento.

¹Unix Timestamp <https://www.unixtimestamp.com/>

Figura 3.5: Funcionamento simplificado do código nas unidades de sensoriamento.

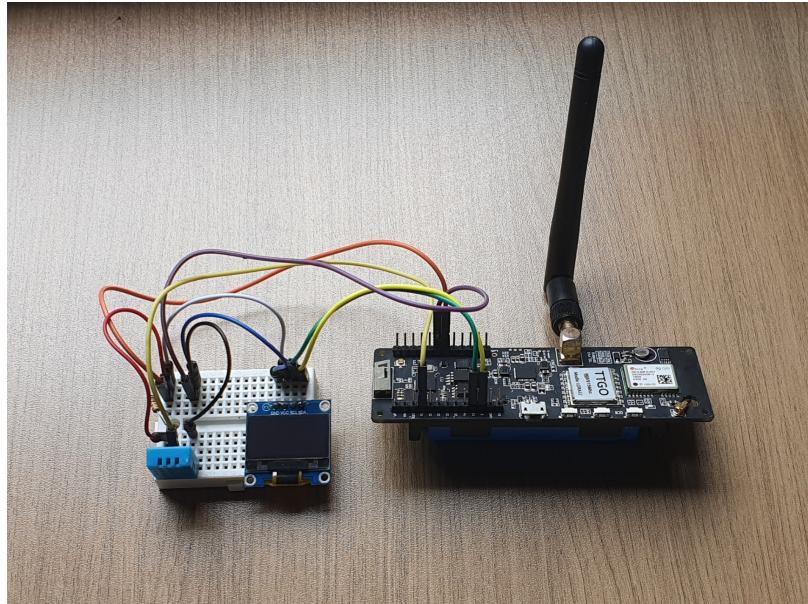


Fonte: Figura do Autor.

O código possibilita a expansão do projeto de maneira prática. Caso deseja-se aferir uma nova métrica, por exemplo pressão barométrica, basta adicionar os comandos para recolher os dados a partir do novo sensor/módulo e criar um novo campo no JSON montado.

Também pode-se destacar que o intervalo de tempo entre aferições, ou taxa de amostragem, é feita a cada cinco segundos. Esse parâmetro pode ser alterado no código. A montagem da unidade de sensoriamento está exemplificada na figura 3.6.

Figura 3.6: Montagem da unidade de sensoriamento.



Fonte: Figura do Autor.

3.2.2 Gateway

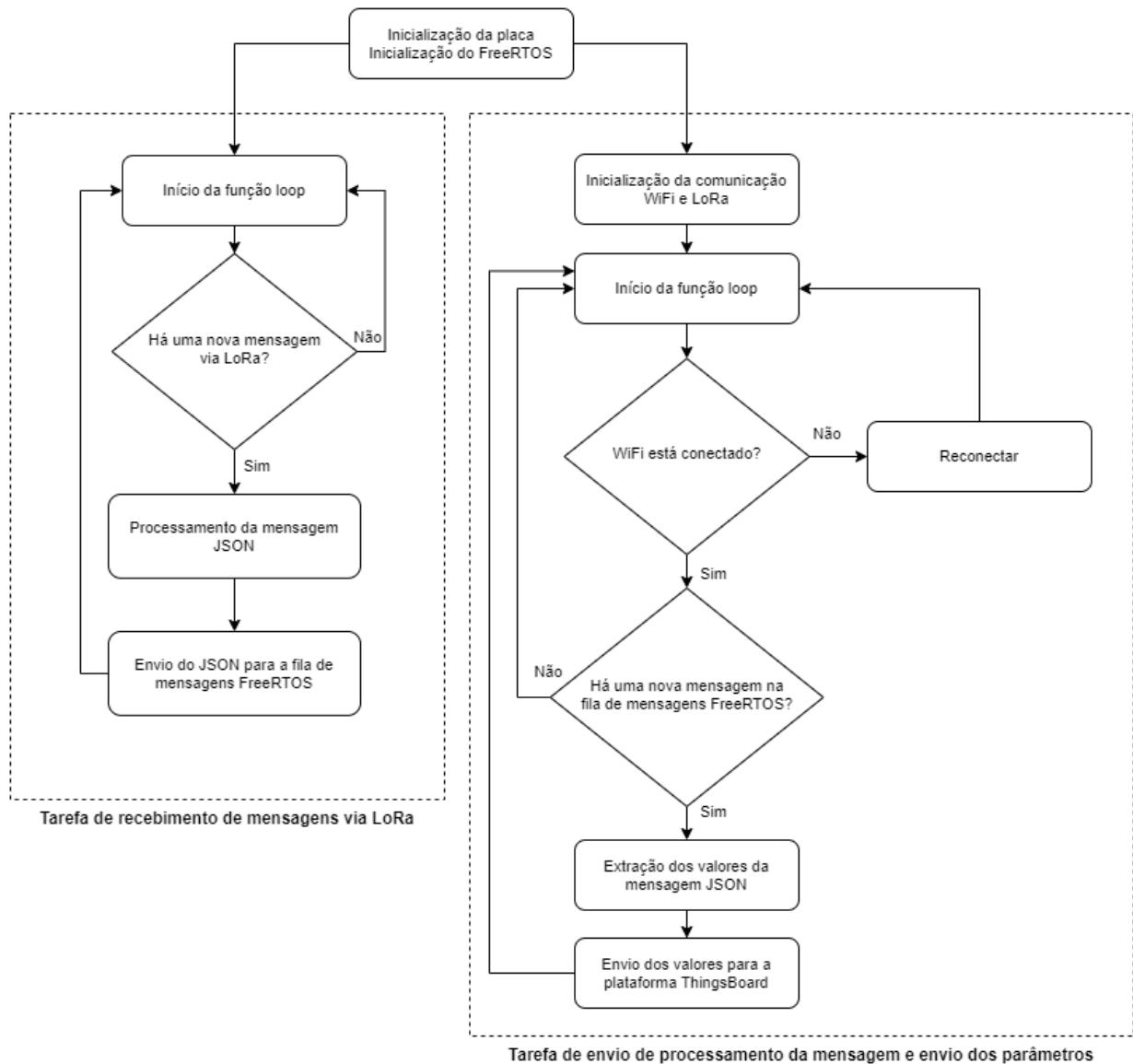
O Gateway é responsável pelo recebimento, processamento e reenvio das informações encaminhadas pelas unidades de sensoriamento. Apesar do desenvolvimento inicialmente ser similar ao das unidades de sensoriamento, há uma grande diferença no paradigma de programação a ser utilizado por esse módulo. Como as mensagens via LoRa podem ser recebidas a qualquer momento, há a necessidade de se monitorar constantemente o LoRa para verificar se há uma mensagem nova. Porém, após o recebimento de uma mensagem, é necessário processá-la, o que pode acabar bloqueando o recebimento de outras mensagens (de outras unidades de sensoriamento, por exemplo) quando se utiliza uma programação linear como a demonstrada pela figura 3.5. Ou seja, caso o paradigma de programação linear fosse utilizado no desenvolvimento do Gateway, seu desempenho em receber mensagens de múltiplos nós de sensoriamento poderia ser prejudicado.

Nesse contexto, é introduzido o FreeRTOS (*Free Real Time Operating System*)¹ com o propósito de criar tarefas que podem ser executadas paralelamente, de forma não linear. Como descrito pela seção 3.1, o microcontrolador utilizado pela placa TTGO T-Beam é o ESP32, com dois núcleos de processamento, o que acaba sendo extremamente útil nesse tipo de situação.

A figura 3.7 mostra que duas tarefas principais são executadas no Gateway: o recebimento de mensagens JSON via LoRa e o processamento/envio dessa mensagem para a plataforma ThingsBoard. Com o FreeRTOS, é possível fazer com que essas tarefas possam ser executadas de forma assíncrona. As duas tarefas se comunicam através de uma fila. Em alguns momentos no algoritmo, é necessária uma continuidade do processamento, sendo indispensável a criação de um semáforo, o qual permite dar exclusividade de execução ao longo de tarefas que não possam ser executadas paralelamente (Criação da conexão WiFi, aferição das medidas do módulo GPS, entre outros).

¹Free RTOS <https://www.freertos.org/>

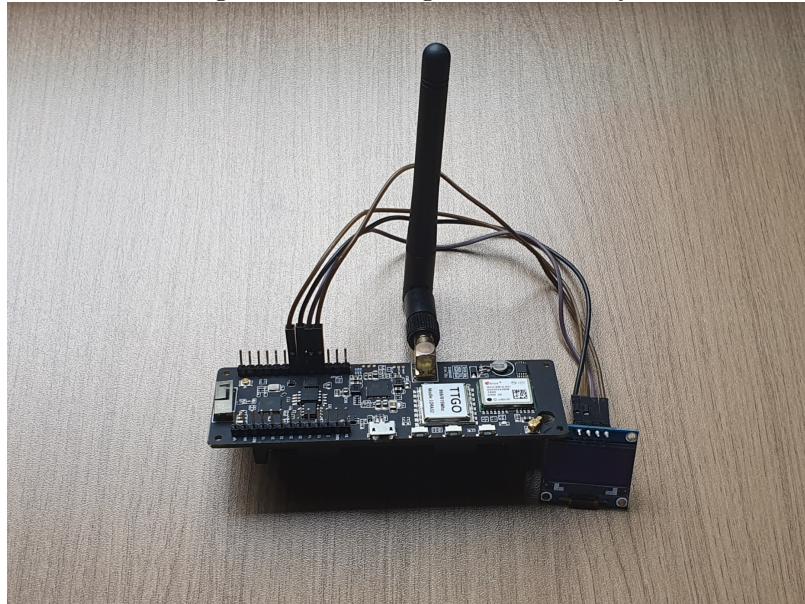
Figura 3.7: Funcionamento simplificado do Gateway.



Fonte: Figura do Autor.

O Gateway é também responsável pela identificação de qual unidade de sensoriamento mandou o pacote recebido. Os tokens de acesso de cada unidade (os quais serão detalhados na subseção 3.3.3.1) ficam armazenados no Gateway. A partir da identificação da placa através do parâmetro *board_id*, descrito na subseção 3.2.1, o Gateway seleciona a credencial de acesso, possibilitando a diferenciação de mensagens entre as diversas unidades. A montagem do hardware do Gateway pode ser observada na figura 3.8.

Figura 3.8: Montagem do Gateway.



Fonte: Figura do Autor.

3.3 Implementação da plataforma IoT

Como foi discutido na subseção 2.2.1.1, a plataforma ThingsBoard suporta a instalação em diversos sistemas operacionais, dispositivos e serviços de computação na nuvem. Visando facilitar o acesso à plataforma ao longo do projeto, foi escolhido utilizar uma plataforma de computação na nuvem.

O serviço disponibilizado pela AWS para computação na nuvem é o Amazon EC2 (*Elastic Compute Cloud*)¹. É um serviço web que disponibiliza capacidade computacional segura e redimensionável na nuvem. A interface feita com as instâncias criadas pode ser feita de forma simples via web. Além disso, a Amazon disponibiliza um nível gratuito por 12 meses², no qual é possível utilizar instâncias do tipo t2.micro (1 CPU virtual, 2.5 GHz / 1 GB de Memória RAM / Performance de rede baixa a moderada) por 750 horas ao mês. O poder de processamento de uma instância desse tipo, apesar de ser baixo quando comparado com computadores modernos, é o suficiente para executar o ThingsBoard com a arquitetura escolhida[27].

¹Amazon EC2 <https://aws.amazon.com/pt/ec2/>

²Nível gratuito da AWS <https://aws.amazon.com/pt/free/>

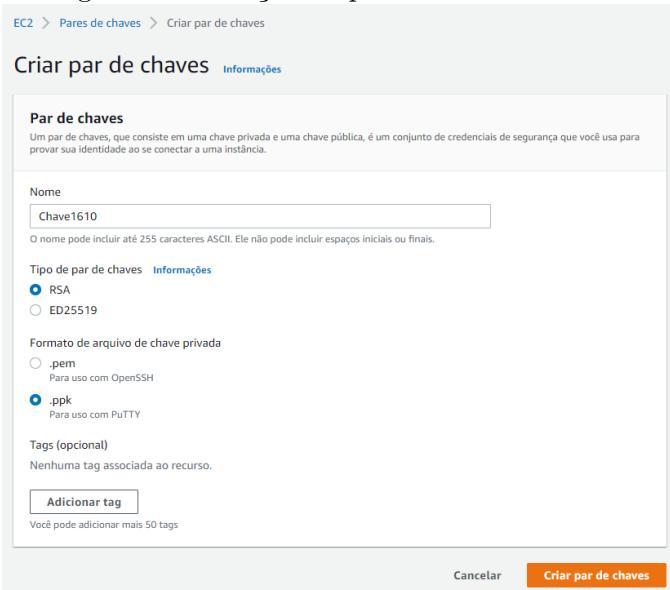
3.3.1 Configuração do Amazon EC2

Para criar uma instância no serviço Amazon EC2, primeiramente é necessária a criação de uma conta na AWS¹. Após a criação da conta, o acesso ao console AWS será disponibilizado. Todos os serviços da AWS podem ser acessados através desse console, porém o projeto utilizará apenas o serviço de computação na nuvem EC2.

3.3.1.1 Criação de par de chaves

O próximo passo é a criação de um par de chaves, o qual vai garantir a segurança no acesso à instância criada. Para criar esse par de chaves, basta acessar o console Amazon EC2² e no painel de navegação na esquerda, selecionar a opção "Pares de chaves". Em seguida, deve ser selecionada a opção "Criar par de chaves", no canto superior direito. Nas opções de criação, é necessário escolher um nome arbitrário para o par de chaves (limite de até 255 caracteres ASCII, sem o uso de espaços no início ou no final do nome). Não é necessária a modificação das outras opções, bastando deixar selecionadas as opções padrões. Caso uma chave já exista, ela pode ser reaproveitada. Essa etapa está exemplificada na figura 3.9.

Figura 3.9: Criação do par de chaves na AWS.



Fonte: Figura do Autor.

Mais detalhes sobre a criação de pares de chave podem ser encontrados no guia da Amazon³.

¹Acesso ao console AWS <https://console.aws.amazon.com/console/home>

²Console Amazon EC2 <https://console.aws.amazon.com/ec2/>

³Criação de pares de chave na AWS

https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html

3.3.1.2 Criação da instância

Após a criação do par de chaves, é necessário o retorno ao console EC2. A opção "Executar instâncias" no canto superior direito deve ser selecionada. Um assistente de criação de Instâncias será iniciado, ilustrado pela figura 3.10.

A primeira etapa dentro desse assistente é a seleção de qual imagem de máquina será utilizada nessa instância, ou seja, qual sistema operacional será executada. Conforme a recomendação disponibilizada no guia da ThingsBoard[33], a imagem a ser selecionada é a *Ubuntu Server 20.04 LTS* em arquitetura 64 bits (x86).

A segunda etapa consiste na escolha do tipo de instância a ser criada. Como o intuito do projeto é de provar um conceito, a princípio será utilizado o tipo de instância *t2.micro*, o qual é o tipo com maior capacidade de processamento e que é compatível com o nível gratuito.

Figura 3.10: Seleção do tipo de instância no Amazon EC2.

The screenshot shows the AWS EC2 Instance Creation Wizard, Step 2: Choose instance type. The table lists the following information:

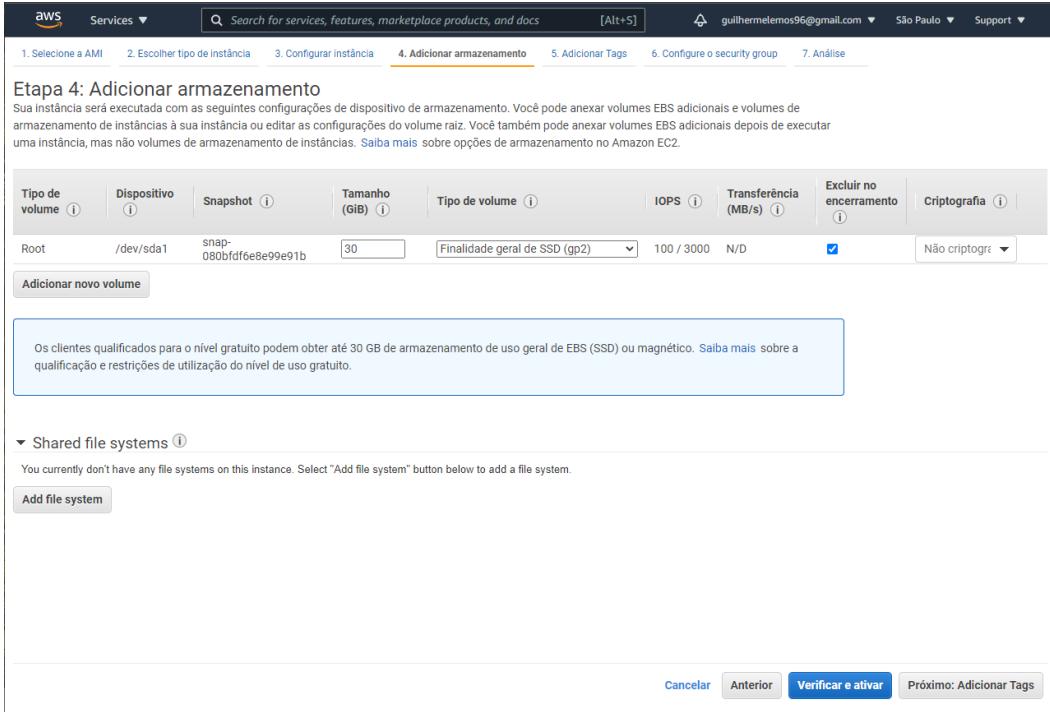
Família	Tipo	vCPUs	Memória (GiB)	Armazenamento da instância (GB)	Disponível otimizado para EBS	Desempenho de rede	Compatibilidade com IPv6
t2	t2.nano	1	0.5	Somente EBS	-	Baixo a moderado	Sim
t2	t2.micro qualificado para o nível gratuito	1	1	Somente EBS	-	Baixo a moderado	Sim
t2	t2.small	1	2	Somente EBS	-	Baixo a moderado	Sim
t2	t2.medium	2	4	Somente EBS	-	Baixo a moderado	Sim
t2	t2.large	2	8	Somente EBS	-	Baixo a moderado	Sim
t2	t2.xlarge	4	16	Somente EBS	-	Moderado	Sim
t2	t2.2xlarge	8	32	Somente EBS	-	Moderado	Sim
t3	t3.nano	2	0.5	Somente EBS	Sim	Até 5 Gigabit	Sim
t3	t3.micro	2	1	Somente EBS	Sim	Até 5 Gigabit	Sim
t3	t3.small	2	2	Somente EBS	Sim	Até 5 Gigabit	Sim
t3	t3.medium	2	4	Somente EBS	Sim	Até 5 Gigabit	Sim

Buttons at the bottom: Cancelar, Anterior, Verificar e ativar, Próximo: Configure os detalhes da instância.

Fonte: Figura do Autor.

Feita a seleção do tipo de instância, virá a terceira etapa "Configurar instância". Nessa etapa, tudo pode ser deixado como padrão e pode-se avançar para a próxima etapa, na qual será escolhido o armazenamento para a sua instância. Como especificado pelo próprio assistente na figura 3.11, é possível criar um volume de até 30 GB dentro do nível gratuito.

Figura 3.11: Seleção do armazenamento.



Fonte: Figura do Autor.

A etapa seguinte, "Adicionar Tags", também pode ser pulada, deixando as opções padrões preenchidas. Já na etapa "Configure o security group" é necessária a criação de um novo grupo de segurança (o nome é arbitrário). Esse grupo de segurança controla as regras de tráfegos de rede a partir de endereços de origem especificados. Como o projeto não ficará em execução constantemente e poderá ser acessado a partir de múltiplos endereços de origem, não foram colocadas restrições de acesso. As portas abertas e seus respectivos protocolos para tráfego de rede estão demonstradas na figura 3.12 e devem ser replicadas de acordo com a documentação da ThingsBoard [33]:

Figura 3.12: Regras de tráfego de rede.

Tipo	Protocolo	Intervalo de Portas	Origem	Descrição
HTTP	TCP	80	Personaliz.	0.0.0.0/0, ::/0 TB-TCP2
Regra personalizada	TCP	8080	Personaliz.	0.0.0.0/0 Por exemplo SSH for Admin Desk
SSH	TCP	22	Personaliz.	0.0.0.0/0, ::/0 SSH
Regra personalizada	UDP	5683	Personaliz.	0.0.0.0/0, ::/0 TB-UDP
Regra personalizada	TCP	1883	Personaliz.	0.0.0.0/0, ::/0 TB-TCP4
HTTPS	TCP	443	Personaliz.	0.0.0.0/0, ::/0 TB-TCP3

Aviso
Regras com origem 0.0.0.0/0 permitem que todos os endereços IP acessem sua instância. Recomendamos configurar regras de grupo de segurança para permitir o acesso apenas de endereços IP conhecidos.

Cancelar **Anterior** **Verificar e ativar**

Fonte: Figura do Autor.

Após a definição de regras para o tráfego de rede, a configuração está concluída. Depois de selecionar a opção "Verificar e ativar" (canto inferior direito), o assistente mostrará uma página resumo com todos os detalhes que foram discutidos nas etapas anteriores. Por fim, após o clique em "Executar", será requisitada a seleção do par de chaves criada na subseção 3.3.1.1. Feita a seleção das chaves, a instância estará finalmente criada, como mostrado na figura 3.13.

Figura 3.13: Instância criada.

Name	ID de instância	Estado da instância	Tipo de inst...	Verificação de status	Status do al...	Zona de dispon...
-	i-0dcc8a53e67b314d7	Interrompido	t2.micro	-	Sem alar...	sa-east-1a

Fonte: Figura do Autor.

Para acessar a instância recém criada, é necessário retornar ao console EC2, selecionar a instância e pressionar a opção "Estado da instância -> Iniciar Instância". Após alguns segundos a inicialização estará concluída. Em seguida, pode-se acessar o terminal Ubuntu a partir da opção "Conectar", também presente no console EC2, como ilustrado na figura 3.14.

Figura 3.14: Instância Ubuntu Server em execução no Amazon EC2.

The screenshot shows a terminal window with the following content:

```
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1020-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Fri Nov  5 07:12:51 UTC 2021

System load: 0.37      Processes:          102
Usage of /: 4.7% of 29.02GB  Users logged in:     0
Memory usage: 19%           IPv4 address for eth0: 172.31.5.240
Swap usage:  0%

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-5-240:~$
```

At the bottom of the terminal window, there is a status bar with the text:

i-0b1db552da88af7b2
Public IPs: 18.231.21.93 Private IPs: 172.31.5.240

Fonte: Figura do Autor.

O acesso à instância também pode ser feito através de outras formas, por exemplo através de um cliente SSH. A instância agora funciona como um computador remoto executando o sistema operacional escolhido. Mais detalhes da criação de uma instância no serviço Amazon EC2 podem ser encontrados no guia disponibilizado pela Amazon¹.

3.3.2 Instalação da ThingsBoard

Com a criação da instância remota, agora é necessária a instalação da plataforma ThingsBoard. A primeira etapa necessária é a instalação do Java 11 (OpenJDK), já que a plataforma é desenvolvida na linguagem Java. Os seguintes comandos devem ser executados no terminal demonstrado pela figura 3.14:

```
$ sudo apt update
$ sudo apt install openjdk-11-jdk
```

A próxima etapa é o download e a instalação do serviço ThingsBoard em si:

```
$ wget https://github.com/thingsboard/thingsboard/releases/download
    ↳ /v3.3.1/thingsboard-3.3.1.deb
```

¹Criação de uma instância EC2

https://docs.aws.amazon.com/pt_br/efs/latest/ug/gs-step-one-create-ec2-resources.html

```
$ sudo dpkg -i thingsboard-3.3.1.deb
```

Concluída a instalação, agora é necessário configurar a plataforma como foi discutido na subseção 2.2.1.1, ou seja, utilizando o banco de dados PostgreSQL. A instalação do PostgreSQL é feita através dos seguintes comandos:

```
$ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.  
→ asc | sudo apt-key add -  
$ RELEASE=$(lsb_release -cs)  
$ echo "deb http://apt.postgresql.org/pub/repos/apt/ ${RELEASE} -  
→ pgdg main | sudo tee /etc/apt/sources.list.d/pgdg.list  
$ sudo apt update  
$ sudo apt -y install postgresql-12  
$ sudo service postgresql start
```

Após a instalação do PostgreSQL, é necessária a definição de uma senha para o usuário principal, o que pode ser feito com os comandos:

```
$ sudo su - postgres  
$ psql  
# \password  
# \q
```

Depois de ter definido a senha do usuário principal, é preciso criar a base de dados com a qual a ThingsBoard se comunicará. É imprescindível o retorno ao usuário padrão do Ubuntu, o que pode ser alcançado com o atalho no teclado "Ctrl + D". Os comandos a seguir criam a base de dados dentro do PostgreSQL:

```
$ psql -U postgres -d postgres -h 127.0.0.1 -W  
# CREATE DATABASE thingsboard;  
# \q
```

Com isso, a configuração do banco de dados está concluída. É indispensável o armazenamento seguro da senha criada nessa etapa, já que ela poderá fornecer acesso a todos os dados coletados pela aplicação. A seguir, o arquivo de configuração da ThingsBoard precisa ser editado para localizar e acessar a base de dados recém criada. O arquivo pode ser acessado com o comando:

```
$ sudo nano /etc/thingsboard/conf/thingsboard.conf
```

As seguintes linhas devem ser adicionadas ao final do arquivo de configuração:

```
export DATABASE_ENTITIES_TYPE=sql  
export DATABASE_TS_TYPE=sql  
export SPRING_JPA_DATABASE_PLATFORM=org.hibernate.dialect.  
→ PostgreSQLDialect  
export SPRING_DRIVER_CLASS_NAME=org.postgresql.Driver
```

```

export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/
    ↗ thingsboard
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=SENHA_POSTGRESQL_AQUI
export SPRING_DATASOURCE_MAXIMUM_POOL_SIZE=5
export SQL_POSTGRES_TS_KV_PARTITIONING=MONTHS
export JAVA_OPTS="$JAVA_OPTS -Xms256M -Xmx256M"

```

É indispensável a substituição da palavra "SENHA_POSTGRESQL_AQUI" pela senha de acesso do usuário principal PostgreSQL definida anteriormente.

Com a instalação e configuração da ThingsBoard, além da instalação e configuração do banco de dados, o script a seguir pode ser executado para carregar os dados demonstrativos que já são incorporados à edição de comunidade e também para inicializar o serviço ThingsBoard:

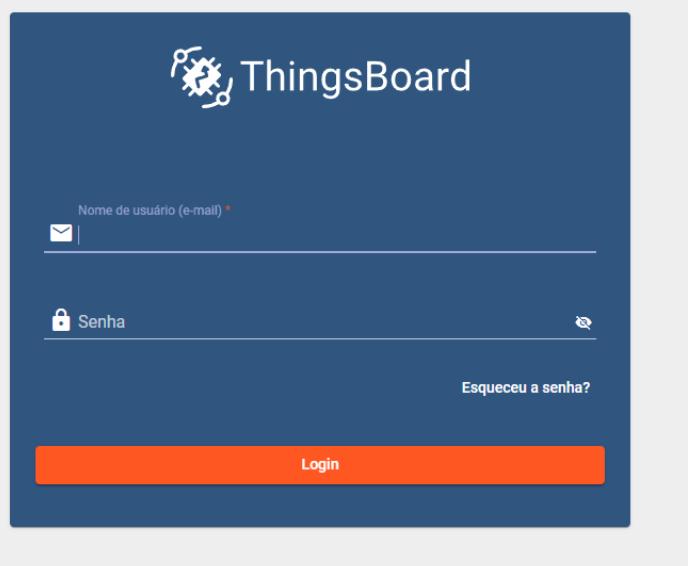
```

$ sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo
$ sudo service thingsboard start

```

Uma vez iniciado, será possível acessar a interface web através do endereço público, definido pelo serviço Amazon EC2 e que pode ser encontrado na figura 3.14. A porta de acesso deve ser especificada como 8080. Ou seja, caso o endereço público seja 18.231.21.93, por exemplo, a interface web estará disponível em 18.231.21.93:8080. A página de login será carregada como mostrado na figura 3.15:

Figura 3.15: Tela de login após a instalação da ThingsBoard.



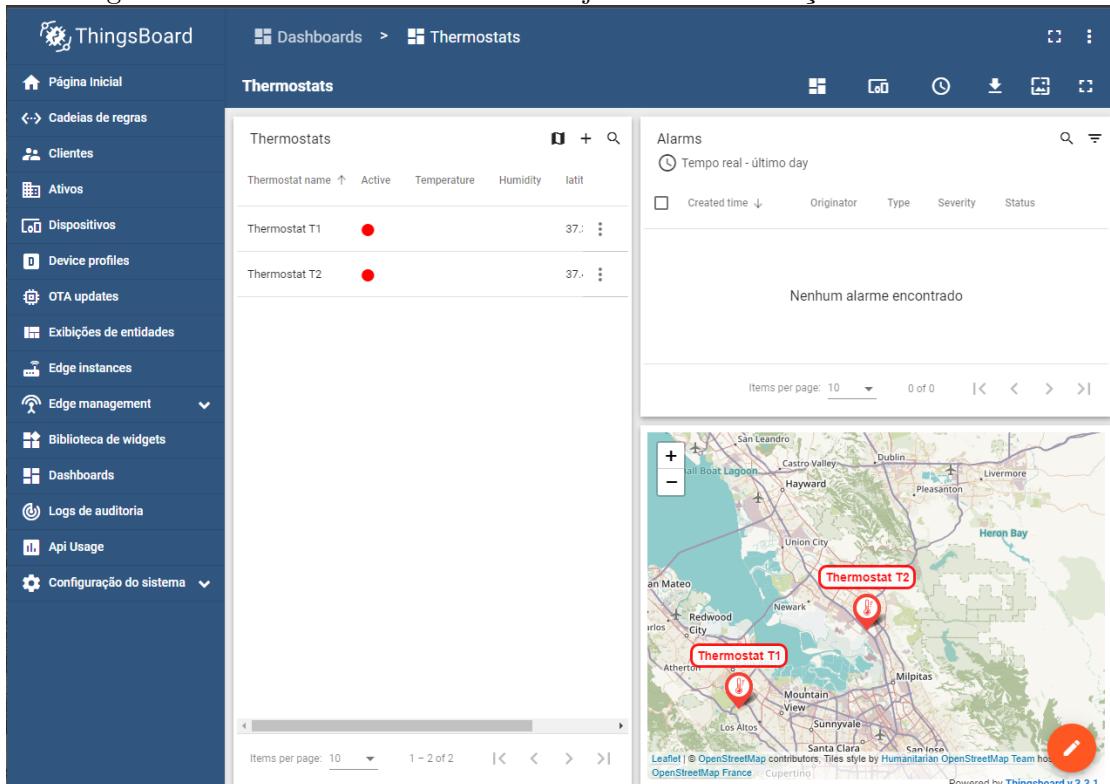
Fonte: Figura do Autor.

Para acessar a plataforma, basta utilizar o acesso padrão pré-definido:

- **Email:** tenant@thingsboard.org
- **Senha:** tenant

Com isso, já é possível utilizar e visualizar os exemplos demonstrativos já presentes na edição de comunidade. Por exemplo, junto com a instalação há um dashboard demonstrativo de monitoramento de termostatos com a visualização estática da localização de cada termostato, como pode ser visto na figura 3.16. Para acessá-lo basta clicar em "Dashboards -> Thermostats -> Abrir Dashboard" e então o dashboard poderá ser visualizado.

Figura 3.16: Dashboard demonstrativo já incluído na edição de comunidade.



Fonte: Figura do Autor.

Maiores detalhes na instalação da aplicação no Ubuntu Server estão disponíveis no guia de instalação fornecido pela própria ThingsBoard¹.

3.3.3 Configuração da ThingsBoard

Finalizada a instalação da ThingsBoard, é necessária a configuração da plataforma para o recebimento de dados do Gateway e também a criação do dashboard para a visualização desses dados ser apropriada no contexto do projeto.

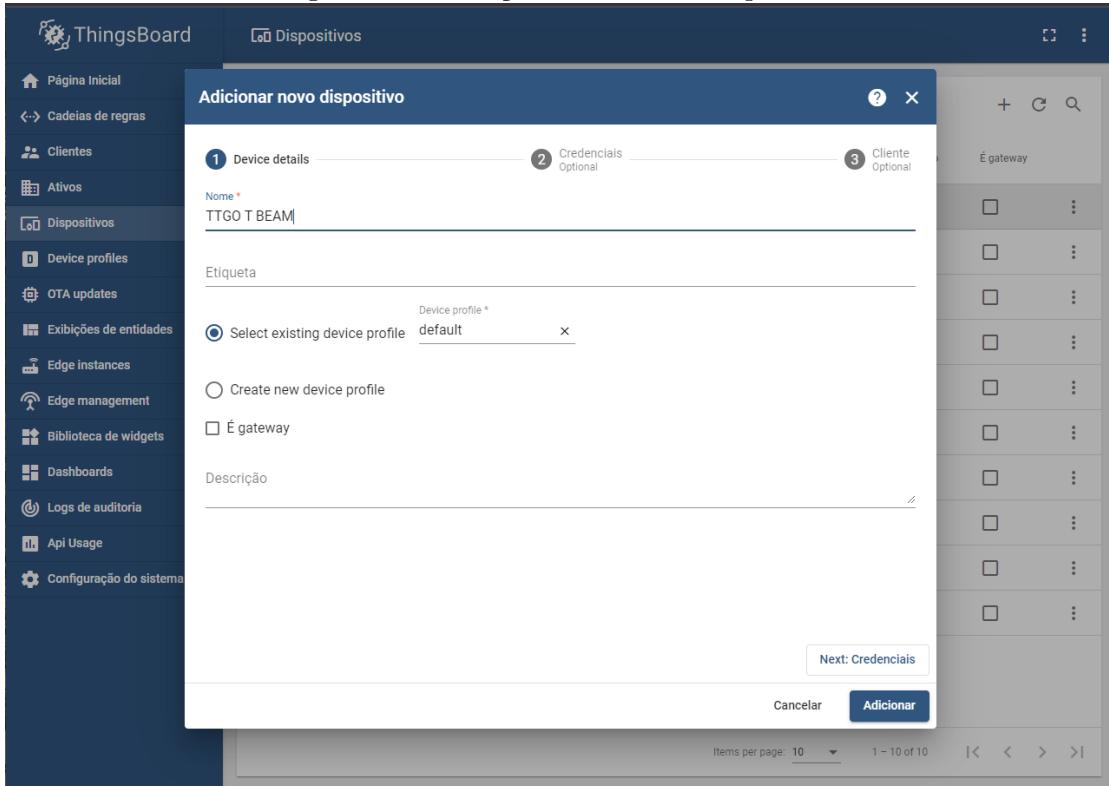
¹Guia de instalação ThingsBoard - Ubuntu Server

<https://thingsboard.io/docs/user-guide/install/ubuntu/>

3.3.3.1 Criação de novos dispositivos

Para receber dados de dispositivos não simulados, é necessário criar um novo dispositivo dentro da plataforma. Apesar do ThingsBoard se comunicar com o Gateway, essa comunicação é feita de forma que o Gateway simula as unidades de sensoriamento nele, guardando os tokens de acesso para cada dispositivo. Em resumo, os dispositivos são criados em relação às unidades de sensoriamento em si e não em relação ao Gateway. Para criar um novo dispositivo, é preciso acessar no menu lateral a opção "Dispositivos" e então clicar no símbolo +, localizado no canto superior direito da página. É necessária apenas a criação de um nome para o dispositivo a ser adicionado, no exemplo da figura 3.17, ele se chama "TTGO T BEAM":

Figura 3.17: Criação de um novo dispositivo.



Fonte: Figura do Autor.

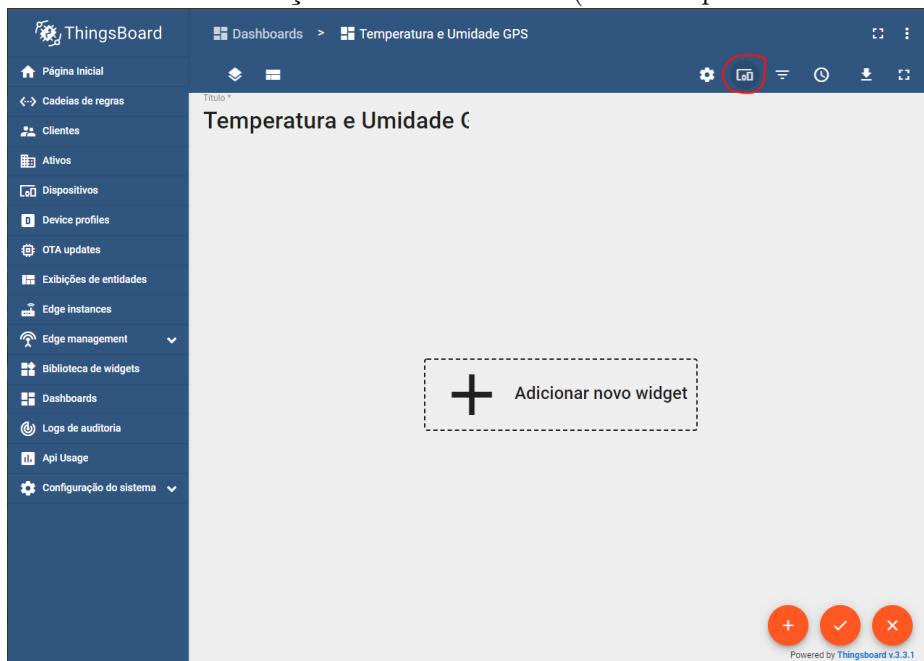
Após a criação do novo dispositivo, ele estará presente na lista da ThingsBoard. A informação a ser coletada nesse momento é o token de acesso definido para esse dispositivo. É através desse token que a plataforma saberá quais dados enviados pertencem a um dispositivo em específico. Para conseguir esse token, é preciso clicar no dispositivo recém criado e em seguida clicar em "Copiar token de acesso". Por exemplo, o token de acesso do dispositivo criado na etapa anterior é ShTYLZlubRk0492SGwu3. Esse token será utilizado no algoritmo descrito pela subseção 3.2.2.

3.3.3.2 Criação do dashboard

Apesar de existirem alguns dashboards de exemplo já incluídos na instalação, nenhum deles cumpre por completo os objetivos estabelecidos na subseção 1.2. Por isso, será criado um novo dashboard. Para começar o processo, basta acessar o menu "Dashboards" no menu de navegação e clicar no símbolo + no canto superior direito da tela. Apenas o título é obrigatório para a sua criação, as demais opções podem ser deixadas em branco. O dashboard criado aparecerá na lista e para editá-lo, basta selecioná-lo e clicar em "Abrir dashboard". Ele será então aberto e estará totalmente em branco como o esperado.

Os dispositivos criados na subseção 3.3.3.1 não podem ser referenciados diretamente pelo dashboard, sendo necessária a criação de um *Alias* (Apelido). Como poderão ser utilizadas múltiplas unidades de sensoriamento, o Alias deve se referir a uma lista de dispositivos. O primeiro passo para criar um novo Alias é entrar no modo de edição do dashboard, bastando clicar no ícone laranja (lápis) no canto inferior direito da tela. Posteriormente, será necessário clicar no ícone demonstrado pela figura 3.18:

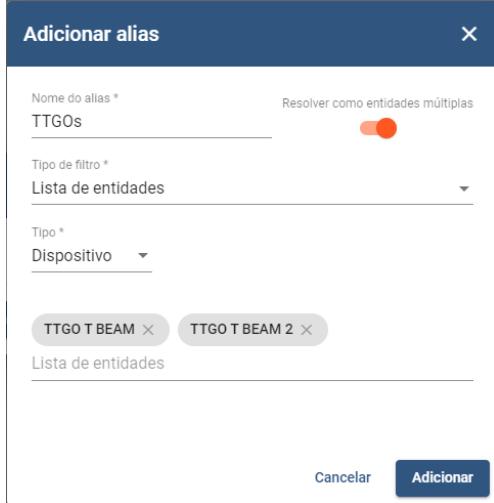
Figura 3.18: Acesso à criação de um novo Alias (marcado pelo círculo vermelho).



Fonte: Figura do Autor.

O novo Alias deve ser criado como indicado na figura 3.19. Caso seja adicionada uma nova unidade de sensoriamento, ela deve ser adicionada na lista de entidades, junto aos demais dispositivos:

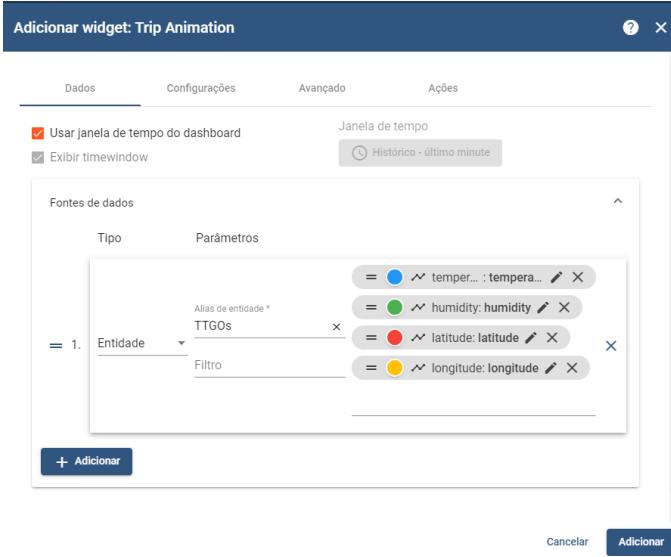
Figura 3.19: Configuração do Alias a ser utilizado.



Fonte: Figura do Autor.

Finalizada a configuração do Alias, pode-se adicionar os *widgets* (módulos) de visualização de dados. Para isso, o botão "Adicionar novo widget" pode ser clicado, o que abrirá a biblioteca de widgets. Diversos tipos de widgets estão disponíveis para visualizar os dados da forma mais apropriada. Um dos pacotes disponíveis é o de mapas, o qual será utilizado para visualizar a posição das unidades de sensoriamento. O widget "Trip animation" será utilizado, pois ele é o único que permite a visualização da localização e de outras fontes de dados (como temperatura e umidade) ao longo de uma linha do tempo. Feita a seleção do widget, uma janela se abrirá para selecionar a fonte de dados a ser utilizada, como ilustrado na figura 3.20.

Figura 3.20: Configuração da fonte de dados no widget.



Fonte: Figura do Autor.

Depois de selecionar a fonte de dados, algumas configurações avançadas devem ser feitas, com o propósito de não apenas visualizar a posição de cada unidade de sensoriamento, mas também os

dados de temperatura e umidade. Também será feita uma edição no rótulo dos pontos do mapa, com o propósito de facilitar a visualização. Na aba "Avançado", a opção "Use label function" deve ser marcada. No caixa de texto imediatamente abaixo, a seguinte função JavaScript deve ser colocada:

```
return '<span style="border: solid rgb(255, 0, 0); border-radius:  
        10px; color: solid rgb(255, 0, 0); background-color: #fff;  
        padding: 3px 5px; font-size: 14px">' + '${entityLabel}' + '</  
        span>';
```

A função acima modifica a estética do rótulo dos pontos no mapa, tornando-o mais visível. Ainda na aba "Avançado", é preciso substituir a estrutura da tooltip (legenda) pelo código abaixo, o qual permitirá a observação dos dados de temperatura e umidade associados a um ponto no mapa:

```
<b>${entityName}</b><br/><br/><b>Latitude:</b> ${latitude:7}<br/><b  
        &gt;>Longitude:</b> ${longitude:7}<br/><b>Temperature:</b> ${  
        &gt;temperature}<br/><b>Humidity:</b> ${humidity}<br/>  
<link-act name='History'>Data History</link-act>
```

Também é interessante a observação dos dados de temperatura e umidade ao longo do tempo, sem a visualização do mapa. O dashboard da ThingsBoard permite a criação de "Estados" de dashboard. Na prática, é uma página oculta que pode ser acessada somente a partir de um certo comando. Para criar um estado novo, o ícone circulado em vermelho na figura 3.21 deve ser pressionado:

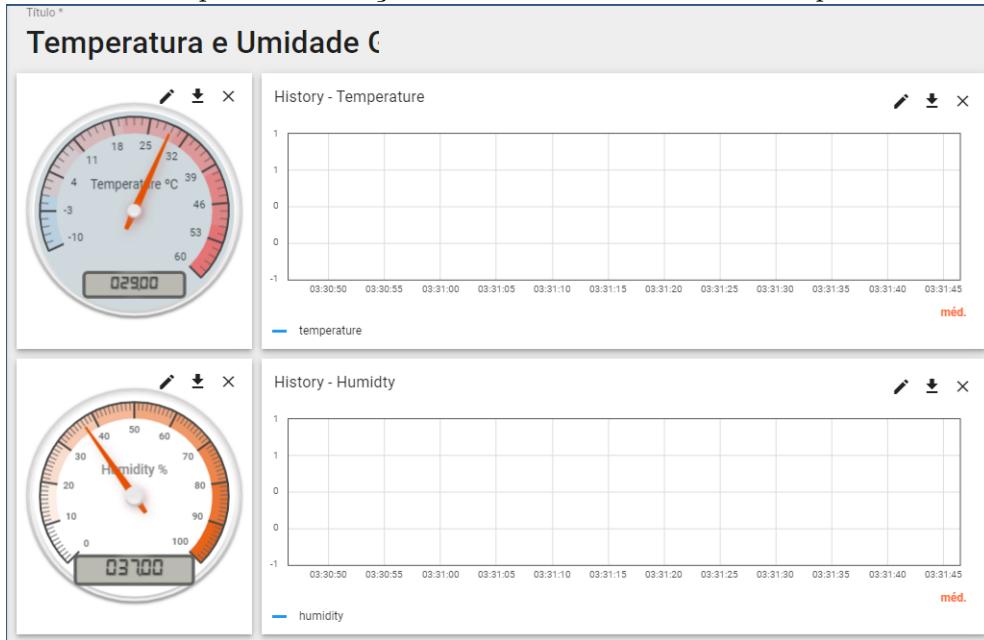
Figura 3.21: Acesso aos estados do dashboard.



Fonte: Figura do Autor.

Com o novo estado criado, ele pode ser acessado clicando em "default", no canto esquerdo da figura 3.21. Uma página em branco, similar à página quando o dashboard foi criado, aparecerá. Como descrito anteriormente, quaisquer tipos de widgets podem ser adicionados. No caso, para visualizar os dados de temperatura e umidade, foram adicionados mostradores analógicos instantâneos (*Radial Gauge*) e gráficos de linha para mostrar a variação dessas métricas ao longo do tempo, representados na figura 3.22.

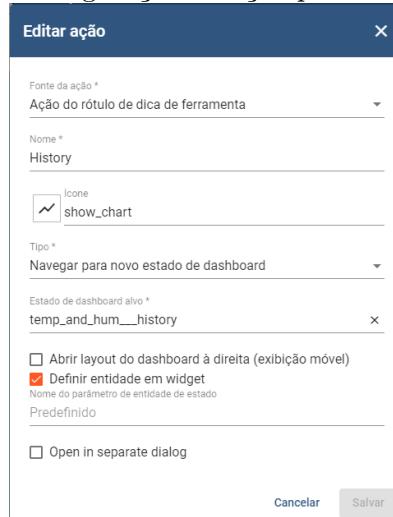
Figura 3.22: Estado para visualização detalhada das métricas de temperatura e umidade.



Fonte: Figura do Autor.

Apesar do estado estar criado e populado com widgets, ele estará inacessível, já que não há um link definido. Para isso, será necessário editar novamente o widget de mapa "Trip animation". A aba "Ações" deve ser selecionada e uma nova ação deve ser criada e configurada como exibido na figura 3.23.

Figura 3.23: Configuração da ação para acessar o estado.



Fonte: Figura do Autor.

A criação do dashboard está concluída, porém existem inúmeras possibilidades de customização e funcionalidades que não foram exploradas nesse projeto. Essas opções podem ser verificadas no guia da ThingsBoard sobre IoT dashboards¹.

¹Guia sobre IoT Dashboards <https://thingsboard.io/docs/user-guide/dashboards/>

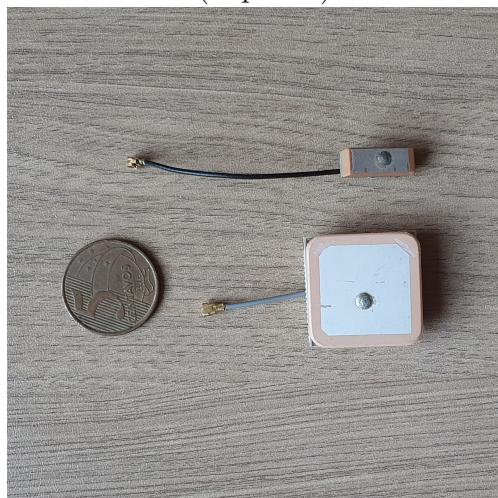
3.4 Problemas e dificuldades

Ao longo do desenvolvimento do projeto, foram enfrentadas diversas dificuldades que devem ser destacadas, com o intuito de que o leitor deste trabalho possa evitá-los, seja no desenvolvimento desse projeto ou de outro similar.

3.4.1 Sinal GPS fraco

A primeira dificuldade ao se implementar o código foi o fraco sinal GPS dentro de construções. Essa dificuldade já era esperada, porém foi intensificada devido à baixa performance da antena cerâmica disponibilizada junto à placa TTGO T-Beam, de forma que mesmo com a placa próxima à janela, não era possível adquirir dados válidos de localização. A figura 3.24 mostra um comparativo entre a antena da placa e uma antena que foi adquirida com um módulo GPS avulso (mesmo módulo utilizado na placa):

Figura 3.24: Antena GPS do módulo avulso (esquerda) e antena GPS da TTGO T-Beam (direita).



Fonte: Figura do Autor.

Após a troca pela antena maior, foi possível adquirir dados de localização com o módulo próximo à janela. Além disso, quando estava a céu aberto, a placa conseguia gerar dados de localização mais rapidamente.

3.4.2 CI de gestão de energia AXP192

Apesar do AXP192 trazer versatilidade de fornecimento de tensão para determinados módulos, controle de carregamento de bateria e a possibilidade da economia de energia, o uso do módulo é pouco documentado, principalmente pela fabricante LILYGO. Até mesmo ligar ou desligar a placa é algo que não é amplamente divulgado. A placa inicializa automaticamente quando alimentada pela USB ou pelos pinos de alimentação, mas não quando uma bateria é inserida.

Quatro unidades da TTGO T-Beam foram adquiridas para o projeto, porém uma delas se

tornou inutilizável para atender os objetivos após ser alimentada com um carregador de celular moderno, com tecnologia de carregamento rápido. Depois do acontecimento, a placa funciona parcialmente, porém não é mais possível se comunicar com o IC AXP192. Sendo assim, o fornecimento de tensão para os módulos GPS e LoRa foi interrompido. Essa situação indica que a placa não possui uma proteção contra picos de tensão.

Ainda no tópico sobre falta de proteções em relação ao fornecimento de energia, outra placa foi danificada depois da inserção ao contrário da bateria 18650. A placa ainda funciona normalmente, mas não possui capacidade de funcionamento com baterias, sendo necessária sua alimentação através da porta micro USB. Isso indica que a placa não possui proteção contra inversão acidental de alimentação, problema que já foi corrigido na versão 1.1 da placa, segundo a fabricante LILYGO¹.

3.4.3 Incompatibilidade da tela OLED com o módulo AXP192

A tela OLED utiliza o barramento de comunicação *I²C* (*Inter-Integrated Circuit*), sendo o mesmo utilizado pelo módulo AXP192. Na unidade de sensoriamento foi detectado um conflito de endereços nesse barramento, de forma que para que o módulo fosse ligado e funcionasse com sucesso, era necessário desconectar a tela, ligar a placa com a tela desconectada e somente após a sua inicialização, a tela poderia ser reconectada. A solução, contudo, foi simples e consistiu na utilização de um barramento separado para a tela, por meio de outros pinos.

3.4.4 Aferição da tensão da bateria 18650

A informação da tensão da bateria utilizada na unidade de sensoriamento poderia permitir a estimativa de sua carga, inclusive em números percentuais, de forma que permitisse uma visualização semelhante ao exemplo descrito pela figura 2.5. De acordo com o repositório da biblioteca do módulo AXP192², há um método para a obtenção dessa tensão: *getBatteryVoltage()*, o qual deveria retornar um valor em ponto flutuante. Contudo, ao testar esse método, o único valor retornado era zero. Uma solução para resolver esse problema não foi encontrada e a funcionalidade teve que ser removida do dashboard.

¹TTGO T-Beam V1.1 http://www.lilygo.cn/prod_view.aspx?TypeId=50060&Id=1164&FId=t3:50060:3

²Repositório da biblioteca AXP192 https://github.com/tanakamasayuki/I2C_AXP192

Capítulo 4

Testes e resultados

Neste capítulo serão discutidas as metodologias de testes utilizadas, além da visualização dos resultados obtidos com o projeto. Também serão discorridas as principais dificuldades e obstáculos encontrados ao longo dos testes.

Um esclarecimento importante para esta seção é que todos os testes realizados neste capítulo foram feitos ao longo do período de pandemia (COVID-19) e o acesso a recursos e locais da UnB estava limitado.

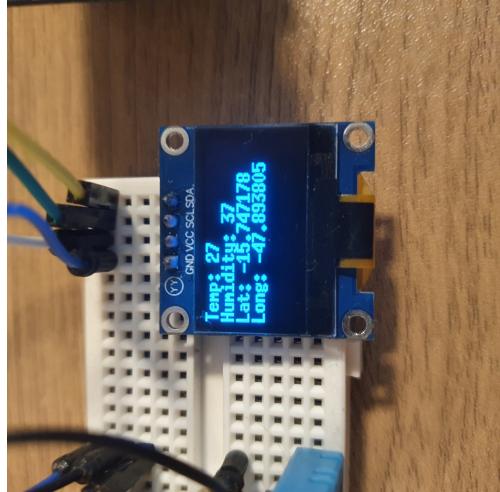
4.1 Testes na unidade de sensoriamento

O código foi desenvolvido e montado por partes, sendo a primeira etapa a obtenção dos dados de temperatura e umidade do sensor DHT 11, o que foi atingido sem grandes problemas utilizando os procedimentos descritos no capítulo anterior.

Em seguida, foi testada a obtenção desses dados junto aos dados de localização, utilizando o exemplo disponibilizado pela fabricante. Apesar do problema descrito pela subseção 3.4.1, a substituição da antena resolveu o problema e os dados de localização foram obtidos com sucesso. Todos os testes foram feitos com o módulo tendo visão direta a céu aberto.

A primeira coleta dos dados do módulo GPS após um período sem uso ou após a translocação do dispositivo para uma nova localidade é chamada de TTFF (*Time to first fix*), ou seja o tempo para a primeira aquisição de dados. Nos testes feitos nessa etapa, o TTFF foi estimado em torno de três minutos. Quando a placa é desligada e religada pouco tempo depois, esse tempo de aquisição de dados cai drasticamente, para de 10 a 30 segundos. É importante destacar que quando o módulo consegue ter um campo de visão para pelo menos um satélite GPS, um LED vermelho (descrito como LED2 pela figura 3.4) pisca periodicamente. Os dados são exibidos como mostrado na figura 4.1.

Figura 4.1: Dados de temperatura, umidade e localização obtidos pela unidade de sensoriamento.



Fonte: Figura do Autor.

Os dados obtidos pelo módulo GPS foram comparados com a localização fornecida por um celular, através do aplicativo de Mapas. No geral, todas as aferições a céu aberto apresentaram precisão satisfatória, com erros de aproximadamente 5 metros em relação a aferição feita pelo celular, o que não prejudica o atingimento dos objetivos do projeto.

Posteriormente, foi feito o teste do envio dos dados via LoRa, o qual será discutido na seção 4.2. É válido destacar que duas funcionalidades extras (em relação ao código base disponibilizado pelo fabricante) do LoRa foram habilitadas com o intuito de garantir a integridade da comunicação entre os módulos. A primeira, implementada através do método *setSyncWord(0xF3)*, garante que o Gateway ignore a leitura de outras possíveis mensagens LoRa, definindo uma chave de identificação entre o Gateway e as unidades de sensoriamento. A segunda, implementada através do método *enableCrc()*, habilita o CRC (*Cyclic redundancy check*) que possibilita a conferência da integridade dos dados ao chegarem no Gateway.

O último teste feito na unidade de sensoriamento é a verificação da duração da bateria. Um parâmetro que impacta diretamente nessa duração é o tempo entre as aferições e envio das variáveis climáticas, ou frequência de amostragem. O código foi testado com esse parâmetro definido para 5 segundos. Com esse intervalo, a placa permaneceu funcional durante cerca de 5 horas contínuas de uso. Caso esse tempo seja reajustado para um valor menor, a bateria terá uma durabilidade menor, já que os módulos GPS e LoRa serão utilizados com maior frequência. De forma semelhante, caso seja ajustado para um tempo maior, a bateria terá uma durabilidade maior.

4.2 Testes com o Gateway

O Gateway foi desenvolvido e testado em partes, de forma similar à unidade de sensoriamento. Como foi utilizado o FreeRTOS, os testes foram feitos para cada tarefa criada, as quais estão descritas na figura 3.7. A primeira etapa realizada foi o teste do recebimento das mensagens via

LoRa, a qual foi alcançada com sucesso. A mensagem JSON foi recebida com êxito a partir do exemplo desenvolvido pela fabricante. Os dados foram verificados através do display OLED, como pode ser visto na figura 4.2.

Figura 4.2: Visualização dos dados enviados pela unidade de sensoriamento no Gateway.



Fonte: Figura do Autor.

A próxima etapa, já na tarefa de processamento de mensagens, foi a extração dos parâmetros a partir da mensagem JSON, a qual também foi executada sem problemas. A comunicação MQTT com a plataforma ThingsBoard, por fim, foi também alcançada com êxito. Até mesmo antes da criação do dashboard, é possível verificar se os dados estão sendo enviados corretamente para a plataforma, através do menu "Dispositivos", selecionando uma das unidades de sensoriamento criadas. No menu do dispositivo, é possível verificar os últimos dados enviados através da aba "Última telemetria", exemplificado através da figura 4.3.

Figura 4.3: Verificação da comunicação entre o Gateway e a plataforma ThingsBoard.

Horário da última atualização Chave ↑	Valor
2021-11-07 04:38:28	humidity
2021-11-07 04:38:28	latitude
2021-11-07 04:38:28	longitude
2021-11-07 04:38:28	temperature
2021-11-07 04:38:28	ts

Fonte: Figura do Autor.

Um importante passo nessa etapa que não pode ser testada de forma pragmática é o alcance da comunicação LoRa. Os testes da seção 4.3 vão possibilitar melhor a demonstração de pontos cegos e falhas dessa comunicação, porém a realização dos testes foi feita em um ambiente com várias possíveis interferências, principalmente a presença de obstáculos físicos.

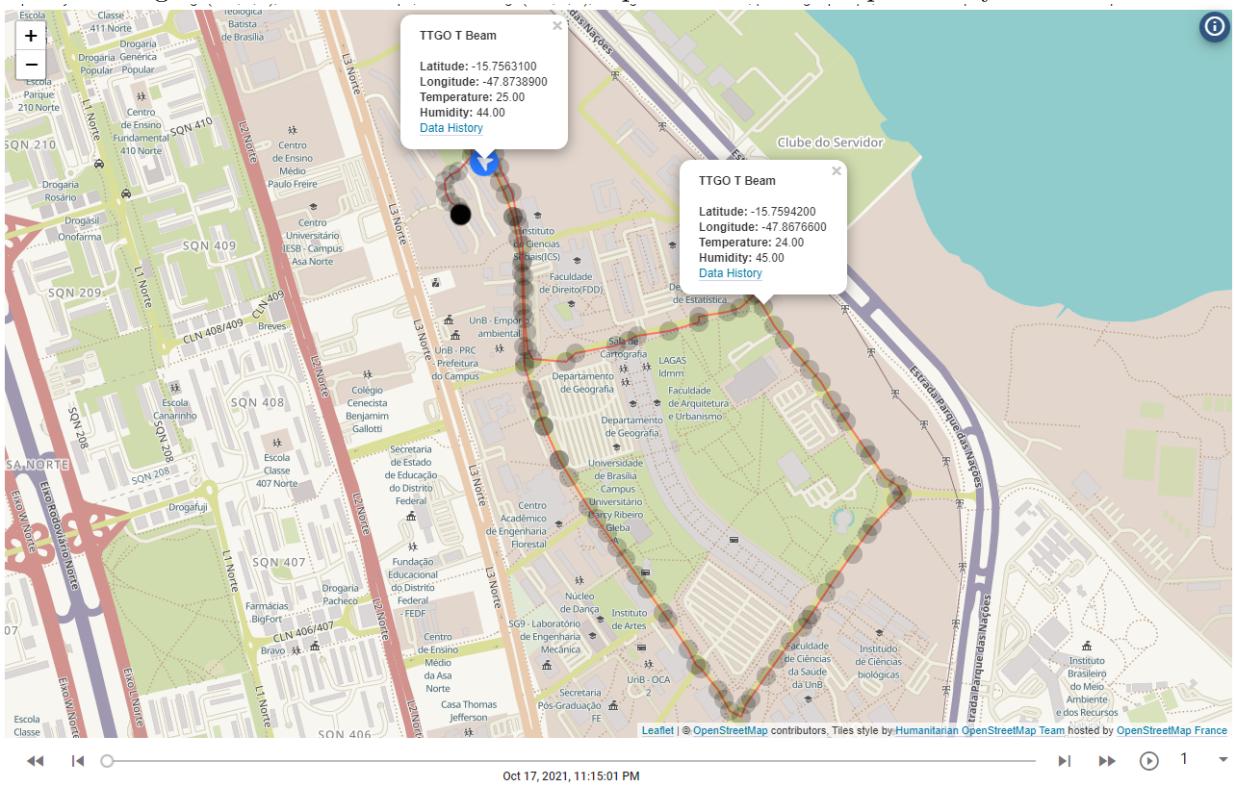
4.3 Testes com a ThingsBoard

Para finalizar o capítulo de testes, a plataforma ThingsBoard foi verificada com o intuito de garantir a visualização dos dados.

- **Teste com apenas uma unidade de sensoriamento e Gateway próximo à unidade**

Primeiramente, para excluir outras possibilidades de erro no projeto, foram realizados testes com apenas uma unidade de sensoriamento. O acesso à internet no Gateway é imprescindível para o funcionamento do projeto como um todo, então o acesso foi gerado através do roteamento da rede do celular (3G/4G). Um trajeto foi feito de automóvel, a aproximadamente 40 km/h, pelo Campus Darcy Ribeiro da Universidade de Brasília. Tanto o Gateway quanto a unidade de sensoriamento estavam dentro do automóvel. O clima era chuvoso e o teste foi executado no período noturno. Os dados foram aferidos e enviados com sucesso, como pode ser visualizado na figura abaixo. Na figura 4.4, a linha laranja mostra o percurso realizado e os círculos indicam um ponto de aferição:

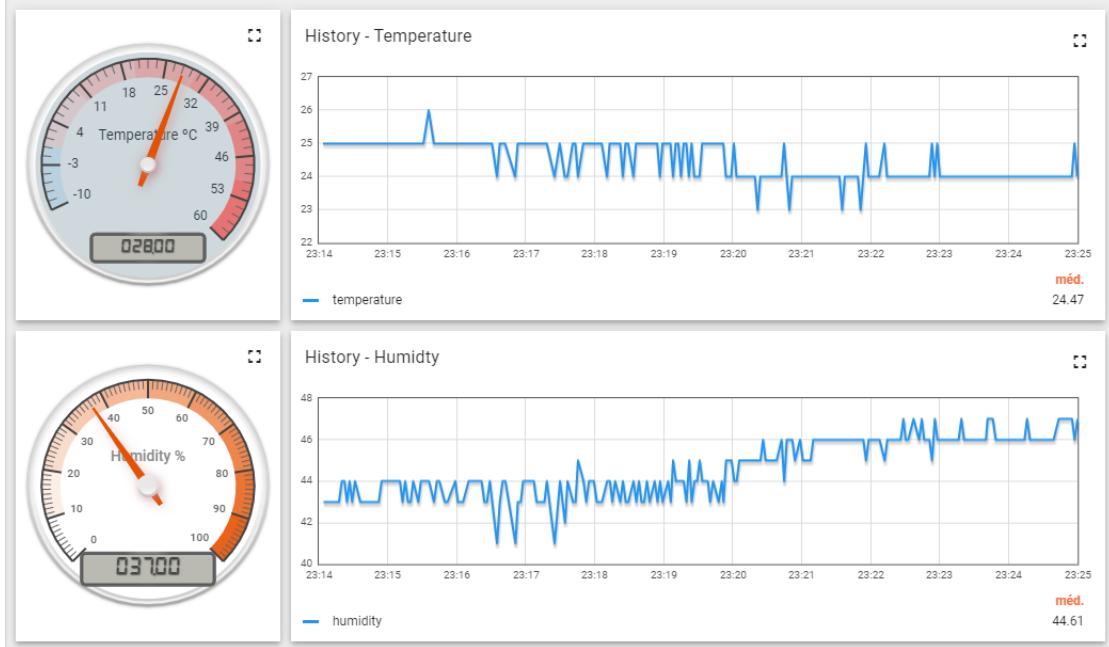
Figura 4.4: Unidade de sensoriamento percorrendo o Campus Darcy Ribeiro.



Fonte: Figura do Autor.

Ao clicar em "Data History", é possível visualizar os dados no estado criado anteriormente, como demonstrado pela figura 4.5.

Figura 4.5: Gráficos de temperatura e umidade ao longo do tempo.



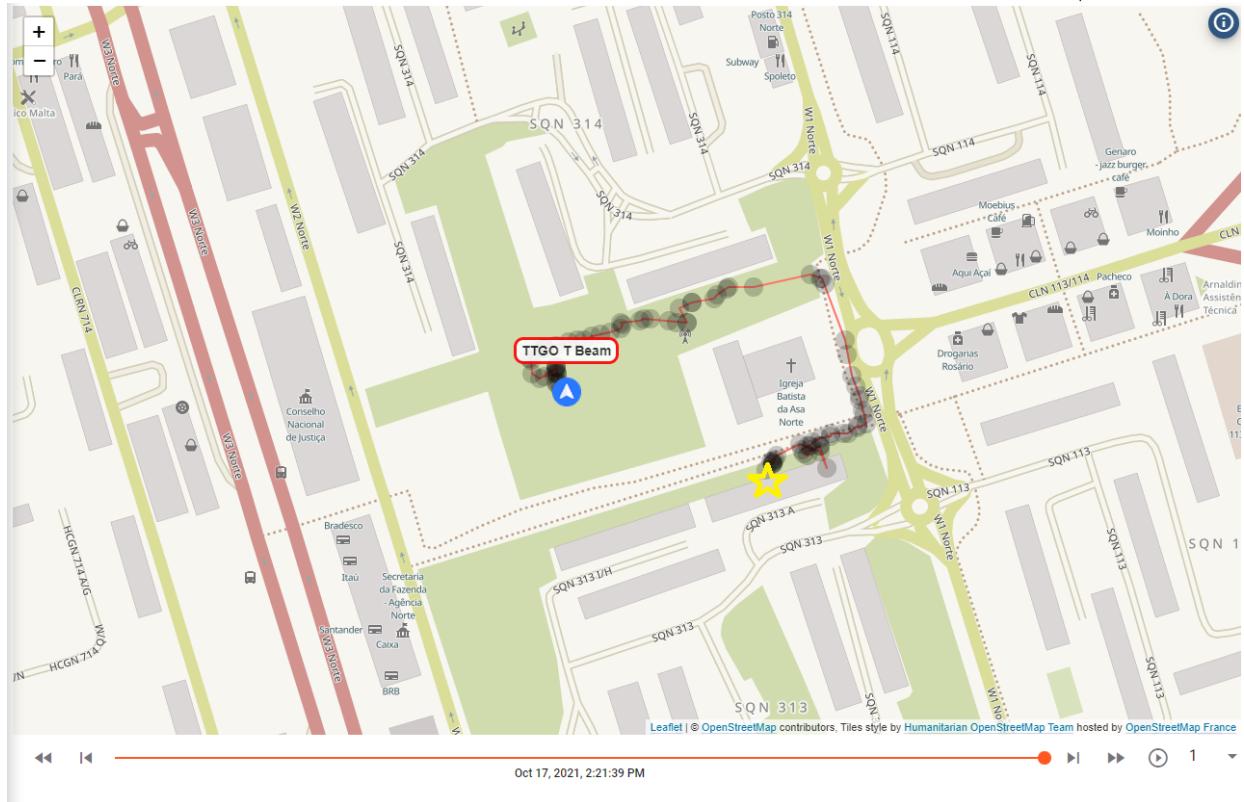
Fonte: Figura do Autor.

O widget escolhido para o dashboard permite a reprodução do percurso através da barra inferior, na qual pode-se percorrer a linha do tempo e dar início a reprodução do caminho. Também é possível acelerar essa reprodução.

- **Teste com apenas uma unidade de sensoriamento e Gateway fixo remoto**

O segundo teste focou no alcance da comunicação LoRa, além da sua suscetibilidade a interferências físicas e foi realizado na Entrequadra Norte 313/314, onde há uma igreja e um campo aberto, além de outras construções e vegetações. Ao longo desse trajeto, o Gateway permaneceu em uma janela (quarto andar) voltada para o percurso, aproximadamente onde está marcada a estrela amarela no mapa da figura 4.6:

Figura 4.6: Unidade de sensoriamento percorrendo a Entrequadra Norte 313/314.



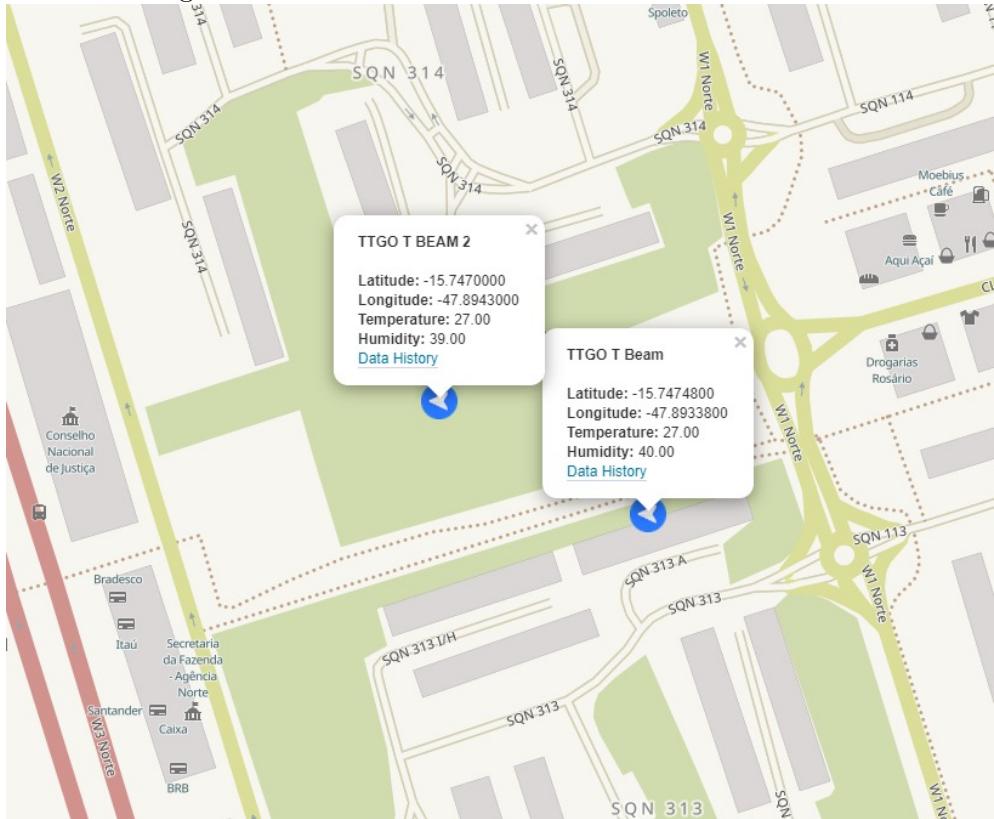
Fonte: Figura do Autor.

O teste foi feito a pé, em velocidade de caminhada, com clima ensolarado e sem nuvens. Em determinados momentos, a igreja ficou entre a unidade de sensoriamento e o Gateway. Esses momentos podem ser conferidos ao longo da linha laranja da figura 4.6, onde percebe-se que há a existência de pontos cegos sem aferições (não há círculos escuros).

- **Teste com duas unidades de sensoriamento estáticas**

O próximo teste efetuado foi com duas unidades de sensoriamento, estando as duas fixas, sem se movimentarem. O objetivo dessa verificação era garantir o funcionamento do Gateway quando duas unidades de sensoriamento estivessem transmitindo dados. Ele foi realizado em condições similares às condições do teste anterior e os resultados podem ser verificados na figura 4.7.

Figura 4.7: Teste com duas unidades de sensoriamento.

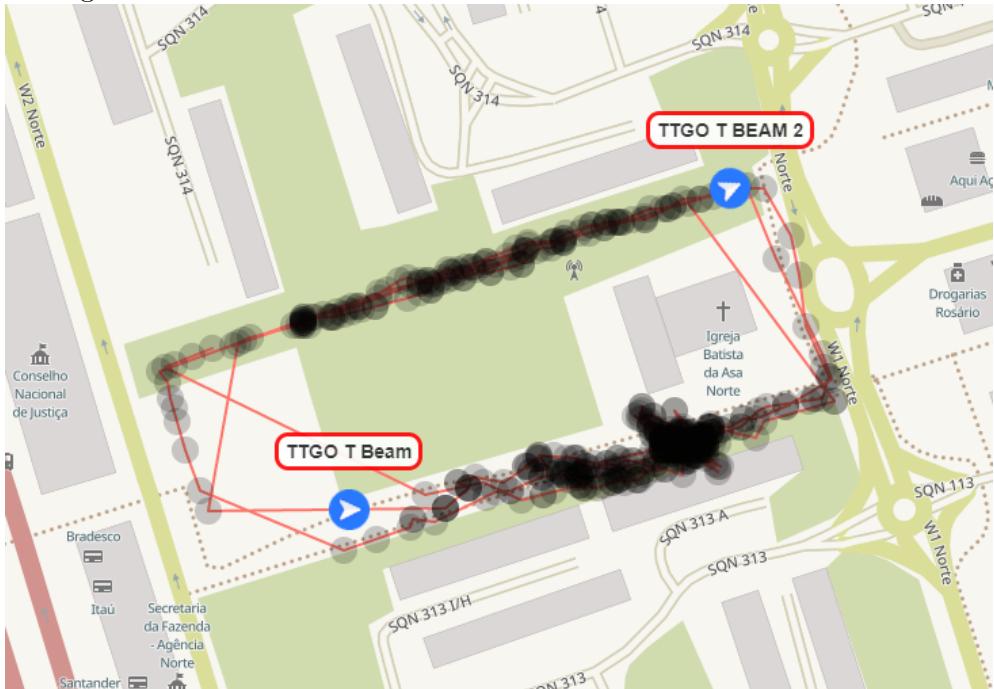


Fonte: Figura do Autor.

- **Teste com duas unidades de sensoriamento em movimento**

O último teste foi executado também no percurso da Entrequadra Norte 313/314, porém agora com as duas unidades de sensoriamento em movimento. Um padrão de caminhada não foi estabelecido, já que várias voltas foram dadas ao longo do percurso. Os trajetos das duas unidades, apesar de serem os mesmos, foram feitos em sentidos opostos e se sobrepunderam, com o intuito de verificar possíveis interferências entre as placas.

Figura 4.8: Teste com duas unidades de sensoriamento em movimento.



Fonte: Figura do Autor.

Como o trajeto foi percorrido diversas vezes, é mais fácil e confiável a identificação dos locais onde ocorreram falhas na comunicação LoRa, sendo esses locais onde há a menor densidade de pontos de aferição ao longo da linha laranja na figura 4.8.

Capítulo 5

Conclusões

O presente trabalho objetivou a implementação de um sistema que permitisse a visualização de métricas ambientais ao longo de um trajeto. A avaliação foi feita pela consistência e confiabilidade dos dados. Ao final, são propostos trabalhos futuros para expandir e melhor o projeto

5.1 Localização

Um dos objetivos principais do projeto é a obtenção dos dados de localização, com o intuito de rastrear o monitoramento de parâmetros ambientais. Pode-se então concluir que esse objetivo foi alcançado com sucesso, mas há limitações de uso. O projeto idealmente não deve ser utilizado em construções fechadas, já que o uso em tais locais não vai gerar dados válidos de localização. Uma alternativa para essa restrição será discutida na subseção 5.5.1.

5.2 Comunicação

A comunicação entre o Gateway e a plataforma ThingsBoard teve sucesso absoluto através do uso do SDK disponibilizado pela ThingsBoard e o protocolo MQTT. Entretanto, é válido ressaltar que o Gateway deve estar conectado à internet (ou alguma rede que o ThingsBoard consiga acessar) durante o uso do projeto, ou os dados serão perdidos.

Já a comunicação entre as unidades de sensoriamento e o Gateway apresentou falhas devido aos obstáculos físicos encontrados ao longo dos percursos, como pôde ser observado nas figuras 4.6 e 4.8. O funcionamento foi pleno apenas quando as placas estavam a uma curta distância do Gateway ou quando a linha de visão entre as unidades tinha poucos obstáculos.

5.3 Visualização de dados

A ThingsBoard apresentou-se extremamente customizável ao longo do desenvolvimento. Sua flexibilidade de escolhas nas montagens dos dashboards, configuração dos widgets e no processamento de mensagens confirmou sua capacidade de atender aos requisitos do projeto.

Contudo, a instância selecionada na subseção 3.3.1.2 se mostrou subprovisionada, ou seja, com capacidade computacional abaixo do necessário para executar a aplicação como um todo. Após um certo período de tempo, a plataforma simplesmente travava e ficava inacessível, sendo necessária a reinicialização da instância no console Amazon EC2. De toda forma, a plataforma pode ser instalada em uma instância com maior capacidade computacional (por exemplo, a t3.micro) ou em alguma máquina física disponível, sendo necessário apenas configurar o acesso dessa máquina através da rede acessada por ela.

5.4 Algoritmos implementados

Os algoritmos desenvolvidos, tanto para as unidades de sensoriamento quanto para o Gateway, foram colocados a prova de uso e se demonstraram capazes de realizar suas tarefas. Os códigos se mantiveram funcionais durante longos períodos de teste (mais de 24h de funcionamento), demonstrando potencial de utilizar o projeto para monitoramentos contínuos ou de longas durações, caso seja necessário.

5.5 Perspectivas Futuras

5.5.1 Melhorias na localização

Com as dificuldades encontradas na obtenção dos dados de localização (subseção 3.4.1 e seção 5.1), uma possibilidade de melhoria no projeto é a utilização da versão sucessora da placa utilizada no projeto, a LILYGO TTGO T-Beam V1.1¹. Ela utiliza o módulo GPS ublox NEO-M8N², o qual é capaz de acessar até três redes GNSS de forma concorrente e que também apresenta compatibilidade com as tecnologias GLONASS, Galileo e BeiDou. Com a utilização de múltiplas GNSS, a obtenção de dados pode ser obtida de forma mais confiável, mais rápida e com maior disponibilidade.

¹LILYGO TTGO T-Beam V1.1

http://www.lilygo.cn/prod_view.aspx?TypeId=50060&Id=1164&FId=t3:50060:3

²ublox NEO-M8N <https://www.u-blox.com/en/product/neo-m8-series>

5.5.2 Melhorias na comunicação entre as unidades de sensoriamento e o Gatemay

Como a comunicação utilizada se mostrou ineficiente em certas situações, algumas alternativas podem ser buscadas. A mais simples talvez seja uma mudança no esquema do projeto, eliminando o uso de Gateways e tornando cada unidade de sensoriamento autônoma. Em contrapartida, é necessário achar uma maneira dessa unidade se manter conectada a uma rede constantemente, o que pode ser alcançado com o uso de roteamento de internet a partir de celulares, como foi feito no primeiro teste da seção 4.3. Há a possibilidade também do uso de módulos que se conectam a rede de celular para fazer o envio desses dados.

Outra opção é a utilização de módulos LoRa mais potentes e com antenas mais sensíveis, porém deve-se avaliar o impacto na vida útil da bateria (a qual também pode ser facilmente substituída por uma de maior capacidade).

5.5.3 Melhorias na visualização de dados

Ao utilizar mais de uma unidade de sensoriamento, caso o trajeto das duas unidades seja coincidente em um intervalo de tempo, a visualização pode ficar confusa e prejudicada. A ThingsBoard permite, através das configurações avançadas do módulo Trip Animation, que a cor da linha da trajetória e dos pontos de medição seja customizada com base nos dados enviados por telemetria. Com essa modificação, será mais fácil de distinguir os percursos e pontos de coleta. Uma solução parcial é a criação de um widget para cada unidade de sensoriamento, segregando os mapas quando houver muita sobreposição entre os trajetos.

5.5.4 Otimização do projeto

Como o foco do projeto foi gerar uma prova de conceito, a otimização dos códigos implementados ficou em segundo plano. Os algoritmos podem ser otimizados tanto para fins de consumo energético quanto para performance do código. Por exemplo, não há a necessidade do fornecimento de energia constante para o módulo LoRa e há muito espaço para a otimização do código Gateway em relação às tarefas implementadas.

Outra otimização que pode ser implementada é a calibragem dos sensores, preferencialmente através da plataforma ThingsBoard (*Rule Engine*). Isso permitirá o ajuste dessa calibragem de forma remota, sem alterar o código das unidades.

5.5.5 Integração com o projeto Mochila Bioclimática

Como foi discutido na introdução, o projeto visa estender as funcionalidades do projeto Mochila Bioclimática. A integração do trabalho atual com a Mochila pode ser feita de inúmeras formas, por exemplo, mantendo a aferição na plataforma escolhida pelo projeto e passando os dados de medição via comunicação serial. Deve-se atentar principalmente à discrepâncias entre os níveis

lógicos das placas.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] VANTAGE Vue® Wireless Weather Station — Davis Instruments. Disponível em: <<https://www.davisinstruments.com/products/vantage-vue-wireless-weather-station>>.
- [2] SANTERRE, R. et al. Single point positioning using gps, glonass and beidou satellites. *Positioning*, v. 05, n. 04, p. 107–114, 2014. ISSN 2150-850X. Disponível em: <<http://dx.doi.org/10.4236/pos.2014.54013>>.
- [3] FENG, X.; YAN, F.; LIU, X. Study of Wireless Communication Technologies on Internet of Things for Precision Agriculture. *Wireless Personal Communications*, v. 108, n. 3, p. 1785–1802, 2019. ISSN 1572834X.
- [4] ALVES, E. D. L. et al. A temperatura do ar e umidade absoluta em cidade de pequeno porte: características espaciais e temporais. *Acta Scientiarum. Human and Social Sciences*, v. 38, n. 2, p. 219, 2016. ISSN 1679-7361.
- [5] ROMERO, M. A. B. et al. Mudanças climáticas e ilhas de calor urbanas. Universidade de Brasília, Faculdade de Arquitetura e Urbanismo; ETB, 2019.
- [6] MIRANDA, R. A. C. de; PEREIRA, F. R. Desenvolvimento de plataforma para monitoramento "automatizado"de dados termo-pluviométricos. *Geosul*, v. 26, n. 52, p. 129, 2012. ISSN 0103-3964.
- [7] MUCELIN, C. A.; BELLINI, M. Lixo e impactos ambientais perceptíveis no ecossistema urbano. *Sociedade & Natureza*, v. 20, n. 1, p. 111–124, 2008. ISSN 0103-1570.
- [8] AZEVEDO, T. et al. Ilhas de calor e aedes aegypti: um estudo preliminar para a cidade de santa bárbara d'oeste, sp – bra, utilizando sensoriamento remoto. In: . [S.l.: s.n.], 2012.
- [9] KOENIGSBERGER, O. H. *Viviendas y edificios en zonas cálidas y tropicales*. Madrid: Parainfo, 1977. 328 p. ISBN 9788428308854.
- [10] ESSENTIAL Climate Variables | World Meteorological Organization. Disponível em: <<https://public.wmo.int/en/programmes/global-climate-observing-system/essential-climate-variables>>.
- [11] ARAÚJO, T. C. de. Sobre a qualidade dos dados em sensoriamento de baixo custo para caracterização ambiental de espaços urbanos. mar 2020. Disponível em: <<http://hdl.handle.net/1822/72042>>.

- [12] BURKE, J. et al. Participatory sensing. In: *In: Workshop on World-Sensor-Web (WSW'06): Mobile Device Centric Sensor Networks and Applications.* [S.l.: s.n.], 2006. p. 117–134.
- [13] WEEWX: open source weather software. Disponível em: <<https://www.weewx.com/>>.
- [14] AMBIENT Weather WS-2902C Smart Weather Station with WiFi Remote Monitoring and Alerts. Disponível em: <<https://ambientweather.com/amws2902.html>>.
- [15] SILVA, R. B. et al. Estações meteorológicas de código aberto: Um projeto de pesquisa e desenvolvimento tecnológico. *Revista Brasileira de Ensino de Física*, v. 37, n. 1, 2015. ISSN 01024744.
- [16] MOURA, R. M. de. Estação meteorológica de baixo custo: Uma contribuição para o monitoramento meteorológico das cidades. v. 1, n. 1, p. 1–8, 2018. ISSN 09240136. Disponível em: <<http://dx.doi.org/10.1016/j.cirp.2016.06.001>>
- [17] ROMERO, M. A. B. et al. Instrumentação para medições na escala microclimática:: uma proposta de mochila bioclimática. *Paranoá: cadernos de arquitetura e urbanismo*, n. 26, p. 96–105, 2020. ISSN 1679-0944.
- [18] SANTOS, B. P. et al. Internet das coisas: da teoria à prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, v. 31, p. 16, 2016.
- [19] BARROCA, D. et al. Gnss: status, modelagem atmosférica e métodos de posicionamento gnss: status, atmospheric modeling and positioning methods. *Rev. Bras. Geom.*, v. 1, n. 1, n. 7.
- [20] LI, X. et al. Precise positioning with current multi-constellation global navigation satellite systems: Gps, glonass, galileo and beidou. *Scientific Reports 2015 5:1*, Nature Publishing Group, v. 5, n. 1, p. 1–14, feb 2015. ISSN 2045-2322. Disponível em: <<https://www.nature.com/articles/srep08328>>.
- [21] COJOCARU, S. et al. GPS-GLONASS-GALILEO: A dynamical comparison. *Journal of Navigation*, v. 62, n. 1, p. 135–150, 2009. ISSN 03734633.
- [22] JEREZ, G. O.; ALVES, D. B. M. GLONASS: Revisão teórica e estado da arte. *Revista Brasileira de Geomática*, v. 6, n. 2, p. 155, 2018. ISSN 2317-4285.
- [23] ASKSSENSORS IoT platform | The easiest application to manage your IoT devices and data in the cloud. Disponível em: <<https://asksensors.com/index.html>>.
- [24] THINGSBOARD - Open-source IoT Platform. 2021. Disponível em: <<https://thingsboard.io/>>.
- [25] GPS Tracking using ESP32 and IoT Platform over MQTT | AskSensors Blog. Disponível em: <<https://blog.asksensors.com/iot-cloud-based-gps-tracking-esp32-gps-neo-6m-module/>>.
- [26] GPS data upload and visualization using LinkIt ONE and ThingsBoard | ThingsBoard Community Edition. Disponível em: <<https://thingsboard.io/docs/samples/linkit-one/gps/>>.

- [27] THINGSBOARD architecture | ThingsBoard Community Edition. Disponível em: <<https://thingsboard.io/docs/reference/>>.
- [28] THINGSBOARD IoT Platform deployment scenarios | ThingsBoard Community Edition. Disponível em: <<https://thingsboard.io/docs/reference/iot-platform-deployment-scenarios/>>.
- [29] RULE Engine Overview | ThingsBoard Community Edition. Disponível em: <<https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/>>.
- [30] ANATEL - Resolução nº 671, de 3 de novembro de 2016. Disponível em: <<https://informacoes.anatel.gov.br/legislacao/resolucoes/2016/911-resolucao-671anexoI>>.
- [31] GITHUB - thingsboard/thingsboard-arduino-sdk: Arduino libarary to connect with ThingsBoard IoT Platform. Disponível em: <<https://github.com/thingsboard/thingsboard-arduino-sdk>>.
- [32] GITHUB - LilyGO/TTGO-T-Beam. Disponível em: <<https://github.com/LilyGO/TTGO-T-Beam>>.
- [33] SELF-HOSTED setup using AWS EC2 instance | ThingsBoard Community Edition. Disponível em: <<https://thingsboard.io/docs/user-guide/install/cluster/aws-self-hosted-setup/>>.

ANEXOS

ANEXO I

Códigos utilizados

I.1 Algoritmo da Unidade de sensoriamento

I.1.1 *utilities.h* - Conjunto de parâmetros

```
1 #define LoRa_frequency 915E6
2
3 #define UNUSE_PIN (0)
4
5 #define TIME_INTERVAL 5000 //Tempo em milisegundos entre amostras
6
7 #define GPS_RX_PIN 34
8 #define GPS_TX_PIN 12
9 #define BUTTON_PIN 38
10 #define BUTTON_PIN_MASK GPIO_SEL_38
11 #define I2C_SDA 21
12 #define I2C_SCL 22
13 #define I2C_SDA2 2
14 #define I2C_SCL2 13
15 #define PMU_IRQ 35
16
17 #define RADIO_SCLK_PIN 5
18 #define RADIO_MISO_PIN 19
19 #define RADIO_MOSI_PIN 27
20 #define RADIO_CS_PIN 18
21 #define RADIO_DIO_PIN 26
22 #define RADIO_RST_PIN 23
23 #define RADIO_DIO1_PIN 33
24 #define RADIO_BUSY_PIN 32
25
```

```

26 #define GPS_BAUD_RATE 9600
27
28 #define DHTPIN 15 // Pino conectado ao sensor DHT
29 #define DHTTYPE DHT11 // DHT 11
30
31 //LEMBRE-SE: Defina um token de acesso para cada BOARD_ID no código do
   ↵ Gateway
32 #define BOARD_ID 1
33 //#define BOARD_ID 2

```

I.1.2 *boards.h* - Inicialização da placa

```

1
2 #include <Arduino.h>
3 #include <SPI.h>
4 #include <Wire.h>
5 #include "utilities.h"
6 #include <U8g2lib.h>
7 #include <axp20x.h>
8
9 U8G2_SSD1306_128X64_NONAME_F_HW_I2C *u8g2 = nullptr;
10 AXP20X_Class PMU;
11
12 bool initPMU()
13 {
14     if (PMU.begin(Wire, AXP192_SLAVE_ADDRESS) == AXP_FAIL) {
15         Serial.println("Erro AXP192");
16         return false;
17     }
18
19     PMU.setPowerOutPut(AXP192_DCDC1, AXP202_OFF);
20     PMU.setPowerOutPut(AXP192_DCDC2, AXP202_OFF);
21     PMU.setPowerOutPut(AXP192_LDO2, AXP202_OFF);
22     PMU.setPowerOutPut(AXP192_LDO3, AXP202_OFF);
23     PMU.setPowerOutPut(AXP192_EXTEN, AXP202_OFF);
24
25     //Fornecimento de tensão para os módulos
26     PMU.setLDO2Voltage(3300); //LoRa VDD
27     PMU.setLDO3Voltage(3300); //GPS VDD
28     PMU.setDCDC1Voltage(3300); //Pino VCC 3.3V próximo aos pinos 21 e
       ↵ 22 é controlado por DCDC1

```

```

29
30 PMU.setPowerOutPut(AXP192_DCDC1, AXP202_ON);
31 PMU.setPowerOutPut(AXP192_LDO2, AXP202_ON);
32 PMU.setPowerOutPut(AXP192_LDO3, AXP202_ON);
33
34
35 pinMode(PMU_IRQ, INPUT_PULLUP);
36 attachInterrupt(PMU_IRQ, [] {
37     // pmu_irq = true;
38 }, FALLING);
39
40 PMU.adc1Enable(AXP202_VBUS_VOL_ADC1 |
41                 AXP202_VBUS_CUR_ADC1 |
42                 AXP202_BATT_CUR_ADC1 |
43                 AXP202_BATT_VOL_ADC1,
44                 AXP202_ON);
45
46 PMU.enableIRQ(AXP202_VBUS_REMOVED_IRQ |
47                 AXP202_VBUS_CONNECT_IRQ |
48                 AXP202_BATT_REMOVED_IRQ |
49                 AXP202_BATT_CONNECT_IRQ,
50                 AXP202_ON);
51 PMU.clearIRQ();
52
53 return true;
54 }
55
56 void disablePeripherals()
57 {
58     PMU.setPowerOutPut(AXP192_DCDC1, AXP202_OFF);
59     PMU.setPowerOutPut(AXP192_LDO2, AXP202_OFF);
60     PMU.setPowerOutPut(AXP192_LDO3, AXP202_OFF);
61 }
62
63
64 void initBoard()
65 {
66     Serial.begin(115200);
67     Serial.println("initBoard");
68     SPI.begin(RADIO_SCLK_PIN, RADIO_MISO_PIN, RADIO_MOSI_PIN);
69     Wire.begin(I2C_SDA, I2C_SCL);
70     Wire1.begin(I2C_SDA2, I2C_SCL2);

```

```

71
72 Serial1.begin(GPS_BAUD_RATE, SERIAL_8N1, GPS_RX_PIN, GPS_TX_PIN);
73
74 initPMU();
75
76 Wire1.beginTransmission(0x3C);
77 if (Wire1.endTransmission() == 0) {
78   Serial.println("Started OLED");
79   u8g2 = new U8G2_SSD1306_128X64_NONAME_F_HW_I2C(U8G2_R0,
80             → U8X8_PIN_NONE, I2C_SCL2, I2C_SDA2);
81   u8g2->begin();
82   u8g2->clearBuffer();
83   u8g2->setFlipMode(0);
84   u8g2->setFontMode(1);
85   u8g2->setDrawColor(1);
86   u8g2->setFontDirection(0);
87   u8g2->firstPage();
88   do {
89     u8g2->setFont(u8g2_font_inb19_mr);
90     u8g2->drawStr(0, 30, "LilyGo");
91     u8g2->drawHLine(2, 35, 47);
92     u8g2->drawHLine(3, 36, 47);
93     u8g2->drawVLine(45, 32, 12);
94     u8g2->drawVLine(46, 33, 12);
95     u8g2->setFont(u8g2_font_inb19_mf);
96     u8g2->drawStr(58, 60, "LoRa");
97   } while ( u8g2->nextPage() );
98   u8g2->sendBuffer();
99   u8g2->setFont(u8g2_font_7x13B_tf);
100 }
101 }
```

I.1.3 Algoritmo principal da unidade de sensoriamento

```

1 #include <LoRa.h>
2 #include "boards.h"
3 #include <DHT.h>
4 #include <ArduinoJson.h>
5 #include <TinyGPS++.h>
6 #include <UnixTime.h>
```

```

7
8 UnixTime timestamp(3);
9 uint32_t unix;
10 String temp_time = "";
11
12 DHT dht(DHTPIN, DHTTYPE);
13 TinyGPSPlus gps;
14
15 unsigned long counter = 0;
16 String umidade_str = "";
17 String temp_str = "";
18 double latitude = 0.0, longitude = 0.0;
19
20 boolean debug = false;
21 unsigned long lastSend = 0;
22
23 void displayInfoGPS()
24 {
25     Serial.print(F("Location: "));
26     if (gps.location.isValid()) {
27         Serial.print(gps.location.lat(), 6);
28         Serial.print(F(","));
29         Serial.print(gps.location.lng(), 6);
30     } else {
31         Serial.print(F("INVALID"));
32     }
33
34     Serial.print(F(" Date/Time: "));
35     if (gps.date.isValid()) {
36         Serial.print(gps.date.month());
37         Serial.print(F("/"));
38         Serial.print(gps.date.day());
39         Serial.print(F("/"));
40         Serial.print(gps.date.year());
41         Serial.print(F("Unix string:"));
42         Serial.println(temp_time);
43     } else {
44         Serial.print(F("INVALID"));
45     }
46
47     Serial.print(F(" "));
48     if (gps.time.isValid()) {

```

```

49     if (gps.time.hour() < 10) Serial.print(F("0"));
50     Serial.print(gps.time.hour());
51     Serial.print(F(":"));
52     if (gps.time.minute() < 10) Serial.print(F("0"));
53     Serial.print(gps.time.minute());
54     Serial.print(F(":"));
55     if (gps.time.second() < 10) Serial.print(F("0"));
56     Serial.print(gps.time.second());
57     Serial.print(F("."));
58     if (gps.time.centisecond() < 10) Serial.print(F("0"));
59     Serial.print(gps.time.centisecond());
60 } else {
61     Serial.print(F("INVALID"));
62 }
63
64 Serial.println();
65 }
66
67 String composeJson(){
68     StaticJsonDocument<400> data;
69     String string;
70     //populate JsonFormat
71     data["board_id"] = BOARD_ID;
72     data["packet_id"] = counter++;
73     data["ts"] = temp_time;
74     data["temperature"] = temp_str;
75     data["humidity"] = umidade_str;
76     data["latitude"] = latitude;
77     data["longitude"] = longitude;
78
79     serializeJson(data, string);
80     if (true){
81         Serial.println("INFO: The following string will be send...");
82         Serial.println(string);
83     }
84     return string;
85 }
86
87 void LoRaSend(String packet){
88     LoRa.beginPacket();
89     LoRa.print(packet);
90     LoRa.endPacket();

```

```

91 }
92
93 void setup()
94 {
95     initBoard(); // Um atraso eh necessario apos a inicializacao da
96     // placa
97     delay(1500);
98
99     dht.begin();
100
101    Serial.println("LoRa Sender");
102    LoRa.setPins(RADIO_CS_PIN, RADIO_RST_PIN, RADIO_DIO_PIN);
103    if (!LoRa.begin(LoRa_frequency)) {
104        Serial.println("Starting LoRa failed!");
105        while (1);
106    }
107    LoRa.setSyncWord(0xF3);
108    LoRa.enableCrc();
109 }
110
111 void loop()
112 {
113     if (millis() - lastSend > TIME_INTERVAL) {
114         int h = dht.readHumidity();
115         int t = dht.readTemperature();
116
117         if (isnan(h) || isnan(t)) {
118             Serial.println(F("Failed to read from DHT sensor!"));
119             delay(1000);
120             return;
121         }
122
123         while (Serial1.available() > 0) {
124             if (gps.encode(Serial1.read())){
125                 latitude = gps.location.lat();
126                 longitude = gps.location.lng();
127                 timestamp.setDateTime(gps.date.year(),gps.date.month(),gps.
128                     // date.day(),gps.time.hour(),gps.time.minute(),gps.time.
129                     // second());
130                 unix = timestamp.getUnix();
131                 temp_time = String(unix) + String(gps.time.centisecond());
132                 displayInfoGPS();
133             }
134         }
135     }
136 }

```

```

130         }
131     }
132
133     umidade_str = String(h);
134     temp_str = String(t);
135
136     Serial.print(F("Humidity: "));
137     Serial.print(h);
138     Serial.print(F("% Temperature: "));
139     Serial.print(t);
140     Serial.print(F("C "));
141     Serial.print("Sending packet: ");
142     Serial.println(counter);
143
144     // Envia o pacote somente se todos os dados forem validos
145     if(isnan(h) || isnan(t) || !gps.time.isValid() || !gps.location.
146         → isValid())
147         Serial.println("There is invalid data... Data won't be
148             → transmitted");
149     else
150         LoRaSend(composeJson());
151
152     //Imprimir valores na tela OLED
153     if (u8g2) {
154         char buf[256];
155         u8g2->clearBuffer();
156         /u8g2->drawStr(0, 9, "Transmitting: OK!");
157         snprintf(buf, sizeof(buf), "Temp: %s", temp_str);
158         u8g2->drawStr(0, 9, buf);
159         snprintf(buf, sizeof(buf), "Humidity: %s", umidade_str);
160         u8g2->drawStr(0, 20, buf);
161         snprintf(buf, sizeof(buf), "Lat: %f", latitude);
162         u8g2->drawStr(0, 31, buf);
163         snprintf(buf, sizeof(buf), "Long: %f", longitude);
164         u8g2->sendBuffer();
165     }
166
167     lastSend = millis();
168     delay(1);
169 }
```

I.2 Algoritmo do Gateway

I.2.1 *utilities.h* - Conjunto de parâmetros

```
1 #define LoRa_frequency 915E6
2
3 #define UNUSE_PIN (0)
4
5 #define GPS_RX_PIN 34
6 #define GPS_TX_PIN 12
7 #define BUTTON_PIN 38
8 #define BUTTON_PIN_MASK GPIO_SEL_38
9 #define I2C_SDA 21
10 #define I2C_SCL 22
11 #define PMU_IRQ 35
12 #define RADIO_SCLK_PIN 5
13 #define RADIO_MISO_PIN 19
14 #define RADIO_MOSI_PIN 27
15 #define RADIO_CS_PIN 18
16 #define RADIO_DIO_PIN 26
17 #define RADIO_RST_PIN 23
18 #define RADIO_DIO1_PIN 33
19 #define RADIO_BUSY_PIN 32
20 #define GPS_BAUD_RATE 9600
21
22 //Parametros ThingsBoard
23
24 // WiFi access point
25 #define WIFI_AP_NAME "NOME_DA_REDE_WIFI"
26
27 // WiFi password
28 #define WIFI_PASSWORD "SENHA_REDE_WIFI"
29
30 //Endereco publico do local onde o ThingsBoard esta instalado
31 #define THINGSBOARD_SERVER "thingsboard.hopto.org"
32
33 // Tokens de acesso dos dispositivos
34 //LEMBRE-SE: Caso um novo dispositivo seja adicionado, seu token deve
35 //             → ser inserido abaixo
35 #define TOKEN_1 "fCOviMrHljIHbyHnW7D7"
36 #define TOKEN_2 "Ap6uSgd8EecGG8bGyffFG"
```

I.2.2 boards.h - Inicialização da placa

```
1 #include <Arduino.h>
2 #include <SPI.h>
3 #include <Wire.h>
4 #include "utilities.h"
5 #include <U8g2lib.h>
6 #include <axp20x.h>
7
8 U8G2_SSD1306_128X64_NONAME_F_HW_I2C *u8g2 = nullptr;
9 AXP20X_Class PMU;
10
11 bool initPMU()
12 {
13     if (PMU.begin(Wire, AXP192_SLAVE_ADDRESS) == AXP_FAIL) {
14         Serial.println("Erro AXP192");
15         return false;
16     }
17
18     PMU.setPowerOutPut(AXP192_DCDC1, AXP202_OFF);
19     PMU.setPowerOutPut(AXP192_DCDC2, AXP202_OFF);
20     PMU.setPowerOutPut(AXP192_LDO2, AXP202_OFF);
21     PMU.setPowerOutPut(AXP192_LDO3, AXP202_OFF);
22     PMU.setPowerOutPut(AXP192_EXTEN, AXP202_OFF);
23
24     //Fornecimento de tensao para os modulos
25     PMU.setLDO2Voltage(3300); //LoRa VDD
26     PMU.setDCDC1Voltage(3300); //Pino VCC 3.3V proximo aos pinos 21 e
27         ↪ 22 eh controlado por DCDC1
28
29     PMU.setPowerOutPut(AXP192_DCDC1, AXP202_ON);
30     PMU.setPowerOutPut(AXP192_LDO2, AXP202_ON);
31     PMU.setPowerOutPut(AXP192_LDO3, AXP202_ON);
32
33     pinMode(PMU_IRQ, INPUT_PULLUP);
34     attachInterrupt(PMU_IRQ, [] {
35         // pmu_irq = true;
36     }, FALLING);
37
38     PMU.adc1Enable(AXP202_VBUS_VOL_ADC1 |
39                     AXP202_VBUS_CUR_ADC1 |
```

```

40         AXP202_BATT_CUR_ADC1 |  

41         AXP202_BATT_VOL_ADC1,  

42         AXP202_ON);  

43  

44 PMU.enableIRQ(AXP202_VBUS_REMOVED_IRQ |  

45             AXP202_VBUS_CONNECT_IRQ |  

46             AXP202_BATT_REMOVED_IRQ |  

47             AXP202_BATT_CONNECT_IRQ,  

48             AXP202_ON);  

49 PMU.clearIRQ();  

50  

51     return true;  

52 }  

53  

54 void disablePeripherals()  

55 {  

56     PMU.setPowerOutPut(AXP192_DCDC1, AXP202_OFF);  

57     PMU.setPowerOutPut(AXP192_LDO2, AXP202_OFF);  

58     PMU.setPowerOutPut(AXP192_LDO3, AXP202_OFF);  

59 }  

60  

61  

62 void initBoard()  

63 {  

64     Serial.begin(115200);  

65     Serial.println("initBoard");  

66     SPI.begin(RADIO_SCLK_PIN, RADIO_MISO_PIN, RADIO_MOSI_PIN);  

67     Wire.begin(I2C_SDA, I2C_SCL);  

68  

69     Serial1.begin(GPS_BAUD_RATE, SERIAL_8N1, GPS_RX_PIN, GPS_TX_PIN);  

70  

71     initPMU();  

72  

73     Wire.beginTransmission(0x3C);  

74     if (Wire.endTransmission() == 0) {  

75         Serial.println("Started OLED");  

76         u8g2 = new U8G2_SSD1306_128X64_NONAME_F_HW_I2C(U8G2_R0,  

77             → U8X8_PIN_NONE, I2C_SCL, I2C_SDA);  

78         u8g2->begin();  

79         u8g2->clearBuffer();  

80         u8g2->setFlipMode(0);  

81         u8g2->setFontMode(1);

```

```

81     u8g2->setDrawColor(1);
82     u8g2->setFontDirection(0);
83     u8g2->firstPage();
84     do {
85         u8g2->setFont(u8g2_font_inb19_mr);
86         u8g2->drawStr(0, 30, "LilyGo");
87         u8g2->drawHLine(2, 35, 47);
88         u8g2->drawHLine(3, 36, 47);
89         u8g2->drawVLine(45, 32, 12);
90         u8g2->drawVLine(46, 33, 12);
91         u8g2->setFont(u8g2_font_inb19_mf);
92         u8g2->drawStr(58, 60, "LoRa");
93     } while (u8g2->nextPage());
94     u8g2->sendBuffer();
95     u8g2->setFont(u8g2_font_7x13B_tf);
96 }
97
98 }
```

I.2.3 Algoritmo principal do Gateway

```

1 #include <ThingsBoard.h>
2 #include <LoRa.h>
3 #include "boards.h"
4 #include <ArduinoJson.h>
5 #include <WiFi.h>
6
7 WiFiClient espClient;
8 ThingsBoard tb(espClient);
9
10 int status = WL_IDLE_STATUS;
11
12 bool debug = false;
13
14 String received = "";
15 int rxCheckPercent = -1;
16
17 unsigned long packetId = 0;
18 unsigned short int boardId;
19 String umidade_str = "";
20 String temp_str = "";
```

```

21 double longitude = 0.0;
22 double latitude = 0.0;
23 String timestamp = "";
24
25 #define TICKS_ESPERA_ENVIO_JSON ( TickType_t )10000
26 #define TEMPO_PARA_VERIFICAR_CONEXAO 1000 //ms
27 #define TAMANHO_FILA_POSICOES_JSON 100
28
29 //Fila de dados - RTOS
30 QueueHandle_t xQueue_JSON;
31
32 //Semaforo - RTOS
33 SemaphoreHandle_t xSerial_semaphore;
34
35 //Tarefa que recebe a mensagem via LoRa e encaminha para a fila
36 void taskLoraParser(void *pvParameters){
37     while(true) {
38
39         vTaskDelay(10 / portTICK_PERIOD_MS);
40
41         xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
42         int packetSize = LoRa.parsePacket();
43
44         if(packetSize) {
45             if(debug) {
46                 //print incoming packet size
47                 Serial.print("INFO: Incoming packet size: ");
48                 Serial.println(packetSize);
49             }
50
51             received = "";
52
53             //read the incoming packet
54             for (int i = 0; i < packetSize; i++) {
55                 received = received + (char)LoRa.read();
56             }
57
58             //print the incoming packet with RSSI
59             rxCheckPercent = LoRa.packetRssi();
60
61             //map rssи value to percentage
62             rxCheckPercent = map(rxCheckPercent, -145, -30, 0, 100);

```

```

63
64     xQueueSend(xQueue_JSON, ( void * ) &received,
65                 → TICKS_ESPERA_ENVIO_JSON) ;
66
67     if(debug) {
68         Serial.print("INFO: Received packet '");
69         Serial.print(received);
70         Serial.print("' with RSSI ");
71         Serial.println(rxCheckPercent);
72     }
73 }
74 xSemaphoreGive(xSerial_semaphore);
75 }
76 }
77
78 void InitWiFi()
79 {
80     Serial.println("Connecting to AP ...");
81     // attempt to connect to WiFi network
82
83     WiFi.begin(WIFI_AP_NAME, WIFI_PASSWORD);
84     while (WiFi.status() != WL_CONNECTED) {
85         delay(500);
86         Serial.print(".");
87     }
88     Serial.println("Connected to AP");
89 }
90
91 void InitLoRa(){
92     if(debug)
93         Serial.println("LoRa Receiver");
94     LoRa.setPins(RADIO_CS_PIN, RADIO_RST_PIN, RADIO_DIO_PIN);
95     if (!LoRa.begin(LoRa_frequency)) {
96         if(debug)
97             Serial.println("Starting LoRa failed!");
98         while (1);
99     }
100    LoRa.setSyncWord(0xF3);
101    LoRa.enableCrc();
102 }
103

```

```

104 void reconnect() {
105     // Loop until we're reconnected
106     status = WiFi.status();
107     if (status != WL_CONNECTED) {
108         WiFi.begin(WIFI_AP_NAME, WIFI_PASSWORD);
109         while (WiFi.status() != WL_CONNECTED) {
110             vTaskDelay( 500 / portTICK_PERIOD_MS );
111             Serial.print(".");
112         }
113         Serial.println("Connected to AP");
114     }
115 }
116
117 //Tarefa que processa a mensagem recebida e envia para a ThingsBoard
118 void taskWifiTBSender(void *pvParameter) {
119
120     String JSON_str = "";
121
122     xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
123     InitWiFi(); //Connect to wifi
124     xSemaphoreGive(xSerial_semaphore);
125
126     xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
127     InitLoRa(); //Connect and setup LoRa
128     xSemaphoreGive(xSerial_semaphore);
129
130
131     while(true) {
132
133         vTaskDelay( TEMPO_PARA_VERIFICAR_CONEXAO / portTICK_PERIOD_MS );
134         //Check if Wifi is connected, else tries to reconnect
135         xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
136         if (WiFi.status() != WL_CONNECTED) {
137             reconnect();
138         }
139         xSemaphoreGive(xSerial_semaphore);
140
141
142         if( xQueueReceive( xQueue_JSON, &( JSON_str ),
143                         TICKS_ESPERA_ENVIO_JSON ) ){
144
145             String boardToken = "";

```

```

145 StaticJsonDocument<300> data;
146
147 //convert incoming string in json object using ArduinoJson.h
148 //→ library
149 DeserializationError error = deserializeJson(data, JSON_str);
150 if (error) {
151   if(debug){
152     xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
153     Serial.print(F("ERROR: deserializeJson() failed: "));
154     Serial.println(error.f_str());
155     xSemaphoreGive(xSerial_semaphore);
156   }
157 }
158
159 //Extracao dos valores do JSON
160 packetId = data["packet_id"];
161 boardId = data["board_id"];
162 timestamp = (const char*)data["ts"];
163 temp_str = (const char*)data["temperature"];
164 umidade_str = (const char*)data["humidity"];
165 latitude = data["latitude"];
166 longitude = data["longitude"];
167
168 if(debug){
169   xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
170   Serial.print("INFO: ID mes: ");
171   Serial.println(packetId);
172   Serial.print("INFO: DHT 11 Temp: ");
173   Serial.println(temp_str);
174   Serial.print("INFO: DHT 11 Hum: ");
175   Serial.println(umidade_str);
176   xSemaphoreGive(xSerial_semaphore);
177 }
178
179 //LEMBRE-SE: Caso uma nova placa seja adicionada ao sistema, ela
180 //→ devera ser incluida no switch abaixo
181 switch(boardId) {
182
183   case 1 :
184     boardToken = TOKEN_1;
185     break;

```

```

185     case 2 :
186     boardToken = TOKEN_2;
187     break;
188
189 }
190
191 if (debug) {
192     xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
193     Serial.print("Connecting to: ");
194     Serial.print(THINGSBOARD_SERVER);
195     Serial.print(" with token ");
196     Serial.println(boardToken);
197     xSemaphoreGive(xSerial_semaphore);
198 }
199
200 xSemaphoreTake(xSerial_semaphore, portMAX_DELAY );
201
202 if (!tb.connect(THINGSBOARD_SERVER, boardToken.c_str())) {
203     if(debug)
204         Serial.println("Failed to connect");
205 }
206 else{
207     tb.sendTelemetryInt("temperature", temp_str.toInt());
208     tb.sendTelemetryInt("humidity", umidade_str.toInt());
209     tb.sendTelemetryFloat("latitude", latitude);
210     tb.sendTelemetryFloat("longitude", longitude);
211     tb.sendTelemetryString("ts", timestamp.c_str());
212     tb.sendAttributeInt("board_id", boardId);
213     tb.loop();
214     tb.disconnect();
215     if (debug) {
216         Serial.println("Sending data to ThingsBoard");
217         Serial.print("Board ID: ");
218         Serial.println(boardId);
219         Serial.print("Temp:");
220         Serial.print(temp_str.toInt());
221         Serial.print("\tHum:");
222         Serial.println(umidade_str.toInt());
223         Serial.print("Latitude:");
224         Serial.print(latitude);
225         Serial.print("\tLongitude:");
226         Serial.println(longitude);

```

```

227     if (u8g2) {
228         u8g2->clearBuffer();
229         char buf[256];
230         u8g2->drawStr(0, 9, "Received OK!");
231         //u8g2->drawStr(0, 20, recv.c_str());
232         sprintf(buf, sizeof(buf), "Temp:%s", temp_str);
233         u8g2->drawStr(0, 20, buf);
234         sprintf(buf, sizeof(buf), "Humidity:%s", umidade_str);
235         u8g2->drawStr(0, 31, buf);
236         sprintf(buf, sizeof(buf), "Lat:%f", latitude);
237         u8g2->drawStr(0, 42, buf);
238         sprintf(buf, sizeof(buf), "Long:%f", longitude);
239         u8g2->drawStr(0, 54, buf);
240         sprintf(buf, sizeof(buf), "BoardID:%d", boardId);
241         u8g2->drawStr(0, 66, buf);
242         u8g2->sendBuffer();
243     }
244 }
245 }
246 xSemaphoreGive(xSerial_semaphore);
247 }
248 }
249 }
250
251 void setup()
252 {
253     initBoard();
254     vTaskDelay(1500 / portTICK_PERIOD_MS);
255
256     //Criacao da fila
257     xQueue_JSON = xQueueCreate(TAMANHO_FILA_POSICOES_JSON, 12);
258
259     if (xQueue_JSON == NULL) {
260         Serial.println("Falha ao inicializar filas. O programa nao pode
261             → prosseguir. O ESP32 sera reiniciado...");
```

delay(2000);

ESP.restart();

}

//Criacao dos semafornos

xSerial_semaphore = xSemaphoreCreateMutex();

267

```

268 if (xSerial_semaphore == NULL) {
269     Serial.println("Falha ao inicializar semaforos. O programa nao
270         → pode prosseguir. O ESP32 sera reiniciado..."); 
271     delay(2000);
272     ESP.restart();
273 }
274 //Configuracao das tarefas
275 xTaskCreatePinnedToCore(
276     taskLoraParser //Funcao a qual esta implementado o que a tarefa
277         → deve fazer
278     , "LoRa_read" //Nome (para fins de debug, se necessario)
279     , 4096 //Tamanho da stack (em words) reservada para essa tarefa
280     , NULL //Parametros passados (nesse caso, nao ha)
281     , 6 //Prioridade
282     , NULL //Handle da tarefa, opcional (nesse caso, nao ha)
283     , 0);
284
285 xTaskCreatePinnedToCore(
286     taskWifiTBSender
287     , "wifi_TB"
288     , 4096
289     , NULL
290     , 5
291     , NULL
292     , 1);
293 }
294 void loop()
295 {
296 }
297 }
```