Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação
Bacharelado em Engenharia da Computação

João Paulo Sampaio Góes Costa

Sistema de Telemetria para Veículos de Competição Baja baseado em Tecnologia de Comunicação LoRa

São Carlos
2025

João Paulo Sampaio Góes Costa

Sistema de Telemetria para Veículos de Competição Baja baseado em Tecnologia de Comunicação LoRa

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Engenharia da Computação, do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos , como parte dos requisitos para a obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Marcio Merino Fernandes

São Carlos
2025

Agradecimentos

Agradeço principalmente a meus pais, familiares e amigos pelo apoio, carinho e que sempre me impulsionaram a seguir em frente.

Agradeço a equipe Baja UFSCar pela parceria e, em especial, o membro Gabriel Ripper pela ajuda e companhia durante todo o processo de implementação do projeto.

Agradeço ao Prof. Marcio, meu orientador, pela orientação dedicada ao longo do projeto. Agradeço especialmente pelos ensinamentos sobre a escrita acadêmica, que foram fundamentais para o desenvolvimento deste trabalho.

Agradeço ao Prof. Fredy pelo fornecimento dos equipamentos utilizados ao longo projeto.

Agradeço aos meus colegas que conheci no curso e que me ajudaram, incentivaram e me fizeram companhia desde o começo da graduação.

Resumo

A telemetria permite com que se monitore um sistema em tempo real. No autoesporte, ela pode ser utilizada para monitorar o estado dos múltiplos sistemas que garantem o funcionamento do veículo, como sistemas de freio, suspensão motor e volante, permitindo com que a equipe consiga detectar problemas durante testes, treinos e competições, garantindo a segurança do piloto e do veículo.

Este trabalho tem como objetivo projetar um sistema de telemetria para veículos da modalidade Baja. Para isto, foi utilizada a tecnologia de comunicação LoRa, que promete longo alcance e baixo custo de energia que é o ideal para este tipo de aplicação. A fim de atingir o maior desempenho, foram projetados três sistemas diferentes de comunicação LoRa utilizando o módulo LoRa E220-900T220D, assim como os módulos SX1262 e SX1276 presentes nas placas de desenvolvimento Esp32 Heltec Wifi Lora 32 V1 e V3. Também foi utilizada a placa Esp32 WROOM DevKit V1 para a leitura dos sensores embarcados no veículo.

Palavras-chave: LoRa. Telemetria. Baja. Veículo.

Abstract

Telemetry allows us to monitor a system in real time. On motorsports, it can be used to monitor the condition of the variety of systems that ensure the correct operation of the vehicle, like the breaking, suspension, engine and steering wheel systems. This allows the team to be able to detect problems during tests runs, practices and competitions, ensuring the safety of both the pilot and the car.

The aim of this work is to design a telemetry system for Baja vehicles. To achieve this, it was used the LoRa communication technology, which offers long range and low energy consumption, making it ideal for this type of application. Additionally, three different LoRa communication systems were compared using the LoRa module E220-900T220D, as well as the SX1262 and SX1276 modules present in the Esp32 Heltec Wifi Lora 32 V1 and V3 development boards. Lastly, the Esp32 WROOM DevKit V1 board was used for reading the sensors embedded in the vehicle.

Keywords: LoRa. Telemetry. Baja. Vehicle.

Lista de ilustrações

Lista de tabelas

Lista de siglas

LoRa - Long Range
UFSCar - Universidade Federal de São Carlos
SF - Spread Factor
CR - Code Rate
BPS - Bits por segundo
SAE - Society of Automotive Engineers
SPI -Serial Peripheral Interface
UART - Universal Asynchronous Receiver-Transmitter
I2C - Inter-Integrated Circuit
RSSI - Received Signal Strength Indicator
PWM - Pulse Width Modulation
GPIO - General-Purpose Input/Output
IoT - Internet of Things
LPWAN - Low Power Wide Area Network
CSS - Chirp Spread Spectrum

Sumário

1 Introdução

1.1 Contexto do Trabalho

A telemetria se refere ao controle, medição e transmissão de dados coletados por equipamentos remotos para um ou mais servidores centrais responsáveis pelo seu processamento e monitoramento (GOMES, 2022). Ela pode ser utilizada para difersos fins como na indústria, na aviação, medicina, esportes e veículos.

A telemetria em veículos, mais especificamente, nos veículos utilizados em competições, é comumente utilizada para realizar o monitoramento de posição, níveis de fluído, combustível, velocidade, temperatura, entre outros.

Em veículos de competição, os dados coletados através dos sensores implementados no carro devem ser transmitidos para a equipe fora do veículo por meio de comunicação sem fio de forma eficiente e precisa. Essa comunicação deve suportar a distância entre o veículo e o local onde a equipe está alocada, ter baixo consumo de energia e longo alcance. Isso se deve ao fato de que o carro possui uma fonte de energia limitada pelo espaço e peso, a qual também é consumida para outras funcionalidades, como os motores, sistemas de controle e lanternas. (GISI et al., 2021).

Desta forma, é possível garantir a segurança dos passageiros, assim como o monitorar o funcionamento e desempenho do veículo, e auxiliar na detecção de problemas no carro. Isto é importante não apenas em competições, mas em testes e treinos. Também é possível até mesmo fazer ajustes remotamente a partir dos dados recebidos em tempo real. Neste caso, é necessário que o dispositivo presente no veículo também seja capaz de receber e processar os dados transmitidos, além da função inicial de enviar os dados coletados pelos equipamentos.

Para realizar a medição dos diversos dados que se deseja coletar em um veículo de competição, é necessário um hardware versátil, resistente e com baixo consumo de energia. Para uma equipe universitária como o Baja, que tem orçamento mais baixo em relação a uma equipe de corrida profissional, algumas placas de desenvolvimento que atendem perfeitamente estas necessidades são o Arduino Uno, Esp8266 e Esp32. Todos podem ser facilmente programados utilizando o software Arduino, que conta com uma variedade de bibliotecas oficiais e não oficiais criadas por usuários.

O Arduino Uno (Figura 1) é conhecido por ser um hardware simples e fácil de usar. Ele possui 14 pinos digitais, 6 pinos PWM (Pulse Width Modulation) e 6 entradas analógicas. Seu consumo de energia tende a ser maior do que o da Esp8266, que possui até 17 pinos digitais para sensores, e do que o da Esp32 (Figura 2), que apresenta o menor consumo de energia entre as três e o maior número de pinos para sensores, com até 36 GPIOs

(General-Purpose Input/Output).

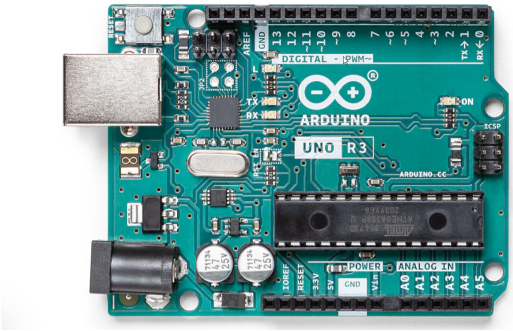

Figura 1 – Placa de desenvolvimento Arduino Uno R3. Fonte - https://store.arduino.cc/

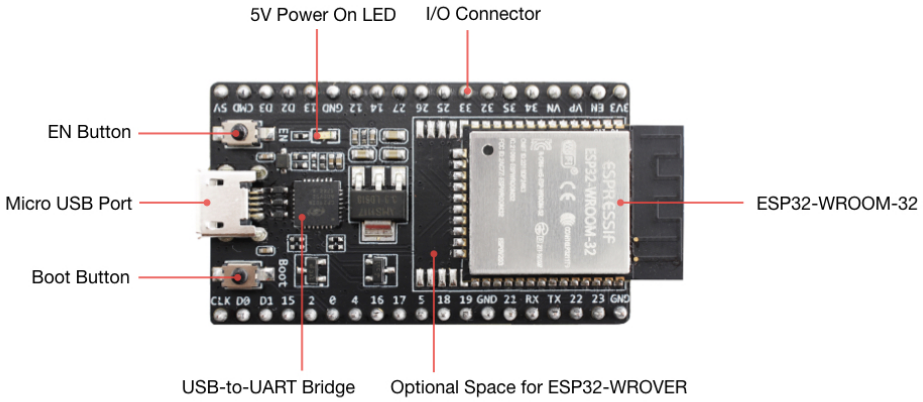

Figura 2 – Placa ESP32 DevKit V1. Fonte - Documentação expressIf https://docs.espressif.com

Para garantir a comunicação entre o veículo e a equipe pelo trajeto completo durante toda a duração da competição, é necessário que o método de comunicação utilizado para a transmissão de telemetria tenha alcance o bastante para cobrir todo o percurso da prova, e, ao mesmo tempo, baixo consumo de energia, visto que a fonte de energia do carro é limitada e, muitas vezes, as competições podem durar diversas horas sem pausas.

Quando se fala em comunicação sem fio de longo alcance e baixo custo, algumas tecnologias aparecem: 3/4G, SigFox, ZigBee e LoRa são as principais.

O LoRa (Long Range) se destaca devido a facilidade de sua implementação e design focado em baixo custo. Diferente do Sigfox, que depende de redes de infraestrutura preexistentes e pode ter cobertura limitada, o LoRa permite que equipes configurem suas próprias redes privadas. Além disso, o LoRa não possui custos de uso de dados, diferentemente das redes 3G/4G, e evita a necessidade de dependência de redes móveis, que podem ser instáveis em áreas isoladas (SUNDARAM; DU; ZHAO, 2020). Um simples

dispositivo pode ser inserido no carro para realizar as medições necessárias por meio de sensores e enviá-las a um outro dispositivo estacionado junto a equipe por meio de comunicação LoRa.

A empresa Heltec criou uma placa de desenvolvimento à partir da Esp32 que conta com um módulo LoRa interno, além de outros como WiFi, Bluetooth e Display. Ele também conta com bibliotecas que facilitam a transmissão e recepção de mensagens LoRa por meio de métodos prontos. A figura 3 contém uma foto desta placa, citando alguns de seus principais recursos.

[!h]

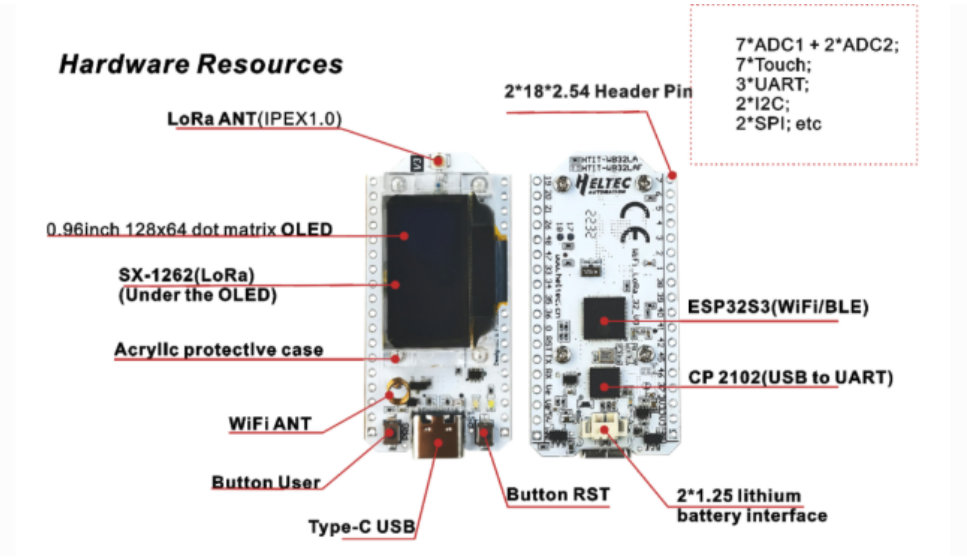

Figura 3 – Placa ESP32-Heltec Wifi LoRa V3. Fonte - Documentação da Heltec https://docs.heltec.org

1.2 Motivação e Justificativa

Como Trabalho de Conclusão de Curso de graduação em Engenharia de Computação, a ideia foi de projetar um sistema real de computação envolvendo hardware e software. Devido a necessidade da equipe Baja UFSCar de implementar um sistema de telemetria para o carro de competição, foi realizada uma parceria com a equipe visando completar os dois objetivos.

O SAE Baja é uma competição que desafia os estudantes de universidades a levarem o conhecimento de engenharia adquirido em sala de aula para a construção e projeção de um veículo off-road. O veículo é um protótipo de estrutura tubular em aço, com quatro ou mais rodas, e motor padrão de 10 HP. A equipe deve desenvolver os sistemas de suspensão, transmissão, freios e chassi. No Brasil a competição nacional recebe o nome de Competição Baja SAE BRASIL e as competições regionais são nomeadas como Etapa Sul, Sudeste e Nordeste. Os vencedores da etapa nacional recebem o direito de competir na prova internacional nos Estados Unidos (SAE-BRASIL,). A figura 4 contém uma

fotografia do veículo da equipe Baja UFSCar durante a competição regional em 2024, na cidade de São Carlos.

[!h]


Figura 4 – Veículo da equipe Baja UFSCAR em competição. Fonte - Instagram @bajaufscar

A modalidade SAE Baja traz alguns desafios para a telemetria. A principal delas é o consumo de energia elétrica, visto que o veículo pode não ter um alternador, sistema que recarrega a bateria do carro, transformando energia mecânica em elétrica. Além disto, é de requisito da equipe que o sistema de telemetria tenha um impacto baixo em relação ao consumo da bateria do carro, uma vez que outros sistemas também utilizam desta mesma fonte de energia elétrica.

Outro desafio, este, em relação a comunicação sem fio da telemetria, é o fato de que o transmissor se move constantemente em relação ao receptor. Isto causa o efeito Doppler, um fenômeno físico ondulatório que ocorre quando existe aproximação ou afastamento relativo entre uma fonte de ondas (transmissor) e um observador (receptor) (HELERBROCK,), o que causa uma alteração na frequência de recepção do sinal seguindo a equação 1:

$$f' = f\left(\frac{v \pm v_o}{v \mp v_s}\right) \tag{1}$$

Onde:

- f' é a frequência percebida pelo observador,
- f é a frequência emitida pela fonte,
- v é a velocidade do som no meio,

- v_o é a velocidade do observador (positiva se se aproximando da fonte),
- v_s é a velocidade da fonte (positiva se se afastando do observador).

Além disto, a movimentação do transmissor pode levar ao surgimento de obstáculos entre o veículo e o receptor.

Por fim, os locais onde são realizadas as competições Baja, podem não contar com uma cobertura completa de redes celular. Isto significa que soluções de comunicação que dependem de tecnologia 3G/4G não são recomendadas.

1.3 Objetivos do Trabalho

1.3.1 Objetivo Geral

O objetivo deste trabalho é projetar e implementar um sistema de telemetria para um veículo automotivo de competição SAE Baja, utilizando a tecnologia de comunicação LoRa, que oferece baixo custo e longo alcance. O trabalho abrange a definição dos equipamentos de hardware, a configuração dos parâmetros da rede de comunicação e a implementação do software necessário.

1.3.2 Objetivos Específicos

A fim de estabelecer um caminho a se seguir para atingir os objetivos gerais descritos anteriormente, foram definidos os seguintes objetivos mais especificos:

a - Estudo teórico sobre a tecnologia LoRa. Primeiramente, foi necessário estudar sobre a tecnologia LoRa, como esta funciona, quais são os equipamentos de Hardware e Software necessários para aplicá-la e como configurar seus parâmetros para a aplicação de telemetria de veículos de competição.

b - Testes iniciais com o equipamento. Foram realizados testes com os diferentes módulos LoRa, afim de verificar principalmente o alcance na comunicação ponto a ponto entre dois módulos, comparando o alcance com a distância real que o sistema deve suportar para as competições da modalidade Baja. A fim de atingir um maior alcance, também foram realizados testes utilizando três dispositivos, no qual um deles atua como intermediário, repassando as mensagens do veículo para o Box.

c - Definição e implementação dos sistemas de comunicação. A partir dos testes iniciais realizados, foram projetados três sistemas de comunicação utilizando a tecnologia de comunicação LoRa. Cada uma utilizando um arranjo diferente dos módulos LoRa disponíveis.

d - Definição de métricas e monitoramento da telemetria. Após os testes iniciais, foram determinados quais os dados a ser transmitidos na telemetria e como monitorá-los. Para

isto, foi escrito um algorítmo em Python para salvar os dados recebidos por LoRa em um arquivo, assim como imprimir um gráfico para facilitar a vizualização.

e - Testes finais com os sensores e veículo. Por fim, foram realizados testes simulando a aplicação real em um veículo, assim como variando os parâmetros da tecnologia LoRa, buscando encontrar qual dos sistemas projetados desempenha melhor em relação a alcance, velocidade de transmissão, consumo de energia e praticidade.

1.4 Organização do Texto

Neste capítulo, foi apresentado os objetivos do trabalho, assim como uma introdução sobre telemetria, tecnologia de comunicação LoRa e algumas opções de dispositivos de hardware que podem ser utilizados para realizar a sua implementação.

O capítulo 2 contém a fundamentação teórica sobre os tópicos de Sistemas Embarcados, Telemetria, Comunicação LoRa e Serial, além de alguns trabalhos de tereceiros que se relacionam com este trabalho, seja pela aplicação de sistemas de telemetria de veículos ou uso da tecnologia de comunicação LoRa.

O capítulo 3 trata de entrar em detalhes sobre a implementação de Hardware e Software realizada no trabalho, discutindo sobre as especificações de cada dispositivo, métodos e parâmetros das bibliotecas necessárias para o seu uso, além de como estes foram aplicados nos sistemas projetados. Já no capítulo 4, são descritos os testes realizados para cada um dos três sistemas de comunicação LoRa projetados, assim como uma análise dos resultados, selecionando, por fim, qual dos sistemas se adequa melhor aos requisitos da aplicação de sistema de telemetria para o veículo de competição Baja.

Por fim, o capítulo 5 apresenta uma breve conclusão apresentando o que foi feito no trabalho, e os códigos utilizados nos testes se encontram no apêndice no fim do documento.

2 Fundamentação Teórica

2.1 Sistemas Embarcados

Sistemas embarcados são dispositivos de computação projetados para realizar tarefas específicas dentro de um sistema maior. Diferentemente de sistemas de propósito geral, como computadores pessoais, os sistemas embarcados integram hardware e software dedicados, otimizados para funções específicas e com recursos limitados, como memória e processamento. Geralmente, esses sistemas operam em tempo real, exigindo alta precisão e confiabilidade, especialmente em aplicações críticas como automação industrial, veículos, e dispositivos médicos. Um sistema embarcado pode incluir microcontroladores ou microprocessadores que executam código otimizado para realizar tarefas de controle, monitoramento, ou comunicação (SOUZA, 2022).

2.2 Telemetria

Telemetria é a tecnologia que permite a coleta e transmissão de dados de sensores a partir de um dispositivo embarcado remoto para uma central de monitoramento. Em sistemas embarcados, a telemetria desempenha um papel fundamental ao possibilitar o monitoramento em tempo real de variáveis críticas, como temperatura, pressão, velocidade, localização ou qualquer outro dado que o sistema necessite (GOMES, 2022) (GENYO, 2024).

Para aplicações automotivas, a telemetria oferece informações essenciais para a manutenção preventiva, otimização de desempenho e resposta a situações de risco. Além disso, a telemetria em sistemas embarcados se beneficia da integração com redes de comunicação sem fio, como LoRa e ZigBee e 3/4G, facilitando a coleta de dados em locais remotos. Esse tipo de monitoramento contínuo e remoto é particularmente importante em ambientes onde os sistemas embarcados estão sujeitos a condições adversas, ajudando a garantir a confiabilidade e a segurança do veículo.

Em veículos de competição, a telemetria pode ser utilizada para monitorar o veículo, medindo o desempenho, garantindo o funcionamento e detectando falhas no veículo e segurança para o(s) pilotos. Por exemplo: Durante uma prova, o um dos sensores de um carro poderia detectar o vazamento do fluído de freio. O sistema de telemetria enviaria esta leitura para a equipe, que por sua vez, poderia avisar o piloto e a organização da prova, evitando um acidente grave.

2.3 Comunicação para sistemas de telemetria

Existem diversas tecnologias diferentes que podem ser utilizadas para realizar a comunicação sem fio para sistema de telemetria. Nesta seção, serão comentadas algumas destas tecnologias, assim como a tecnologia LoRa utilizada para a implementação de telemetria neste trabalho.

2.3.1 SigFox

A rede Sigfox é uma tecnologia de comunicação projetada para aplicações de IoT (Internet das Coisas), sendo amplamente utilizada em telemetria de veículos que demandam baixo consumo de energia e transmissão de pequenos pacotes de dados. A comunicação via Sigfox é baseada em uma rede LPWAN (Low Power Wide Area Network), que permite que dispositivos enviem dados a longas distâncias, com baixa largura de banda. Um lado negativo da tecnologia SigFox é que a empresa SigFox opera e mantém toda a rede de comunicação. Isso inclui as torres de transmissão e os sistemas centrais que processam os dados enviados pelos dispositivos conectados à rede. Desta forma, Para usar a rede Sigfox, é necessário adquirir assinaturas ou planos de serviço diretamente ou por meio de parceiros da empresa (IKPEHAI et al., 2019).

2.3.2 Redes de Comunicação 3G/4G

As redes 3G e 4G são amplamente utilizadas em telemetria de veículos, especialmente em aplicações que exigem transmissão de dados em tempo real e maior largura de banda. Diferente das redes LPWAN (Low Power Wide Area Network), as redes celulares permitem a coleta de um volume maior de informações, como vídeos, dados de sensores complexos e monitoramento em tempo real. Isso é crucial para a segurança e para aplicações mais robustas, como em frotas de veículos de logística e transporte público.

No entanto, essas redes apresentam um consumo de energia mais elevado, o que pode representar um desafio em dispositivos que dependem de baterias de longa duração (DEVALAL; KARTHIKEYAN, 2018).

2.3.3 LoRa

LoRa (Long Range) é uma tecnologia de comunicação que opera em bandas não licensiadas abaixo de 1 GHz. Esta tecnologia teve origem na década de 1940, com uso principalmente militar, e utiliza uma forma de modulação derivada da Chirp Spread Spectrum (CSS), onde os símbolos enviados são chamados de chirp e são caracterizados pela variação constante de frequência ao longo de um intervalo fixo (SINHA; WEI; HWANG, 2017).

Embora a taxa de transmissão da tecnologia LoRa não seja muito alta, variando em torno de algumas dezenas de kilo bits por segundo, ela compensa com robustez, alcance e tolerância a multi-path (DEVALAL; KARTHIKEYAN, 2018). Para a telemetria de veículos, LoRa é utilizada em soluções que necessitam de alcance e flexibilidade, pois, ao contrário das tecnologias SigFox e 3/4G, ela permite a criação de redes privadas, sem necessidade de contratar um provedor de rede externo. Essa característica é vantajosa quando se deseja manter controle total sobre a infraestrutura de comunicação, como em frotas de empresas que operam em áreas específicas.

Com capacidade para transmitir dados a longas distâncias e com baixo consumo de energia, LoRa é ideal para informações básicas de telemetria, como localização e status de funcionamento. Contudo, assim como o Sigfox, possui limitações de banda e é mais adequada para aplicações onde o volume de dados é reduzido e a latência não é um fator crítico. A fácil implementeção, baixo consumo de energia e alto alcance tornam o LoRa a tecnologia ideal para realizar a telemetria de um veículo automóvel de competição.

2.3.3.1 Parâmetros da Modulação LoRa

A tecnologia LoRa utiliza uma modulação baseada em CSS. Esta modulação utiliza um método de espalhamento de frequência, Isto significa que são enviados pulsos que aumentam ou diminuem a frequência constantemente ao longo do tempo como símbolos. A figura 5 e 6 representam dois destes pulsos.

Na figura 5, o gráfico de cima apresenta um símbolo chamado de up-chirp, no qual a frequência aumenta de forma constante ao longo do tempo (eixo horizontal), iniciando da menor frequência da banda, até a maior. O gráfico de baixo representa um down-chirp. No qual a frequência diminui constantemente durante o intervalo de tempo. A figura 6 contém uma representação dos mesmos dois sinais, com o eixo vertical indicando a frequência e o eixo horizontal indicando o tempo.

A modulação LoRa se utiliza de diversos parâmetros que podem impactar na velocidade de transmissão, alcance e robustez do sinal.

A largura de banda determina a frequência mínima e máxima que um chirp pode alcançar longo do tempo. Desta forma, quanto maior a largura de banda, maior a taxa de transmissão da comunicação. Outro parâmetro é a frequência, que indica a frequência central do chirp. Este parâmetro varia entre 430 e 930 MHz.

o Spreading Factor é o parâmetro que determina a taxa de variação da frequência do sinal ao longo do tempo. Isto também implica em quantos bits um símbolo representa e normalmente varia de 7 a 12. Um Spreading Factor de 7 significa que o símbolo representa 7 bits e que é possível formar 2^7 símbolos diferentes.

Isto significa que aumentar o Spreading Factor resulta em um aumento do número de símbolos que podem ser transmitidos, porém também implica em um aumento no tempo

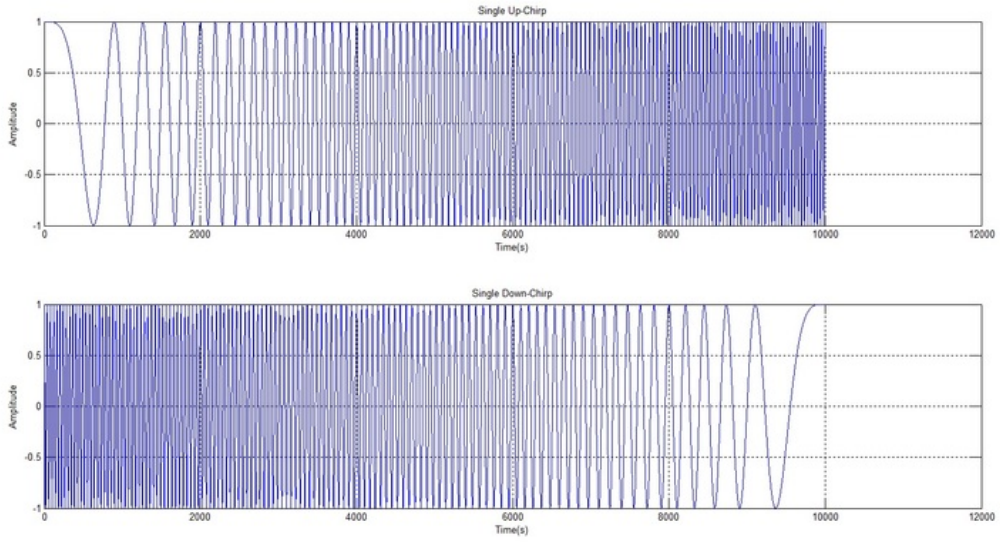

Figura 5 – Exemplo de up-chirp e down-chirp. Símbolos LoRa em que a frequência do sinal varia de modo constante ao longo do tempo.
fonte:<https://www.researchgate.net>

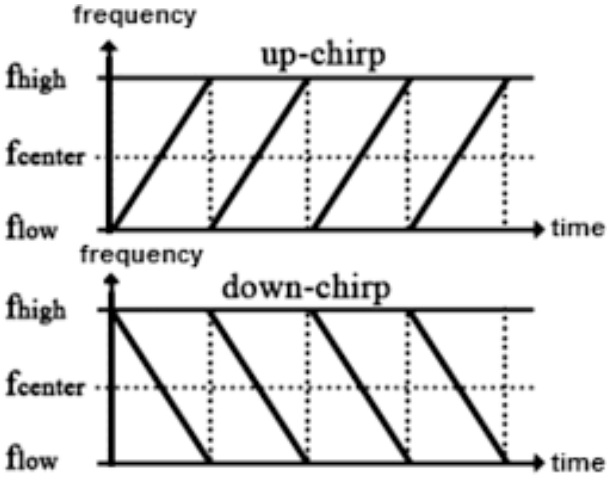

Figura 6 – Exemplo de up-chirp e down-chirp. Símbolos LoRa em que a frequência do sinal varia de modo constante ao longo do tempo.
<https://www.researchgate.net>

de transmissão do símbolo. Desta forma, quanto maior o Spreading Factor, menor a taxa de transmissão de bits, porém maior a robustez do sinal.

Outro parâmetro é o Code Rate, que implica em quantos bits de uma mensagem são "úteis", ou seja, se referem a informação que se deseja transmitir em relação ao total de bits da mensagem. Desta forma, um Code Rate de 4/5 significa que a cada quatro bits úteis, um é de redundância, e é utilizado para correção de erros na transmissão. Quanto menor o Code Rate, maior a robustez e menor a taxa de transmissão (AUGUSTIN et al.,

2016).

A equação 2 demonstra a influência do Spreading Factor (SF), largura de banda (BW) e CodeRate na taxa de transmissão da mensagem (Rb) em bits por segundo.

$$R_b = SF \cdot \left(\frac{BW}{2^{SF}} \right) \cdot CR \tag{2}$$

Além destes parâmetros, também é possível definir o tamanho do preâmbulo da comunicação. Este é composto de símbolos que antecedem os dados da mensagem, e são utilizados para garantir a sincronização entre o transmissor e receptor.

Também é possivel definir a Sync Word, que pode variar de 0 a 0xFF (255 em decimal), que é um parâmetro capaz de isolar a comunicação. Dispositivos LoRa só são capazes de receber mensagens transmitidas com o mesmo sync word. Desta forma, em um ambiente em que sinais LoRa de múltiplas redes se sobrepõem, como em uma competição com vários carros, é possível escolher um valor de sync word para que a sua rede privada receba apenas as mensagens transmitidas por outros dispositivos da mesma rede.

A figura 7 representa a transmissão de uma mensagem LoRa à partir da frequência ao longo do tempo. Cada linha azul da figura é um chirp ou símbolo. Pode-se perceber que neste caso, o Preâmbulo da mensagem é caracterizado por uma sequência de up-chirps seguidos de dois down-chirps , que representam a sync word. Logo após o preâmbulo e sync word, são enviados os dados através de símbolos variados.

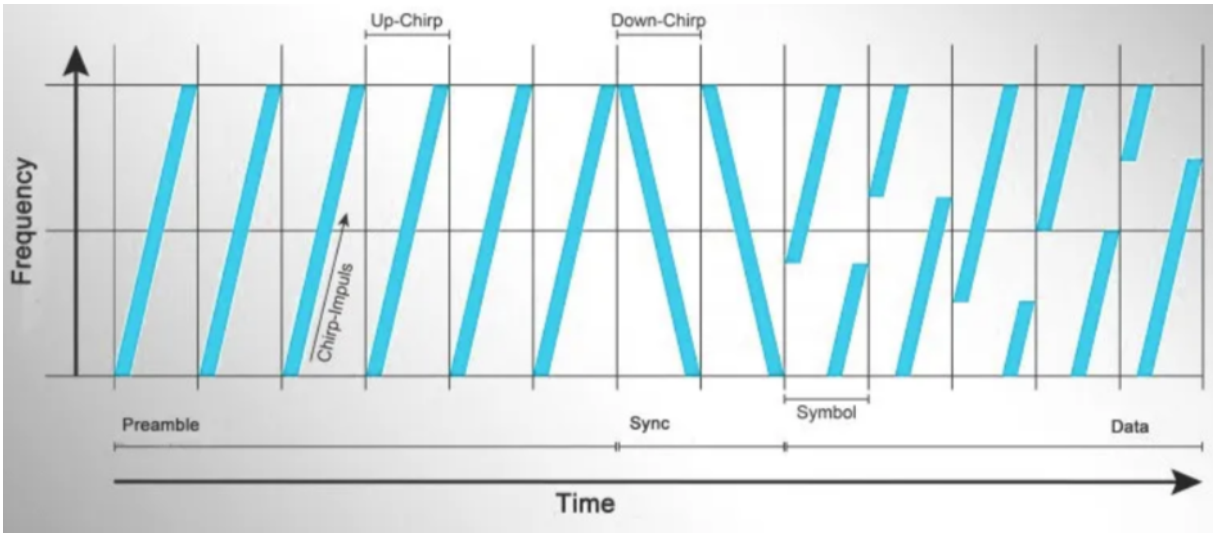

Figura 7 – Variação de frequência de um sinal LoRa ao longo do tempo. Fonte - https://medium.com/@prajzler/

2.3.3.2 Placa de dessenvolvimento Heltec Lora WIFI 32

Existem diversos equipamentos que implementam a modulação LoRa. Um deles é a placa Heltec Lora WIFI 32 produzido pela empresa Heltec Automations. Esta conta

com módulos de WIFI, Bluetooth e LoRa. Também conta com um display OLED de 0,94”128×64. Com o Auxílio de bibliotecas arduino e heltec, é possível projetar códigos que realizem a comunicação Wireless via LoRa com facilidade. Também é possível escolher alguns dos parâmetros da modulação LoRa discutidos anteriormente, como Spreading Factor, que varia entre 6 e 12, e Code Rate que pode variar de 4/5 a 4/8. A frequência é regulada de acordo com a região do dispositivo. Para o Brasil, esta frequência é de 915 MHz. A largura de banda pode ter um valor fixo entre 7,8 e 250 KHz.

2.3.4 RSSI

RSSI - Received Signal Strength Indicator é a medida que quantifica a intensidade com qual um sinal de comunicação é recebido por um equipamento. O RSSI é similar a medida de potência de sinal em decibeis, que quantifica a potência com qual um sinal é transmitido. Para a medição de RSSI, quanto mais próximo de 0, maior a intensidade do sinal recebido.

Na comunicação LoRa, esta é uma medida que pode ser utilizada para classificar a qualidade da comunicação, além de outras medidas como porcentagem de pacotes perdidos e alcance. Entretanto, é importante ressaltar que a escala de RSSI é determinado pelo seu fabricante, desta forma, não é uma maneira eficiente de se analisar o desempenho de comunicação entre equipamentos de diferentes modelos.

2.3.5 Comunicação Serial

A comunicação serial é um método de transmissão de dados em que os bits são enviados sequencialmente, um de cada vez, por um meio, como um fio. Ela pode ser feita por um canal só, onde todos os dispositivos se comunicam por um fio só, ou por mais canais.

A comunicação serial tem dois tipos: Na comunicação serial tipo síncrona, implementada na comunicação SPI e I2C, um canal de clock é compartilhado entre os dispositivos. Neste caso, a cada subida do sinal de clock, o receptor lê o bit do canal de transmissão. Uma representação deste processo pode ser observada na figura 8.

Já na comunicação serial do tipo assíncrona, mais especificamente do tipo UART, que é comumente empregado nos microcontroladores, não há canal de clock, porém é necessário estabelecer uma Baud Rate: uma taxa em que os dados são transmitidos e lidos. Também é necessário acrescentar um bit de início, chamado de start bit, e um bit de fim, chamado de stop bit, ao fim da mensagem (TIER, 2019). A figura 9 demonstra uma representação deste processo ao transmitir o símbolo ’S’ da tabela ASCII.

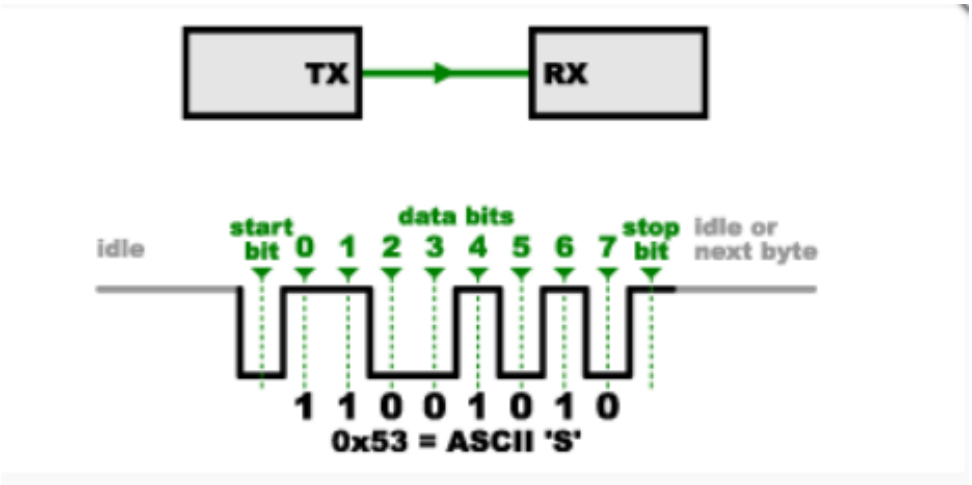

Figura 8 – Comunicação serial do tipo síncrona. Fonte - https://www.robocore.net/

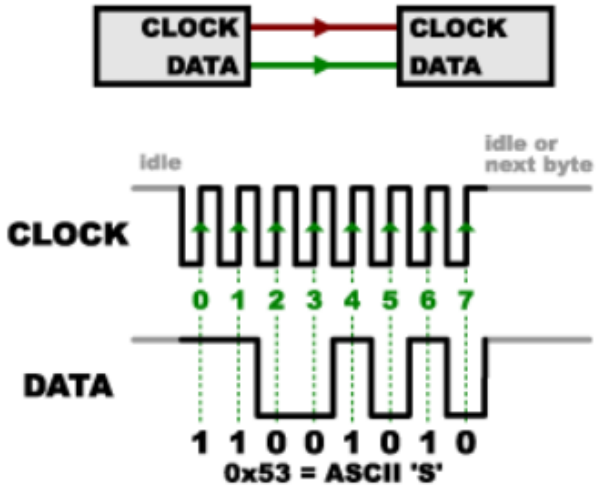

Figura 9 – Comunicação serial do tipo assíncrona. Fonte - ROBOCORE

2.4 Trabalhos Relacionados

Nesta seção, foram trazidos alguns trabalhos que se aproximam com este trabalho, discutindo sobre tecnologias utilizadas em telemetria de veículos e também a tecnologia LoRa.

O trabalho de Gisi et al. 2021 apresenta o sistema de telemetria utilizado no Astra, o primeiro carro movido a energia solar desenvolvido pela Universidade de Kansas, e que compete no Formula Sun Grand Prix, em que o veículo deve completar o maior número de voltas durante três dias. O projeto utiliza a metodologia de rede CAN - Controller Area Network, onde cada subsistema dentro do carro é um nó capaz de se comunicar independentemente sem a necessidade de um controlador central. Para a comunicação do veículo com o externo, foi utilizada uma placa Arduino Due com módulo XBee3 Cellular LTE-M e a tecnologia 4G, de modo que os dados são enviados para um servidor que é acessado para se realizar o monitoramento.

Almeida et al 2023, faz um estudo comparativo entre as tecnologias de comunicação por radiofrequência LoRa e ZigBee, com o objetivo de verificar qual tecnologia é a melhor escolha para um projeto de veículo de competição Fórmula-SAE.

Para isto, foram inseridos um módulo LoRa com a placa ESP32 Heltec Wifi LoRa 32 V2, um módulo ZigBee-PRO e um módulo GPS em um carro-teste que percorreu um percurso em uma pista em linha reta. Uma base imóvel com módulos ZigBee e LoRa recebia as mensagens transmitidas pelo carro-teste e utilizava a localização GPS para determinar a distância entre o carro e a base quando a mensagem foi enviada. Também era determinado o RSSI do sinal. Utilizando a distância e RSSI de cada mensagem, foi possível concluir que, para este tipo de aplicação, a tecnologia LoRa é a mais recomendada.

Já em Dias 2023, são utilizadas duas placas Heltec Wifi LoRa 32 - as mesmas utilizadas neste trabalho - para realizar a transmissão de mensagens de dentro de um veículo automóvel para um local externo, verificando o alcance e a intensidade do sinal deste modelo de placa em um ambiente urbano.

No trabalho de Neto 2021, o autor implementa um sistema de telemetria para a equipe Cerrado Baja SAE utilizando a tecnologia LoRa com o módulo RA-02 e a placa Arduino Uno. O autor também fala de fundamentos da comunicação entre dispositivos, como redundância ciclica, bit de paridade e checksum. Além da transmissão dos dados, também são discutidos os sensores implantados no carro para detecção de nível de combustível, inclinação do veículo, temperatura do motor, entre outros.

Tier 2019, também utiliza a placa Arduino Uno, porém com um módulo LoRa SX1276, o mesmo empregado na placa Heltec Wifi LoRa 32 V3, para realizar a telemetria do carro da equipe Fórmula UFSM da Universidade Federal de Santa Maria. Para a comunicação interna entre os nós do veículo, também é utilizada a metodologia CAN com comunicação serial.

O autor entra em detalhes sobre a implementação de todos os sensores e escolha de parâmetros do módulo LoRa, levando em consideração o efeito dos parâmetros de Spreading Factor, largura de banda e comprimento do preâmbulo na taxa de transmissão de bit e alcance. Escolhendo por um Spreading Factor de 7 e largura de banda de 125 KHz, o autor chega a uma distância máxima de transmissão de aproximadamente 745m.

O autor também utiliza-se da tecnologia Bluetooth para disponibilizar os dados em um display para o piloto do veículo.

Renzone et al. 2024 trata de validar a robustez da tecnologia LoRa contra o efeito Doppler, que ocorre quando um transmissor ou receptor de radiofreqûencia se move em relação ao seu par. Para isto, foram feitos dois cenários de testes: Um em que um carro equipado com um equipamento LoRa percorria um trajeto em linha reta transmitindo mensagens enquanto um receptor foi colocado no meio da pista, e outro com uma motocicleta em uma pista oval com um receptor LoRa no meio. Em ambos os cenários, foram realizados diversos testes variando a velocidade do veículo e o parâmetro Spreading Factor

do equipamento LoRa.

Os resultados dos testes mostraram que, embora a velocidade do veículo não interfiria com o RSSI do sinal, maiores velocidades tiveram uma perda de pacote maior. No teste em que se passava de 90 km/h, a porcentagem de pacotes recebidos foi de 77% com a pista oval. No cenário com a trajeto em linha reta, foi verificado que o efeito Dopler não tem grande efeito no desempenho da comunicação LoRa, uma vez que nestes testes, embora o RSSI diminuia com o aumento de velocidade, a porcentagem de pacotes recebidos se manteu de 90 a 100% na maior parte dos testes.

3 Metodologia e Desenvolvimento do Projeto

3.1 Visão Geral

Existem diferentes placas e módulos que implementam a tecnologia de comunicação LoRa, assim como diversas bibliotecas criadas para facilitar o uso destes equipamentos.

Neste trabalho, foram utilizadas três placas Esp32 diferentes, a Esp32 WROOM DevKit V1 em conjunto com o módulo LoRa E220-900T22D, e as placas Esp32 Heltec Wifi Lora 32 V1 e V3, que contém os módulos LoRa SX1276 e SX1262 respectivamente.

De modo geral, o sistema de telemetria envolve uma placa de desenvolvimento que é embarcada no veículo ligada aos sensores e dispositivo LoRa para comunicação. Esta placa envia mensagens a uma base que contém um dispositivo LoRa e um programa na linguagem Python para armazenar e disponibilizar os dados do veículo. Um diagrama representando este sistema pode ser observado na figura 10. Neste capítulo, será discutido sobre como foi realizada a implementação de hardware no sistema, assim como o software e bibliotecas utilizadas.

Figura 10 – Representação do Sistema de telemetria - fonte: O Autor

3.2 Implementação de Hardware

Para realizar a telemetria, tanto na parte de ler os dados dos sensores, quanto para envia-los via LoRa, foram utilizadas as placas de desenvolvimento Esp32 WROOM DevKit V1, Esp32 Heltec Wifi Lora 32 V1 e V3 e, como alternativa uma alternativa para o módulo de comunicação LoRa, o E220-900T22D. As subsessões a seguir apresentam os detalhes sobre a implementação de cada um dos dispositivos citados.

3.2.1 Leitura dos sensores

A placa escolhida para realizar as medições dos sensores do carro foi a Esp32 DevKit V1 da EspressIf, que possui módulos BlueTooth, WiFi e 20 GPIOs disponíveis para periféricos e opera, em média, a 80 mA de acordo com seu datasheet disponível em //www.espressif.com. A figura 11 contém informações mais detalhadas dos pinos disponíveis desta placa.

As placas Esp32 Heltec Wifi Lora 32 V1 e V3 também tem pinos de entrada que poderiam ser utilizados para a leitura dos sensores, porém, em testes realizados com estas placas e dois sensores de temperatura diferentes, o max6675 e o LM35, a placa não foi capaz de realizar a leitura corretamente. De modo a não atrasar o cronograma estipulado para o trabalho, para "contornar"este problema, a placa Esp32 Heltec Wifi Lora 32 V1 foi utilizada apenas para o envio dos dados dos sensores pela comunicação LoRa, enquanto a placa Esp32 DevKit V1 ficou responsável pela medição dos sensores e envio dos valores por comunicação serial UART para a primeira placa.

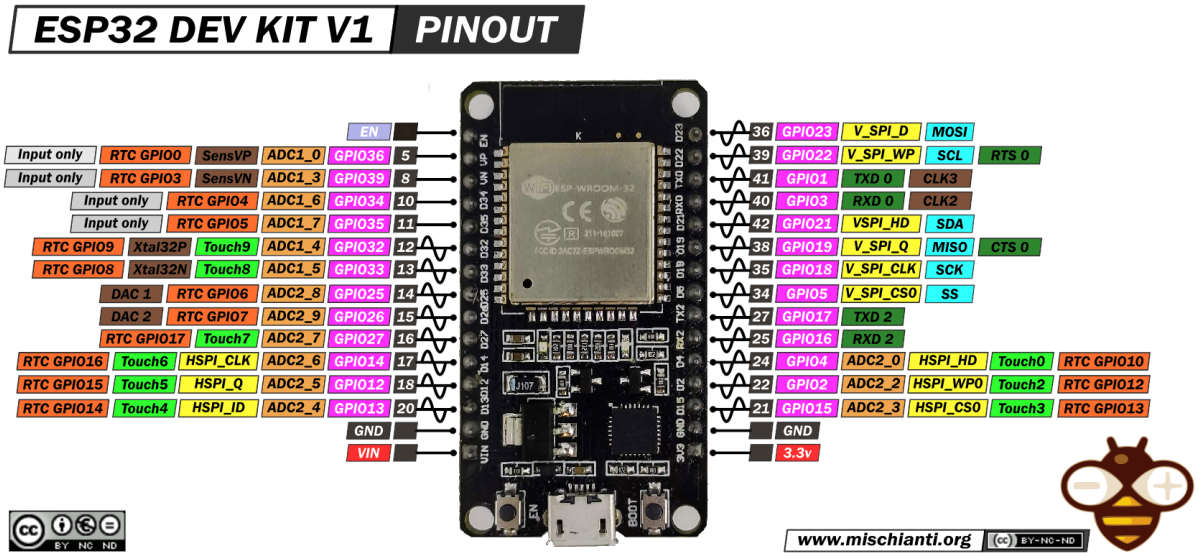

Figura 11 – Diagrama de pinos da Esp32 DevKit V1 - fonte: https://mischianti.org

3.2.2 Comunicação LoRa

A Esp32 Heltec Wifi Lora 32 V1 foi utilizada para receber os dados lidos pela Esp32 DeviKit V1 por comunicação serial e transmitir os dados para outra placa do mesmo modelo via LoRa. Ela é uma placa feita pela empresa Heltec utilizando uma Esp32 e módulo LoRa SX1276. Também conta com um display OLED de 0,94”128×64, módulos Bluetooth, Wifi, 22 pinos GPIO e 6 pinos GPI disponíveis. Um diagrama dos pinos desta placa pode ser observado na figura 12.

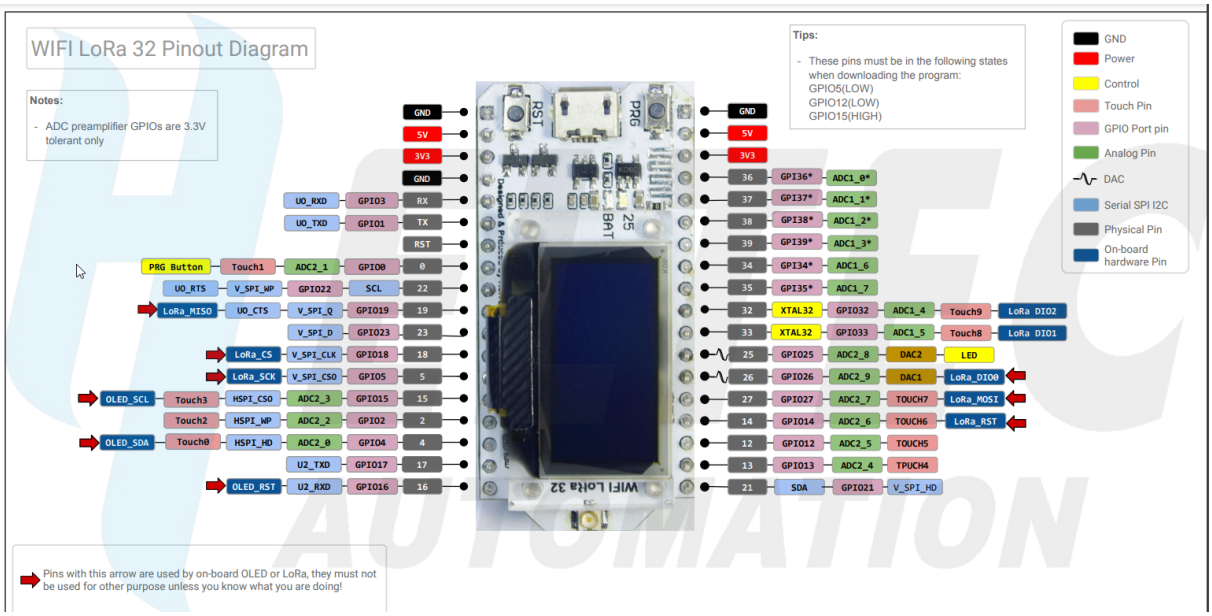

Figura 12 – Diagrama de pinos da Esp32 Heltec LoRa WiFi 32 - fonte: <https://resource.heltec.cn>

Como informado anteriormente, o módulo LoRa incluso nesta placa aceita um Spreading Factor de 7 a 12, frequência de 915 a 928 MHz, largura de banda de 7,8 KHz a 250 KHz, Code Rate de 4/5 a 4/8 e tamanho do preâmbulo de 6 a 65535 símbolos. A definição destes parâmetros pode ser encontrada na API da biblioteca heltec.h disponível para esta placa no link github.com/HelTecAutomation. Na figura 13, é possível observar o consumo desta placa baseado no parâmetro TX Power, que pode ser definido por meio da biblioteca "heltec.h".

A Esp32 Heltec Wifi Lora 32 V1 também foi utilizada para se comunicar com uma placa do V3 do mesmo fabricante. Para isto, como o modelo V1 utiliza o módulo Lora SX1276, e o V3 utiliza o módulo SX1262, que tem parâmetros de largura do preâmbulo e sync word padrão diferentes, foi necessário utilizar a biblioteca "LoRa.h"disponível em <https://github.com/sandeepmistry/arduino-LoRa>.

Para a placa Esp32 Heltec Wifi Lora 32 V3, foi utilizada a biblioteca "heltec_unofficial.h".

Power Consumption(mA)	LoRa 10dB output		50		mA
	LoRa 12dB output		60		mA
	LoRa 15dB output		110		mA
	LoRa 20dB output		130		mA

Figura 13 – Tabela de consumo de energia em miliamperes da placa Esp32 Heltec LoRa WiFi 32 - fonte: Manual de Usuário da placa

3.2.3 Alternativa para a comunicação LoRa

Como alternativa para a Esp Heltec, foi utilizado o módulo LoRa E220-900T22D da empresa Ebyte juntamente com a Esp32 DevKit V1 para realizar a transmissão dos dados coletados pelos sensores. Ele opera com frequência entre 850,125 e 930,125 MHz e consome em média 110 mA para transmissão a 22dBm. Uma representação deste módulo pode ser observado na figura 14.

Figura 14 – Ilustração do módulo LoRa E220-900T22D - fonte: Manual de Usuário do módulo dispnível pela Ebyte

Este módulo define os parâmetros da comunicação LoRa de uma forma diferente: o usuário pode apenas escolher uma taxa de transmissão, e o equipamento define a combinação de Spreading Factor, largura de banda e Code Rate para atender este requisito. Os valores de taxa de transmissão, chamado de air data rate, variam de 2.4 a 62.5 kBits/s. O DataSheet não informa quais são os valores de Spreading Factor, largura de banda e Code Rate para cada valor de air data rate.

O módulo possui 7 pinos, dos quais dois devem ser conectados no Ground e 3.3V da Esp32, outros dois são o RX e TX que são responsáveis pela comunicação serial entre o módulo e a placa, e são conectados ao TX2 e RX2 do Esp32 respectivamente. O pino AUX pode ser utiliado para indicar o status do módulo. Por fim, os pinos M0 e M1 são utilizados para definir o módo de operação. A figura 15 e 16 mostram como foram conectados os pinos da Esp32 WROOM DevKit V1 ao módulo.

Os pinos M0 e M1 podem ser ligados ao ground ou ao 3.3V da placa para se escolher o modo de operação. Para esta aplicação, foi utilizada apenas os modos 0, com ambos ligados ao ground para envio e recepção de mensagens, e o modo 3 para a configuração do módulo. A definição de cada modo pode ser observada na figura 17.

Para facilitar o uso do módulo, foi utilizada a biblioteca LoRa_E220.h disponível em <https://github.com/xreef/EByte_LoRa_E220_Series_Library>.

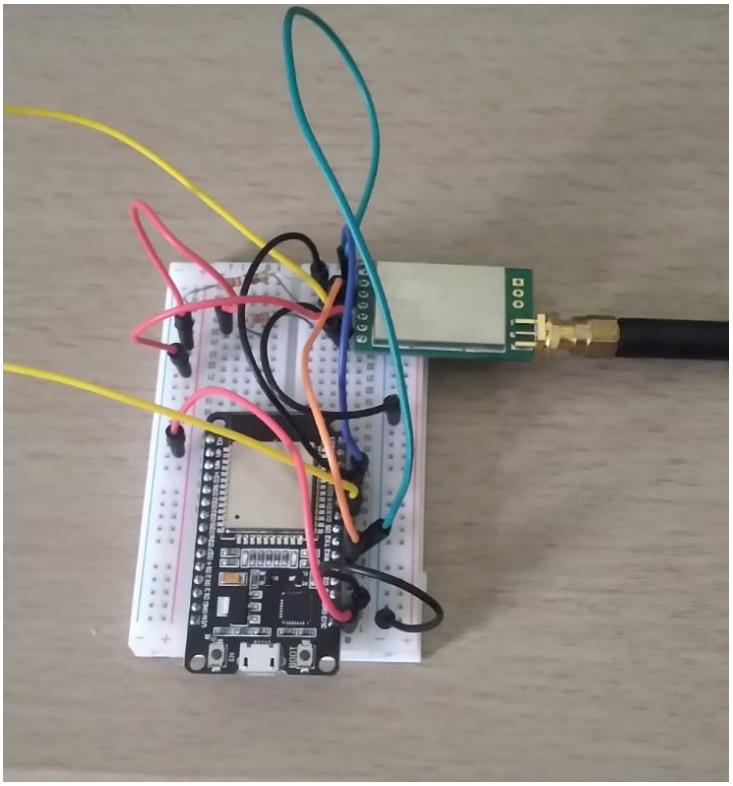

Figura 15 – Foto da configuração dos pinos da Esp32 com o Módulo LoRa E220-900T22D - fonte: O Autor

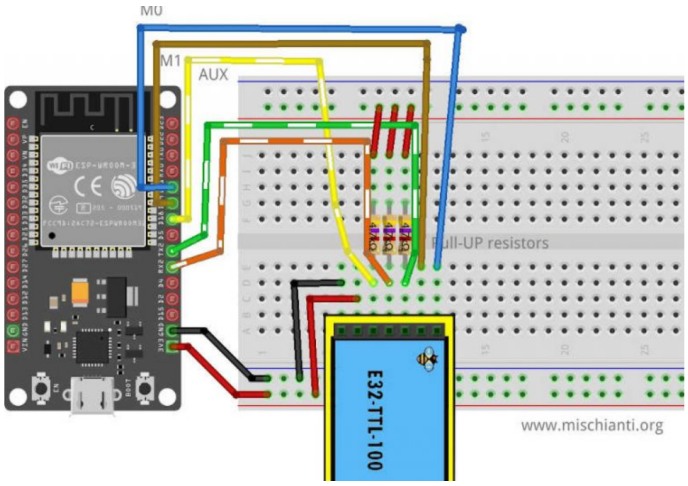

Figura 16 – Configuração dos pinos do Módulo LoRa E220-900T22D e Esp32 WROOM DevKit V1 - fonte: <https://github.com/xreef/>

3.2.4 Sensores

Foram conectados sensores de temperatura, rpm, velocidade e pressão de flúido de freio providenciados pela equipe Baja UFSCar na placa Esp32 WROOM DevKit v1 para a coleta de dados do carro e transmissão por LoRa.

Mode（0-3）	M1	M0	Description	Remark
0 Transmission Mode	0	0	UART and wireless channel are open, transparent transmission is on	
1 WOR Transmitting Mode	0	1	WOR Transmitter	
2 WOR Receiving Mode	1	0	WOR Receiver	Supports wake up over air
3 Deep Sleep Mode	1	1	Module goes to sleep (automatically wake up when configuring parameters)	Supports configuration

Figura 17 – Tabela de modos do módulo LoRa E220-900T22D - fonte: Datasheet do módulo disponível em <https://www.cdebyte.com>

3.3 Implementação de Software

Para cada dispositivo de hardware citado na sessão anterior, foi necessária a instalação e uso de diferentes bilbiotecas. As subseções abaixo entram em detalhes sobre as bibliotecas e métodos essenciais para o funcionamento do sistema de telemetria.

Também foi escrito um programa em Python para armazenar e imprimir os dados dos sensores do veículo em um computador conectado ao dispositivo de receptor da rede de comunicação.

3.3.1 Comunicação LoRa com a placa Esp32 Heltec Wifi Lora 32 V1

Para utilizar a placa Esp32 Heltec Wifi Lora 32 V1 para comunicação LoRa, foram escritos algoritmos utilizando a IDE Arduino disponível em <https://www.arduino.cc/en/software>.

Para programar a placa com o código escrito na IDE, foi necessário inserir a url em github.com na aba de "Preferências"da IDE, como especificado no passo a passo da figura 18, e instalar a placa no "Gerenciador de placas"como na figura 19.

Também foi necessário instalar o driver USB da Silicon Labs disponível no link: www.silabs.com, assim como a biblioteca "heltec.h"disponível no gerenciador de bibliotecas da IDE. A figura 20 mostra onde encontrar o driver e, na figura 21, a biblioteca. Também é necessário selecionar a placa na IDE, como mostra a figura 22, e conectar o dispositivo ao computador com um cabo USB para micro USB apropriado.

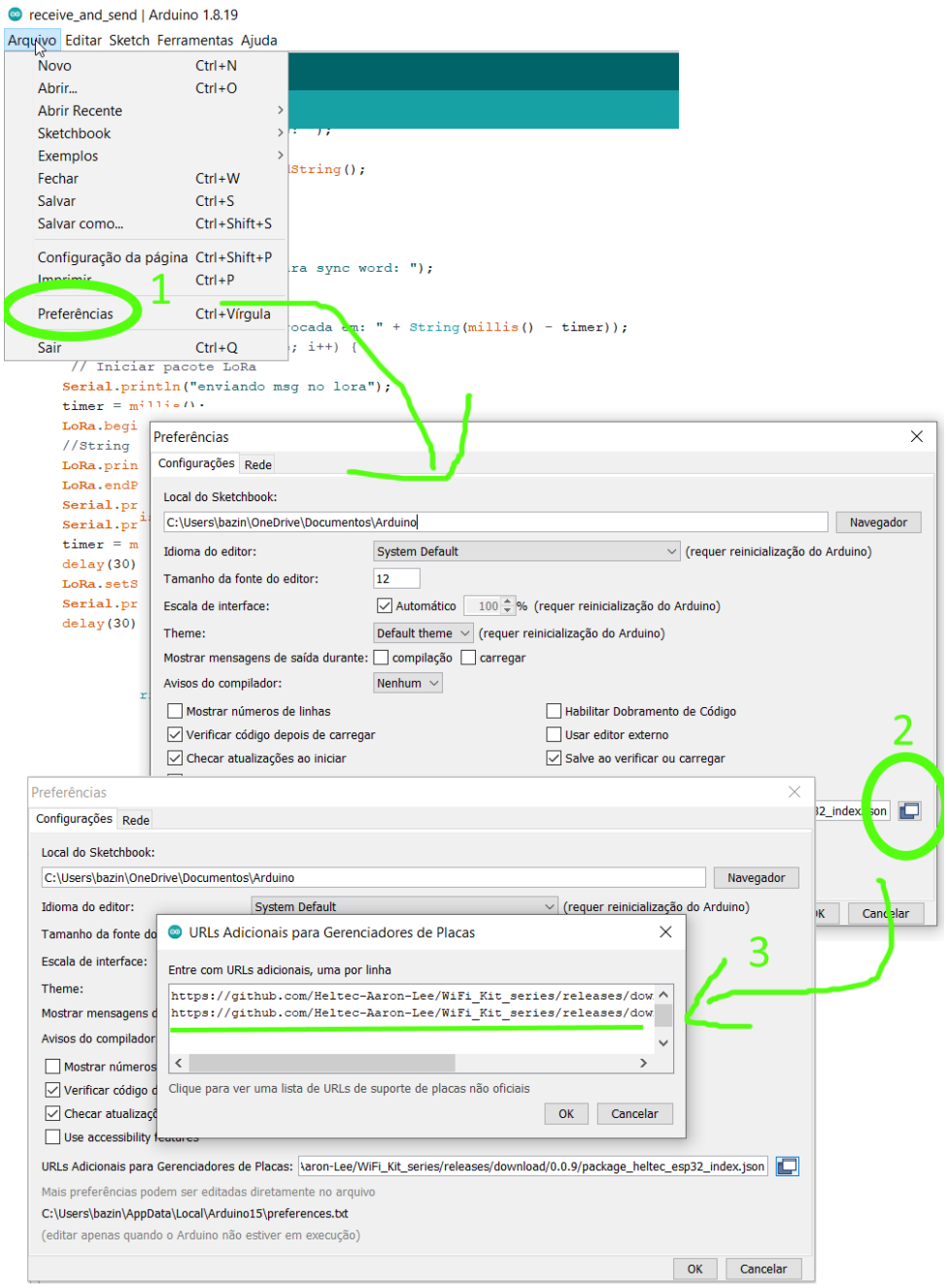

Figura 18 – Passo a passo para instalar a placa heltec - Fonte: O Autor

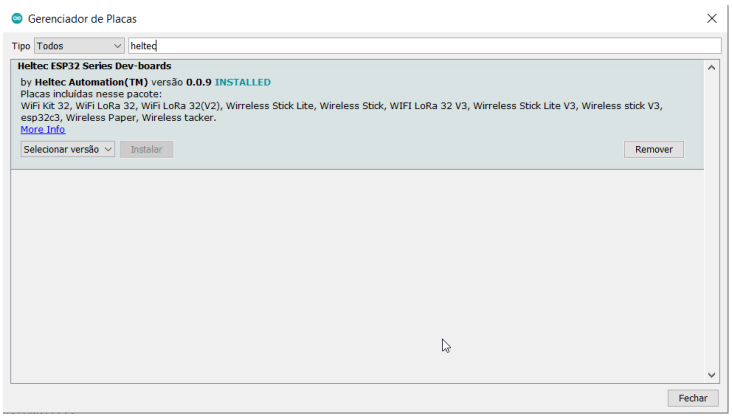

Figura 19 – Instalar a placa heltec no Gerenciados de placas da IDE Arduino - Fonte: O Autor

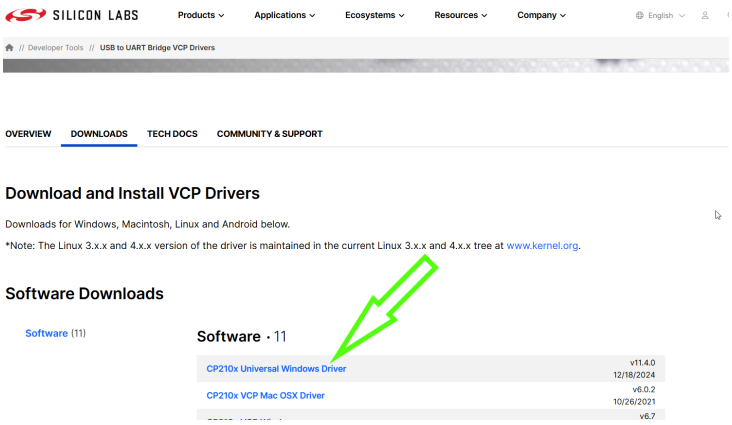

Figura 20 – Driver USB necessário para programar as placas com a IDE Arduino - Fonte: O Autor

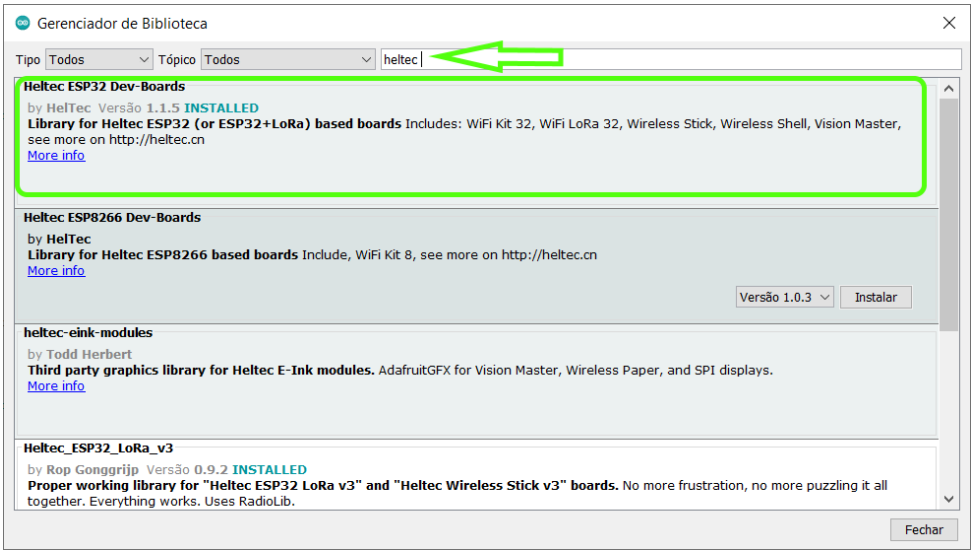

Figura 21 – Biblioteca heltec.h no Gerenciador de bibliotecas na IDE Arduino - fonte: O Autor

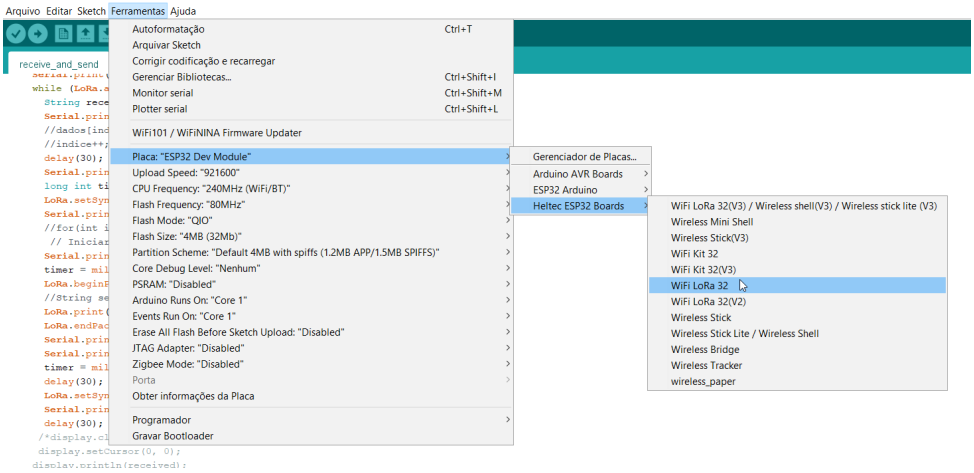

Figura 22 – seleção da placa Esp32 Heltec Wifi Lora 32 V1 na IDE Arduino - fonte: O Autor

A biblioteca "heltec.h"disponibiliza alguns métodos importantes para a comunicação LoRa:

```
1  LoRa.setSPIFrequency(frequency); // Inicia a biblioteca e seleciona a
       frequencia da comunicacao.
2
3  LoRa.beginPacket(); LoRa.endPacket(); // Inicia e finaliza um pacote
       LoRa.
4
5  LoRa.print(message); // Envia uma String.
6  int packetSize = LoRa.parsePacket(); // Retorna um inteiro maior que
       zero caso algum pacote seja recebido.
7
8  int availableBytes = LoRa.available(); // Retorna o numero de bytes
       disponiveis para leitura do pacote recebido.
9
10 byte b = LoRa.read(); // Le o proximo byte do pacote recebido.
11
12 LoRa.setTxPower(txPower); // Seleciona a potencia do sinal de
       transmissao em Decibeis.
13
14 LoRa.setSpreadingFactor(spreadingFactor); // Escolhe o Spreading Factor
       da comunicacao (valores entre 7 e 12).
15
16 LoRa.setSignalBandwidth(signalBandwidth); // Escolhe a largura de banda
       da comunicacao.
17
18 LoRa.setSyncWord(syncWord); // Seleciona a Sync Word utilizada na
       comunicacao.
19
20 LoRa.setPreambleLength(preambleLength); // Determina o tamanho do
       preambulo do pacote.
```

Código 3.1 – Principais métodos da biblioteca "heltec.h"para comunicação LoRa

Também foi utilizada a biblioteca "LoRa.h"disponível em <https://github.com/sandeepmistry/arduino-LoRa> para realizar a comunicação entre a Esp32 Heltec Wifi Lora 32 V1 e V3. Esta biblioteca conta com os mesmos métodos da biblioteca "LoRa.h", com adição dos métodos que podem ser observados no pseudo-código disponível no código 3.2.

3.3.2 Comunicação LoRa com a placa Esp32 Heltec Wifi Lora 32 V3

Para programar a placa Esp32 Heltec Wifi Lora 32 V3, além do driver e biblioteca descritos na sessão anterior, é necessário selecionar a placa na IDE Arduino, como na figura 23.

```
1    LoRa.setPins(CS,RST,DIO0); // Determina os pinos utilizados pelo
         modulo LoRa
2
3    LoRa.parsePacket(); // Retorna o tamanho do pacote recebido. Se for
         maior que zero, significa que um pacote LoRa foi recebido.
4
5    LoRa.readString(); // Retorna a proxima String do pacote.
```

Código 3.2 – Principais métodos da biblioteca "heltec_unofficial.h"para comunicação LoRa

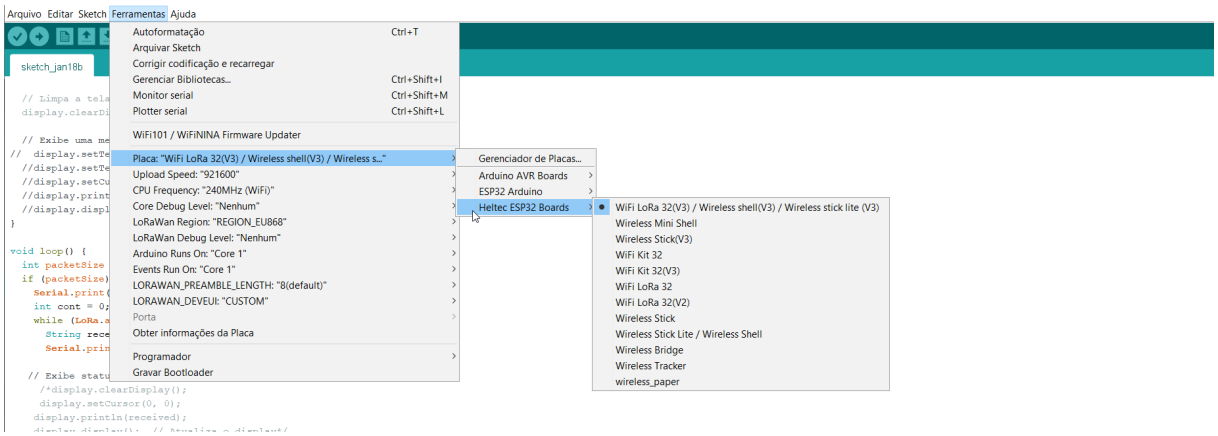

Figura 23 – seleção da placa Esp32 Heltec Wifi Lora 32 V3 na IDE Arduino - fonte: O Autor

Para a comunicação LoRa, foi utilizada a biblioteca "heltec_unofficial.h". Ela contém os seguintes métodos que foram utilizados:

```
1    radio.begin(); Inicia o modulo LoRa.
2
3    radio.setDio1Action(rx); Determina uma funcao de callback para ser
         chamada quando receber uma mensagem LoRa.
4
5    radio.setFrequency(Frequency); //Seleciona a frequencia da
         comunicacao em megahertz.
6
7    radio.setBandwidth(SignalBandwidth);//Seleciona a largura de banda
         em quilohertz.
8
9    radio.setSpreadingFactor(SpreadingFactor);//Seleciona o spreading
         factor entre 7 e 12.
10
11   radio.setOutputPower(OutputPower); //Seleciona a potencia do sinal
         transmitido em decibeis.
12
13   radio.setPreambleLength(PreambleLength); /Determina o preambulo do
```

```
        pacote LoRa.
14
15      radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF); //Permite que o
            modulo comece a receber mensagens LoRa.
16
17      radio.readData(rxdata); //Le pacotes LoRa recebidos e armazena na
            string rxdata.
18
19      _radiolib_status; //Variavel que indica se houve algum erro na
            comunicacao LoRa. Caso nao haja, o valor armazenado sera a
            constante RADIOLIB_ERR_NONE.
```

Código 3.3 – Principais métodos da biblioteca "heltec_unofficial.h"para comunicação LoRa

3.3.3 Programação da placa Esp32 WROOM DevKit V1

Para programar a placa Esp32 WROOM DevKit V1 da Espressif, é necessário instalar a biblioteca da figura 24 e selecionar a placa como na figura 25 na IDE Arduino.

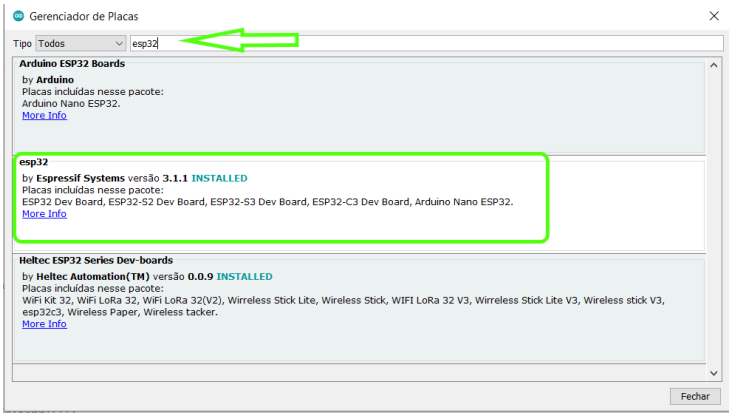

Figura 24 – instalando a placa Esp32 WROOM DevKit V1 na IDE Arduino - fonte: O Autor

3.3.3.1 Programação do módulo LoRa E220-900T22D

Para utilizar o módulo LoRa, foi utilizada a biblioteca "LoRa_E220.h", disponível em <https://github.com/xreef/EByte_LoRa_E220_Series_Library>. A biblioteca conta com múltiplos códigos de exemplo para a configuração e uso do módulo.

Primeiramente, é importante ressaltar que, para o funcionamento do módulo em conjunto com a Esp32 WROOM DevKit V1, é necessário chamar os seguintes métodos no código:

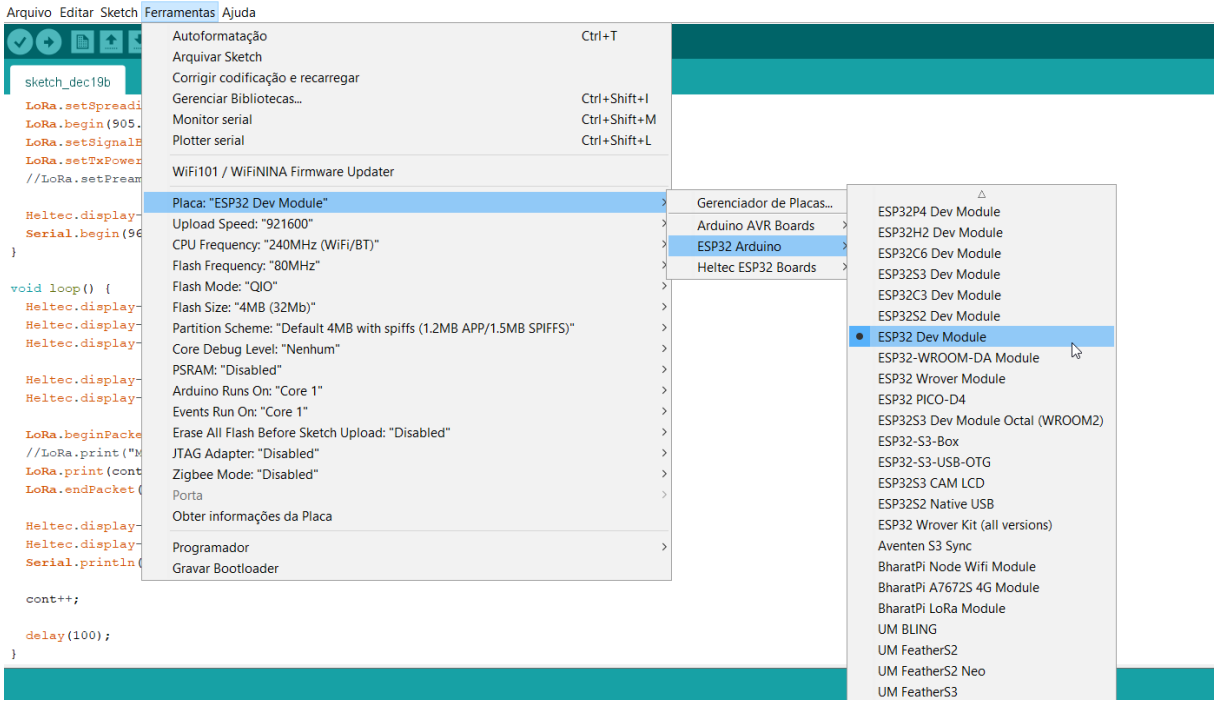

Figura 25 – Selecionando a placa Esp32 WROOM DevKit V1 na IDE Arduino - fonte: O Autor

```
1  LoRa_E220 e220ttl(16, 17, &Serial2, 18, 21, 19, UART_BPS_RATE_9600);
       // Construtor da classe LoRa_E220 da biblioteca.
2
3  Serial2.begin(9600, SERIAL_8N1, 16, 17); // Iniciar as portas
       seriais da placa conectadas ao modulo.
4
5  #define FREQUENCY_915 // Constante necessaria para determinar a
       frequencia da comunicacao LoRa, muito importante neste modelo do
       modulo.
6
7  e220ttl.begin(); // Inicializa a comunicacao LoRa.
```

Código 3.4 – métodos necessários para o funcionamento do módulo LoRa E220-900T22D

Para configurar o módulo, foi utilizado o código exemplo "01_getConfiguration"com as alterações para o funcionamento na Esp32 WRROM DevKit v1 descritas anteriormente. O resultado pode ser observado na figura 26

Em seguida, foi utilizado o código exemplo "01_setConfiguration"da biblioteca para alterar o sync word da comunicação, com a variável configuration.ADDL, e a taxa de transmissão SPED.airDataRate da struct configuration.

Para a comunicação entre os módulos, foram utilizados os métodos e variáveis da biblioteca que podem ser observados no pseudo-código disponível no código 3.5.

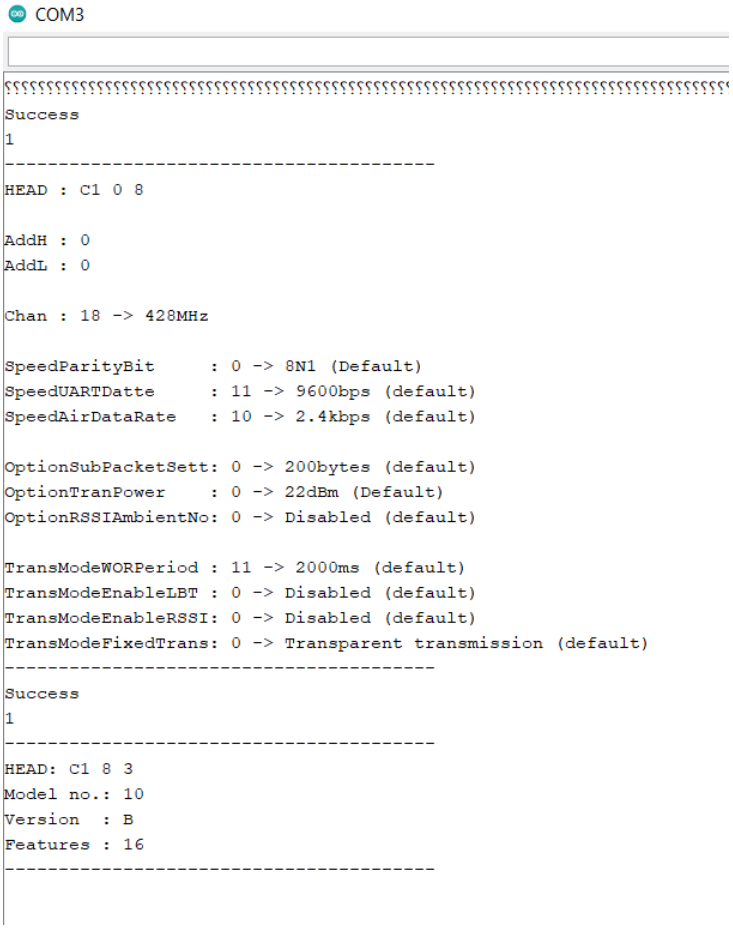

Figura 26 – Configurações iniciais do módulo - fonte: O Autor

```
1    e220ttl.available(); // Verifica se alguma mensagem chegou,
        retornando a quantidade de bytes recebidos.
2
3    e220ttl.receiveMessage(); // Retorna um ResponseContainer com o
        status e dados da mensagem recebida.
4
5    rc.status.code; // Tem valor diferente de 1 caso ocorra algum erro
        na leitura da mensagem.
6
7    rc.data; // Armazena a mensagem LoRa recebida.
8
9    e220ttl.sendMessage(message); // Envia a String message. Tambem
        retorna um ResponseContainer.
```

Código 3.5 – principais métodos para a comunicação LoRa com a biblioteca "e220.h"

3.3.4 Armazenamento e Monitoramento dos dados

Para armazenar e monitorar as medições dos sensores no box, espaço onde a equipe de monitoramento se posicionaria na competição, foi utilizado um programa em Python que lê uma porta serial definida na chamada do programa utilizando a biblioteca "pyserial". O programa salva cada medição recebida em um arquivo .csv, junto ao instante em que foi lida. O programa também utiliza a biblioteca "matplotlib"para desenhar um gráfico para cada uma das medidas à partir dos valores recebidos.

Para diferenciar os dados dos diferentes sensores que são passados pela mesma porta serial no receptor, além do valor da medição, o programa também lê o nome de cada medida e um código definido pelo usuário ao iniciá-lo. Desta forma, a chamada do programa é feita com os seguintes parâmetros:

- porta_serial: portal serial conectada ao dispositivo. Exemplo: COM3
- baud_rate: baud rate da comunicação serial realizada com o dispositivo: Exemplo: 115200
- palavra_chave: palavra chave utilizada para detectar se a string enviada pela porta serial é o valor de uma medição do veículo da equipe. Exemplo: senha
- tipo_dados: string concatenada contendo uma lista de todos os tipos de medições que o programa deve ler separados por vírgula. Para cada tipo de medição, deve ser determinado uma chave que informa a qual medida a medição se refere. Exemplo: t:temperatura,r:rpm,r:RSSI.
- arquivo_csv: Nome base dos arquivos .csv que são criados para armazenar os valores de medições recebidos. Para cada medida é criado um arquivo diferente.

Exemplo de comando para inicializar o programa: python3 readserialandplot.py COM3 115200 senha t:temp,s:rssi teste.csv

4 Experimentos e Análise de Resultados

Neste capítulo, são descritos a série de testes realizados com o equipamento a fim de analisar o desmepenho de diferentes sistemas e parâmetros de comunicação LoRa.

4.1 Testes Iniciais

Nesta seção, são descritos os testes inciais realizados para cada sistema de comunicação LoRa implementado utilizando os equipamentos de hardware e soluções de software descritos no capítulo 3, com o principal objetivo de obter o maior alcance para cada configuração diferente.

4.1.1 Testes iniciais com LoRa

Para um primeiro teste, foram utilizadas as duas placas Esp32 Heltec LoRa Wifi V1 com o objetivo de verificar a implementação e o alcance ponto a ponto da tecnologia LoRa.

Para isto, foram escritos dois algorítmos simples na IDE Arduino utilizando a biblioteca heltec.h disponível para estas placas. Um algorítmo designado para o "Sender", placa que irá transmitir a mensagem, e outra para o "Receiver".

O algorítmo "Sender"mantém uma variável "contador"de tipo inteiro, que é acrescida de uma unidade a cada meio segundo, e envia o valor do contador por LoRa. O algorítmo "Receiver"verifica se alguma mensagem foi recebida e a imprime na porta serial e display OLED da placa. Um diagrama de blocos representando este teste pode ser observado na figura 27.

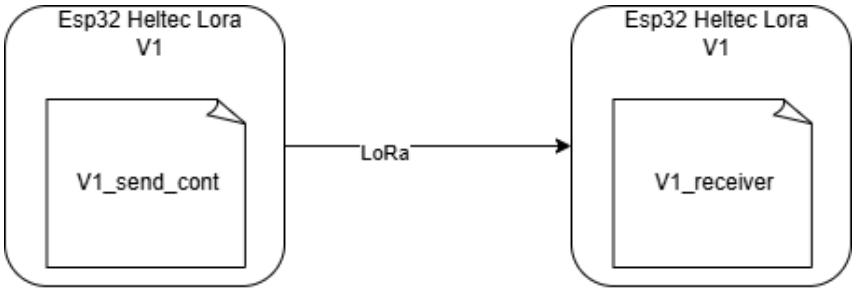

Figura 27 – Diagrama de blocos do teste inicial com LoRa - fonte: O Autor

Também foi necessário configurar os parâmetros da modulação LoRa para a transmissão neste teste. Com o objetivo de conseguir o maior alcance, em troca de uma menor taxa de transmissão, o valor escolhido de Spreading Factor foi 12, que é o maior valor suportado.

A largura de banda escolhida foi a padrão da biblioteca de 125 KHz, e o valor escolhido para a frequência foi de 915 MHz. O Code Rate também foi alterado para 4/8. As antenas que foram utilizadas tinham 17 cm de comprimento.

Após carregar os códigos nas duas placas, foi deixada a placa sender parada em frente ao galpão da equipe Baja Ufscar, enquanto a placa reveiver foi transportada num veículo automóvel pela rua do galpão, verificando até que ponto era possível se afastar da primeira placa sem que o receiver deixasse de receber e disponibilizar as mensagens.

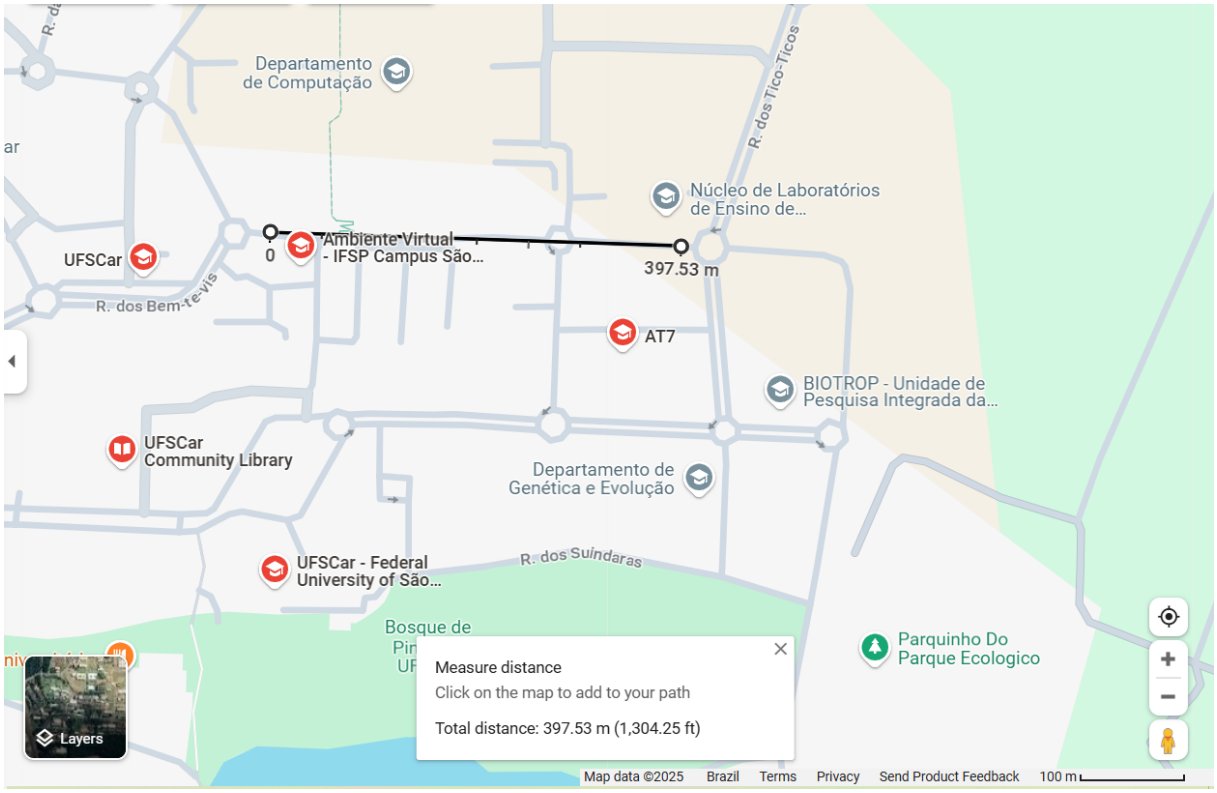

Figura 28 – Trajeto percorrido durante o primeiro teste com os equipamentos LoRa - fonte: O Autor

A figura 28 representa o trajeto percorrido. O alcance máximo atingido foi de aproximadamente 397 m. Durante o trajeto, o sinal da transmissão foi perdido a uma distância de aproximadamente 300m e retornou a partir da distância de 345m. A figura 29 mostra uma visão de satélite da pista da FATEC em São José dos campos onde a etapa nacional da competição Baja SAE ocorre. A distância máxima entre o box mais distante da pista, e o ponto mais longe da mesma é de aproximadamente 535 m. Isto significa que, para os parâmetros de transmissão utilizados neste teste, o veículo ficaria fora do alcance durante algum tempo em cada volta da prova.

Figura 29 – Distância entre o box e o ponto oposto da pista da FATEC em São José dos Campos - Fonte: ARTHUR SANTIAGO NETO

4.1.2 Testes iniciais com o módulo LoRa E220-900T22D

O mesmo teste foi realizado com duas placas Esp32 WROOM DevKit V1 conectadas a um módulo E220 cada. Para isto, dois códigos foram escritos na IDE Arduino utilizando a biblioteca LoRa_E220.h, um para o sender, que enviará o contador, e outro para o receiver, que irá receber a mensagem. A figura 30 contém um diagrama de blocos representando a comunicação entre os dispositivos.

Com este módulo, a distância alcançada foi de aproximadamente 430m, um pouco maior do que a distância alcançada com a placa da Heltec. o trajeto percorrido pode ser obvservado na figura 31.

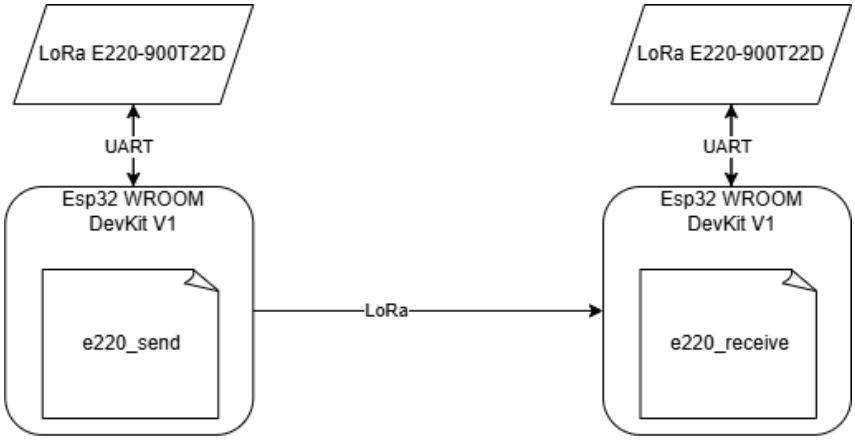

Figura 30 – Diagrama de blocos representando a comunicação nos primeiros testes com o módulo LoRa E220-900T22D - fonte: O Autor

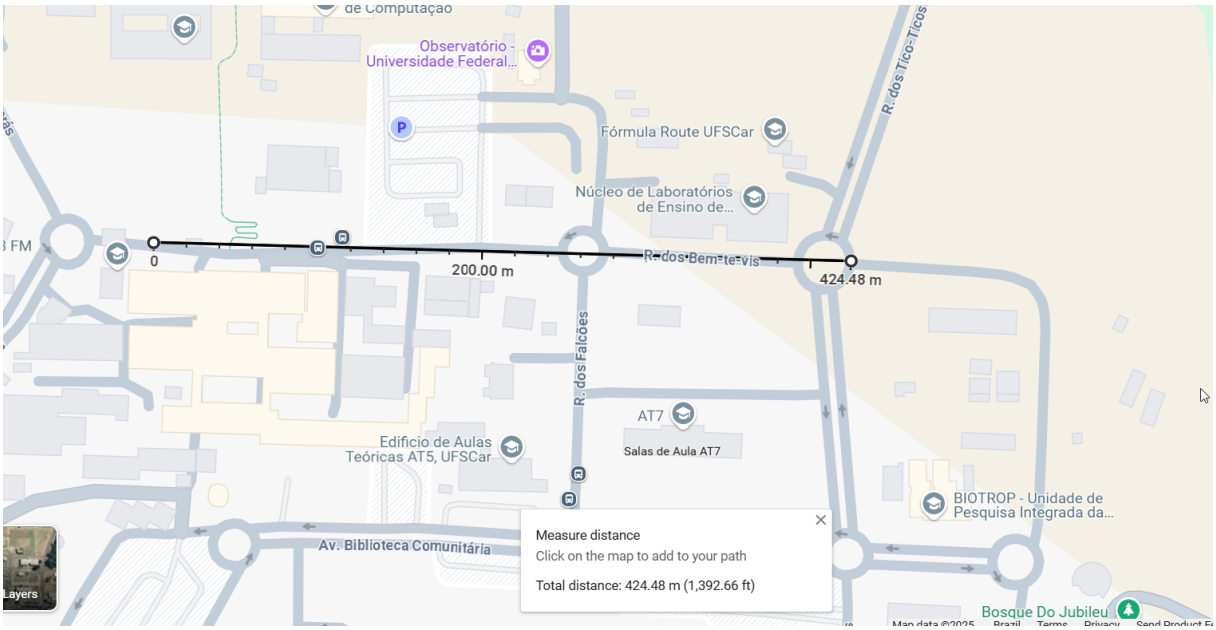

Figura 31 – Distância do trajeto percorrido durante o teste com o módulo LoRa E220 - fonte: O Autor

4.1.3 Testes com dispositivo intermediário

Para atingir um alcance maior do que o atingido nos testes iniciais ponto a ponto entre os dispositivos, foi realizado um teste utilizando três dispositivos LoRa.

Neste teste, um terceiro dispositivo foi inserido no em um ponto do trajeto percorrido pelo veículo de forma que ele recebe o sinal da placa com o algorítmo "sender"e re-transmite a mensagem para a placa com o algorítmo "receiver".

Para que receiver não receba diretamente a mensagem do sender sem passar pelo intermediário, uma diferente Sync Word foi utilizada para a comunicação entre cada um dos dispositivos:

O dispositivo no galpão da Baja UFSCar foi carregado com um código para ler mensagens LoRa com a Sync Word 0x12, enquanto o dispositivo no veículo foi carregado com um código para enviar mensagens com a sync word 0x21.

O terceiro dispositivo (intermediário) foi carregado com um código capaz de ler mensagens com a Sync Word 0x21 e transmiti-las utilizando a Sync Word 0x12. Desta forma, o dispositivo no box não é capaz de ler as mensagens transmitidas pelo dispositivo no veículo mas consegue receber as mensagens re-transmitidas pelo dispositivo intermediário. Este processo de trocar a Sync Word do dispositivo foi atingido utilizando a função LoRa.setSyncWord() da biblioteca "LoRa.h".

Com este sistema, foi possível atingir um alcance de mais de 550 metros, cumprindo o objetivo de cobrir toda a distância do box até o ponto mais distante da pista da competição nacional do SAE Baja. A figura 32 mostra a distância de comunicação atingida no mapa.

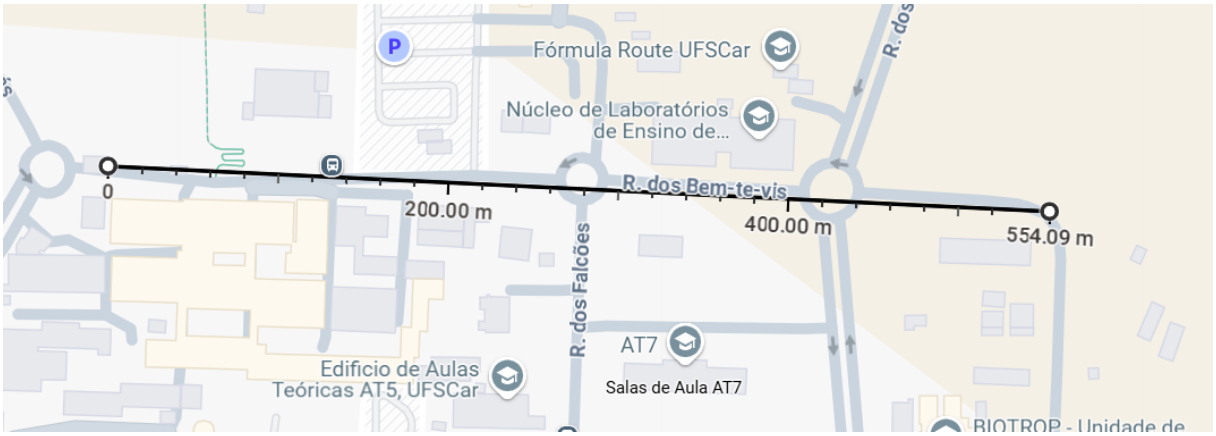

Figura 32 – Distância do trajeto percorrido durante o testes inicias com intermediário - fonte: O Autor

4.2 Testes Finais com os sistemas de comunicação

Para a última seção de testes, o objetivo principal foi refazer os testes realizados anteriormente, no mesmo local, porém, utilizando um dos sensores do carro para enviar dados reais e o programa em Python para salvar e vizualiza-los. A fim de encontrar quais são os melhores parâmetros de cada sistema para a aplicação, foi realizada uma volta no percurso para cada valor diferente de Spreading Factor e Time on Air selecionado. Adicionalmente, também foi registrado o RSSI e taxa de pacotes recebidos a cada 55 metros, a fim de identificar qual dos sistemas de comunicação e quais valores de Spreading Factor teriam melhor desempenho. Os parâmetros largura de banda, Code Rate e TxPower selecionados para comunicação LoRa utilizando as placas Esp32 Heltec LoRa WIFI 32 V1 e V3 nestes testes foi a padrão das bibliotecas "LoRa.h"e "heltec_unnoficial.h", ou seja, 125kHz, 4/5 e 14 dB, respectivamente.

Para realizar o cálculo de pacotes recebidos, foi rodado cada algorítmo de sender de cada um dos 3 sistemas, medindo o tempo que leva para enviar um pacote utilizando a função millis() da biblioteca Arduino.h, no começo e fim do loop de execução. Em seguida, a partir das medições salvas pelo algorítmo em Python, é possível observar o momento em que cada mensagen foi recebida.

Com isto, pode-se verificar aproximadamente quantas mensagens foram enviadas e recebidas em um intervalo de tempo. Para estes testes, O intervalo de tempo utilizado para a medição de número de mensagens foi de três segundos, com exceção nos casos em que o tempo de duração de transmissão de uma mensagem foi superior ao de 1 segundo. Para estes casos, o intervalo utilizado foi de 6 segundos.

Durante os testes, foi observado que em algumas distâncias maiores, o sinal era perdido por alguns segundos. No registro destas medidas, foi inserido um asterísco indicando que durante um intervalo maior de tempo, o número de mensagens recebidas se manteria o mesmo. Isto significa que o número de mensagens recebidas para estas distâncias não reflete a real velocidade de transmissão do sistema.

Dado que os sensores de RPM, pressão do fluido de freio e velocidade tem instalação mais complexa no veículo, o sensor escolhido para gerar os dados nos testes foi o sensor de temperatura. Este sensor foi conectado a placa com um algorítmo que lê o sensor e envia a temperatura por comunicação LoRa.

Para o envio das mensagens contendo a temperatura, foi seguido o seguinte formato: "$senha,$medição,$valor", onde "$senha"foi substituído por uma palavra chave que é usada para verificar se a mensagem pertence ao sistema de comunicação. Para este exemplo, a senha utilizada foi a string "pass". O campo "$medição"contém uma string informando qual o tipo de dado que está sendo enviado. Isto é útil quando mais de um sensor é inserido no sistema e é necessário diferenciar ao que se refere cada medida. Por fim, o campo "$valor"é preenchido com o valor da medição. Neste caso, a temperatura.

4.2.1 Teste ponto a ponto com as placas Esp32 Heltec LoRa WIFI 32 V1

Nesta bateria de testes, foram utilizadas duas placas Esp32 Heltec LoRa WIFI 32 V1 e uma placa Esp32 WROOM DevKit V1.

A placa Esp32 DevKit foi conectada ao sensor de temperatura Max6675 com comunicação serial do tipo SPI e à placa Esp32 Heltec com algorítmo de "sender"por comunicação serial do tipo UART. Este sistema com placa Esp32 DevKit, Esp32 Heltec e sensor de temperatura Max6675 foi inserido em um veículo automóvel que o transportava pelo percurso. Uma foto deste sistema dentro do carro pode ser observado na figura 33.

Por fim, a segunda placa Esp32 Heltec foi carregada com um algorítmo de receiver e mantida imóvel do lado de fora do galpão da equipe Baja UFSCar conectada a um notebook por comunicação serial do tipo UART. No notebook, foi iniciado o código em Python a fim de gravar os dados das medidas de temperatura e RSSI ao longo do tempo.

Figura 33 – Foto retirada durante os testes finais ponto a ponto com Esp32 Heltec LoRa WIFI 32 - fonte: O Autor

Um diagrama de blocos representando a relação entre cada componente pode ser observada na figura 34.

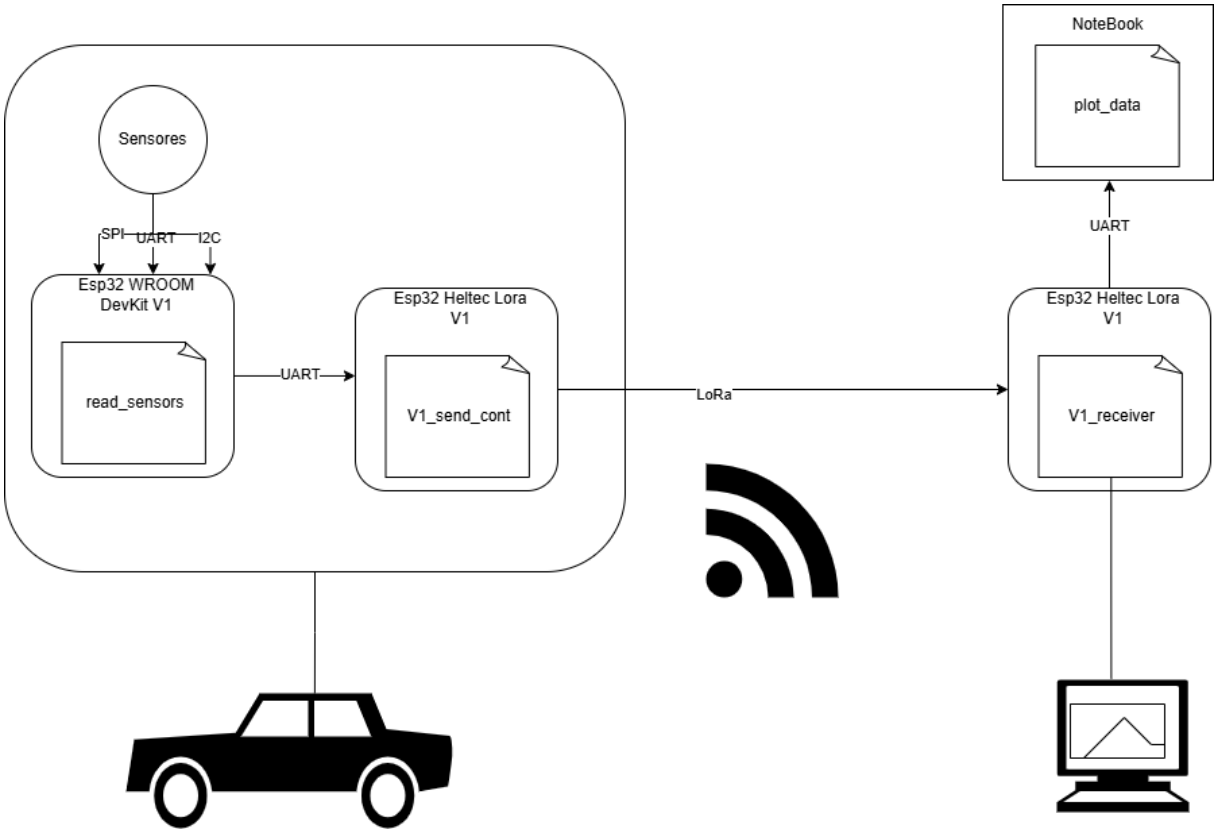

Figura 34 – Diagrama de blocos dos testes finais ponto a ponto com placa Esp32 Heltec LoRa WIFI 32 - fonte: O Autor.

O percurso foi percorrido três vezes, variando o valor do parâmetro Spreading Factor (SF) entre 8, 10 e 12. A tabela 1 apresenta os valores de RSSI e número de mensagens recebidas para cada distância percorrida. Nas últimas linhas da tabela, foi anotado o tempo de envio médio de uma mensagem para cada valor de Spreading Factor, assim como quantas mensagens são enviadas por intervalo de tempo. Os gráficos de RSSI e temperatura para cada passagem pelo percurso podem ser observados na figura 35.

Pode-se observar, a partir dos gráficos impressos pelo algorítmo em Python, que em alguns pontos, houveram picos de temperatura. Isto ocorreu devido ao mal contato ocasional entre a placa com o sensor e a placa com o módulo LoRa durante os testes.

O programa em Python também encontrou alguns problemas ao imprimir as legendas dos gráficos, o que definiu um ponto de melhora para o sistema de telemetria para a versão final que será utilizada em competição.

Distância (m)	**SF = 8**		**SF = 10**		**SF = 12**	
	RSSI	**msgs**	**RSSI**	**msgs**	**RSSI**	**msgs**
0	-99	10	-88	6	-91	4
55	-97	11	-105	7	-105	5
110	-104	12	-111	3	-109	5
165	-106	10	-112	4	-111	4
220			-112	10	-112	5
275			-111	10		
330			-112	10		
385			-111	1*		
440						
495						
550						
Tempo de Envio (ms)	198		447		1486	
Taxa de Envio	15,2 mensagens/3s		6,7 mensagens/3s		4 mensagens/6s	

Tabela 1 – Valores de RSSI e taxa de pacotes recebidos nos testes ponto a ponto com a placa ESP32 Heltec LoRa WIFI 32 em função da distância.

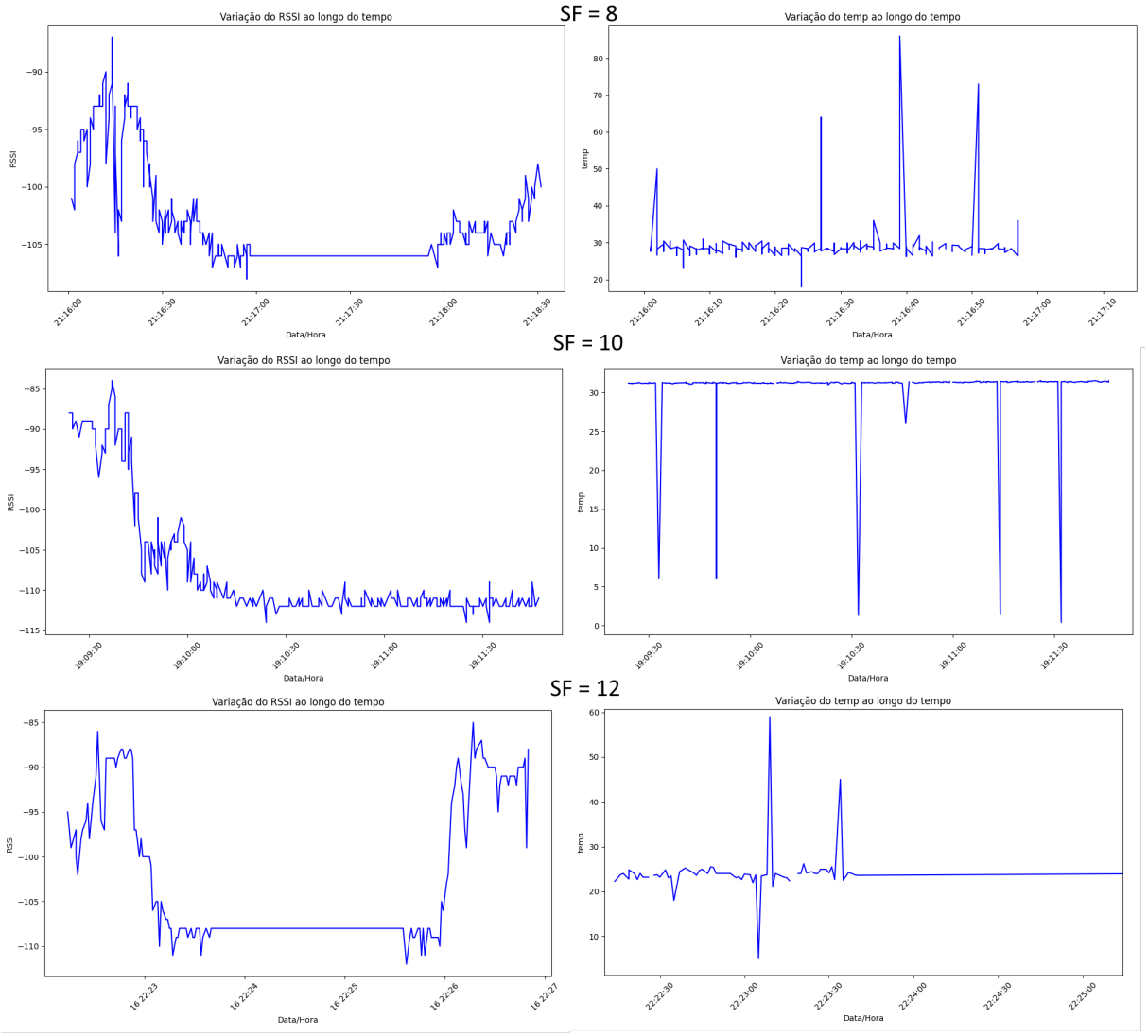

Figura 35 – Gráfico gerado para a temperatura e RSSI ao longo do tempo durante os testes ponto a ponto com a placa Esp32 Heltec LoRa WIFI 32 - fonte: O Autor.

4.2.2 Teste com dispositivo LoRa intermediário

Nesta bateria de testes, assim como nos testes da subseção anterior, foi transportada, em um veículo automóvel, a placa Esp32 WROOM DevKit V1 realizando a leitura de temperatura pelo sensor max6675 e enviando por comunicão serial UART para uma placa Esp32 Heltec LoRa WIFI 32 V1. A placa, por sua vez, envia as medições por comunicação LoRa. Porém, ao invés de uma segunda placa Esp32 Heltec LoRa WIFI 32 V1 receber estes dados e transmiti-los para um notebook com o programa em Python em frente ao galpão da equipe, a segunda placa Esp32 Heltec V1 foi colocada no percurso a uma distância de aproximadamente 165m do galpão com um código capaz de receber as mensagens LoRa da primeira placa Esp32 Heltec V1 e retransmiti-las com uma Sync Word diferente.

Uma placa Esp32 Heltec LoRa WIFI 32 V3 foi colocada em frente ao galpão realizando a tarefa de receber os dados retransmitidos e enviá-los via comunicação serial UART para o notebook com o código em Python. O diagrama de blocos da figura 36 representa o sistema utilizado nesta bateria de testes.

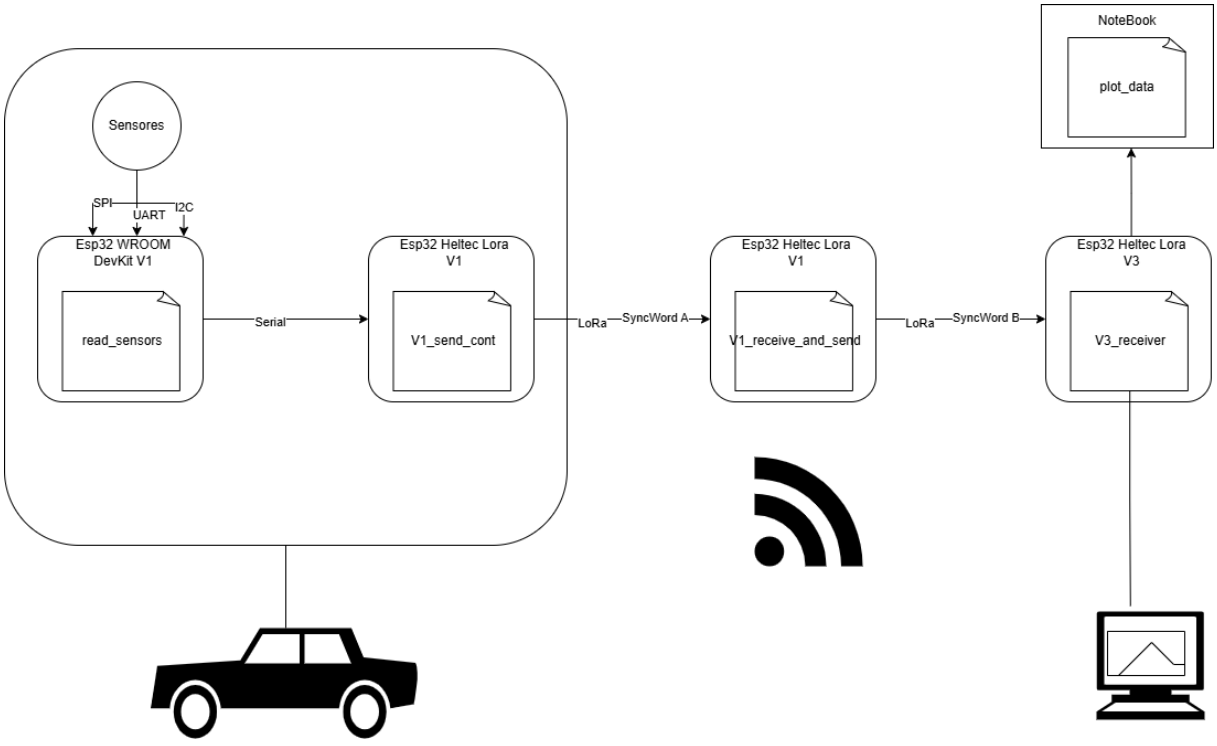

Figura 36 – Diagrama de blocos dos testes finais de comunicação com dispositivo intermediário - fonte: O Autor.

Por fim, foi medido o RSSI e número de mensagens recebidas em cada intervalo de tempo a cada 55m de distância em relação ao galpão da equipe. Os valores destas mediçoes foram inseridos na tabela 2, e os gráficos de temperatura e RSSI gerados podem ser observados na figura 37.

Distância (m)	**SF 8**		**SF 10**		**SF 12**	
	RSSI	**msgs**	**RSSI**	**msgs**	**RSSI**	**msgs**
0	-78	8	-89	3	-77	3
55	-80	8	-90	4	-79	3
110	-78	8	-89	4	-82	3
165	-77	9	-89	4	-86	3
220	-82	9	-89	4	-86	3
275	-86	8	-89	4	-85	3
330	-87	8	-90	4	-87	3
385	-78	2	-92	4	-79	3
440	-77	1*	-83	1	-80	3
495	-81	2*	-95	3	-79	3
550	-82	1*	-86	1*	-81	2
Tempo de envio (ms)	**198**		**447**		**1486**	
Taxa de envio	**15,2/3s**		**6,7/3s**		**4/6s**	

Tabela 2 – Valores de RSSI e taxa de pacotes recebidos nos testes de comunicação com dispositivo intermediário em função da distância e diferentes valores de Spreading Factor

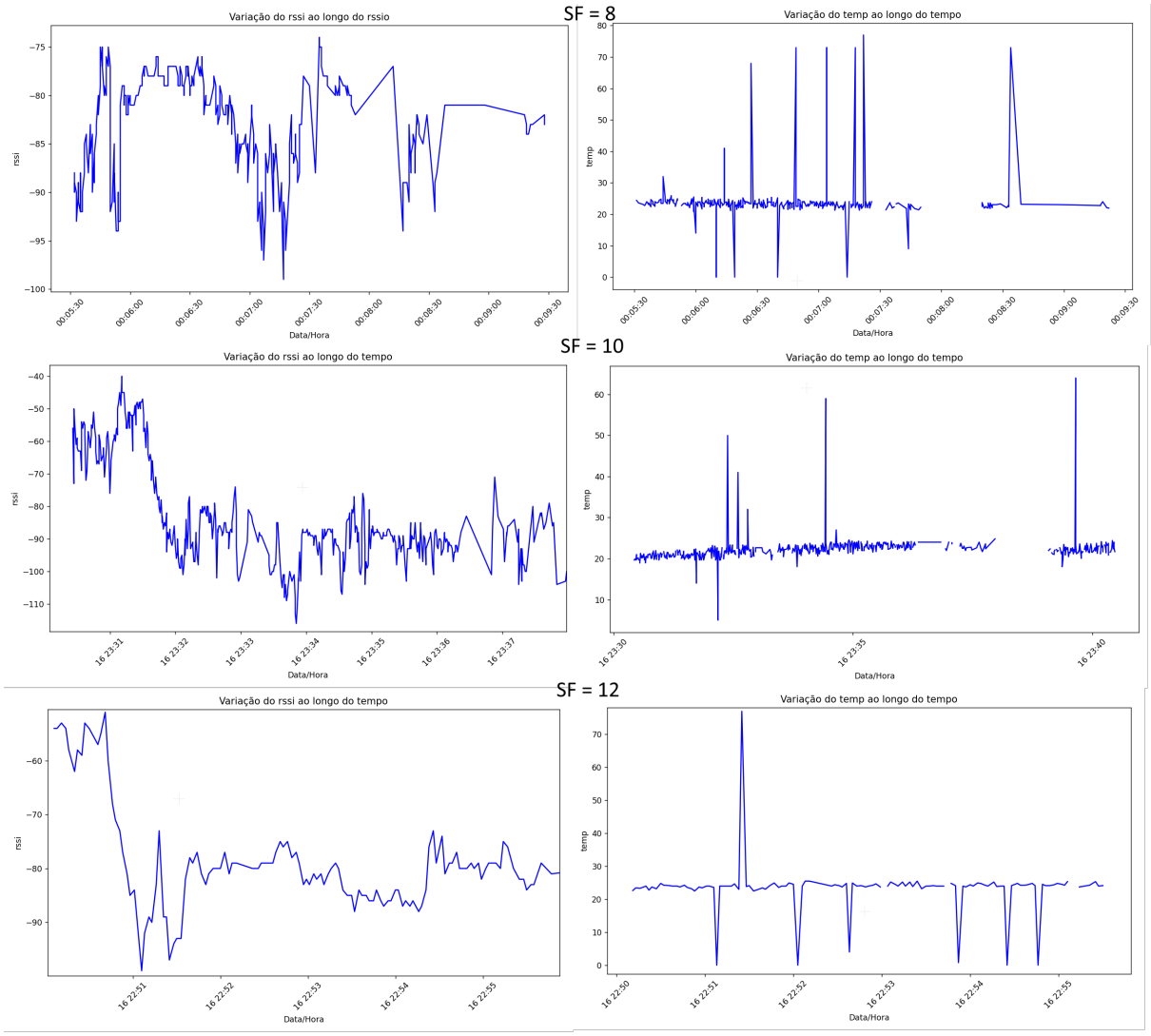

Figura 37 – Gráfico gerado para a temperatura e RSSI ao longo do tempo durante os testes de comunicação com dispositivo intermediário - fonte: O Autor.

4.2.3 Teste final com dispositivo LoRa E220-900T22D

Nesta bateria de testes, foram utilizadas duas placas Esp32 WROOM DevKit V1. Uma das placas foi conectada ao sensor de temperatura max6675 e a um módulo LoRa E220-900T22D para enviar medições de temperatura enquanto foi transportada pelo percurso por um veículo automóvel. A segunda placa foi deixada em frente ao galpão da equipe Baja UFSCar conectada a um notebook com o código em Python salvando as medições em um arquivo e outro módulo LoRa E220-900T22D para receber as mensagens transmitidas. Uma representação deste sistema pode ser observada no diagrama de blocos da figura 38.

O percurso foi percorrido duas vezes com os valores de 62,5 kbps e 2,4 kbps para o parâmetro air data rate. Os resultados das medições de RSSI e número de pacotes por intervalo de tempo ao longo da distância percorrida pode ser observado na tabela 3, enquanto o RSSI e temperatura em função do tempo podem ser observados na figura 39.

A partir do gráfico de RSSI, foi possível observar que o valor desta medida retornado pela biblioteca do módulo LoRa E220 é positivo. Para a melhor leitura deste parâmetro, seria possível realizar o tratamento desta medida, invertendo o sinal do valor lido pelo módulo.

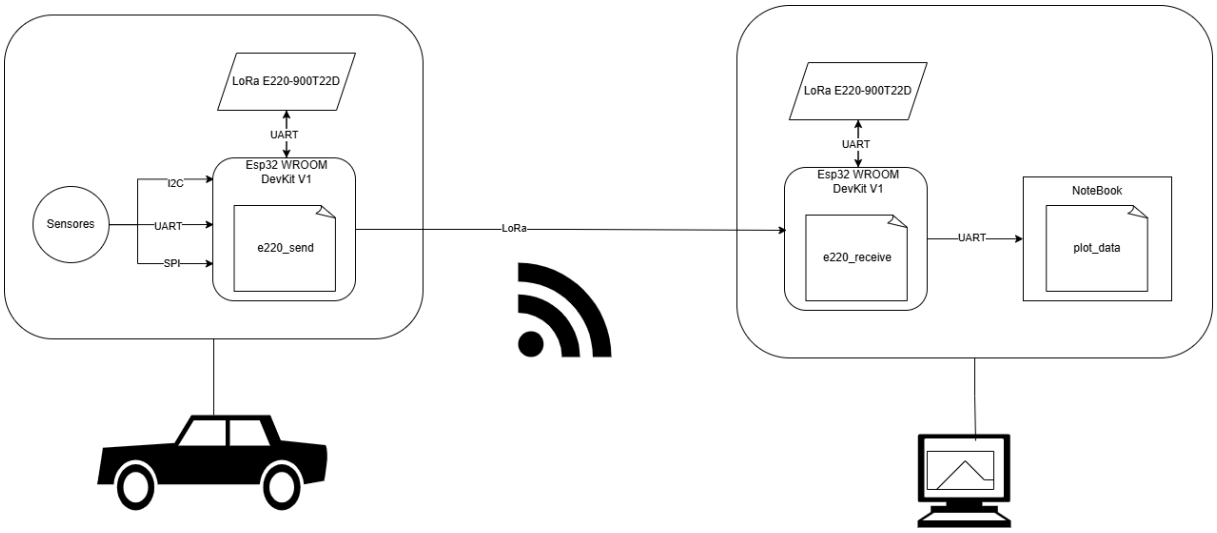

Figura 38 – Diagrama de blocos dos testes finais de comunicação com dispositivo LoRa E220-900T22D - fonte: O Autor.

Distância (m)	**62.5 kbps**		**2.4 kbps**	
	RSSI	**msgs**	**RSSI**	**msgs**
0	-54	2	-56	3
55	-54	2	-52	3
110	-53	2	-52	2
165	-53	2	-51	3
220	-54	2	-54	2
275	-57	2	-53	3
330	-48	2	-52	2
385			-51	2
440			-52	3
495			-52	2
550			-49	2
Tempo de envio (ms)	**119**		**429**	
Taxa de envio	**25,2 mensagens/3s**		**14 mensagens/6s**	

Tabela 3 – Valores de RSSI e taxa de pacotes recebidos nos testes finais de comunicação com dispositivo LoRa E220-900T22D em função da distância e diferentes valores de air data rate

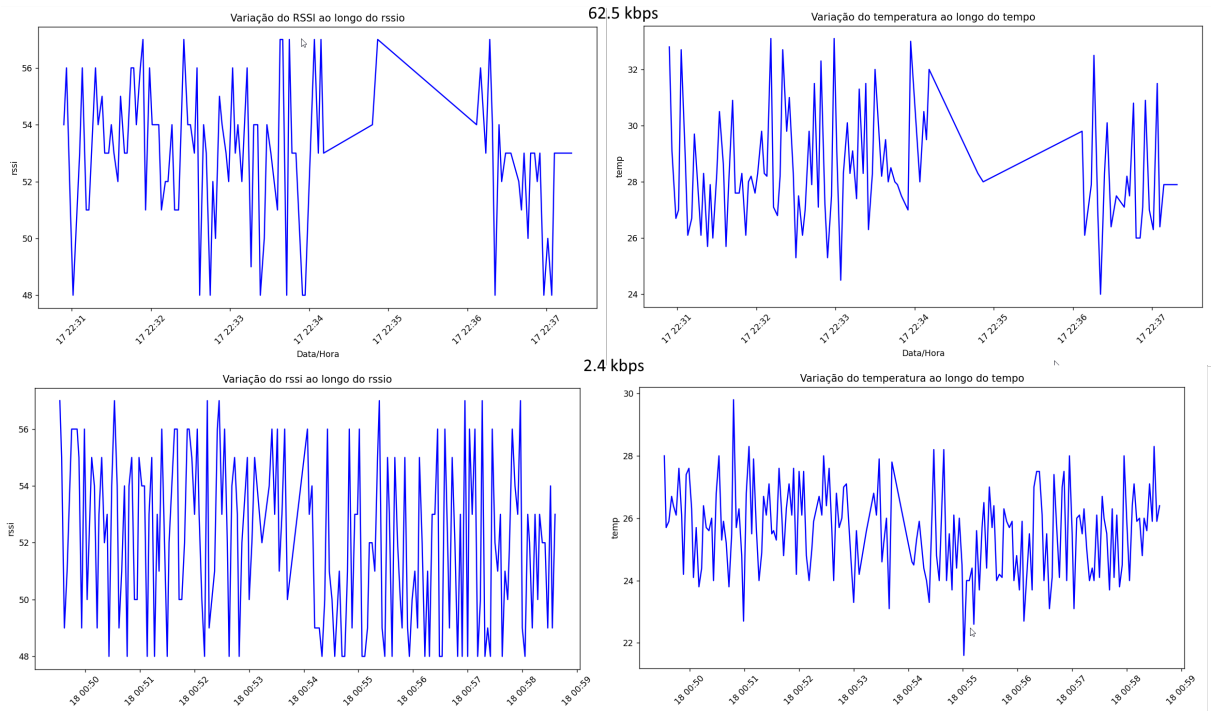

Figura 39 – Gráfico gerado para a temperatura e RSSI ao longo do tempo durante os testes finais de comunicação com dispositivo LoRa E220-900T22D - fonte: O Autor.

4.3 Análise de de Resultados

4.3.1 Alcance versus velocidade de transmissão

A partir dos testes realizados na seção anterior, é possível analisar os diferentes sistemas em relação a dois principais efeitos: 1- alcance e 2- número de mensagens recebidas por segundo.

Observando as tabelas 1 e 2, é possível confirmar que aumentar o parâmetro Spreading Factor da comunicação permite um alcance maior. A única exceção foi a medição nos testes ponto a ponto com as placas Esp32 Heltec LoRa WIFI 32 V1 com o Spreading Factor 10. Isto possivelmente ocorreu porque este teste foi realizado em um outro local, próximo a UFSCar, onde não existem prédios próximos ao percurso. Resultando em uma linha de visão entre as duas placas muito melhor. Isto pode indicar uma grande influência da presença de obstáculos no alcance final do sistema, uma vez que o teste, neste local, levou a um alcance 1,5 vezes maior do que o alcance no local com mais obstáculos e com Spreading Factor maior. Isto revela a importância de conhecer o local em que se deseja realizar a comunicação em relação a presença de obstáculos.

Para garantir que a comunicação funcione tanto em um local com menos obstáculos, quanto em um local com mais, é importante selecionar um Spreading Factor de valor maior, como 10, 11 e 12.

Também é possível perceber que, a medida que o Spreading Factor aumenta, o número de mensagens recebidas por intervalo de tempo caiu. Isto se deve ao fato de que, quanto maior o Spreading Factor, maior o tempo de envio de cada chirp de um pacote na comunicação LoRa, e assim, maior o tempo necessário para enviar cada mensagem.

O mesmo pôde ser observado na tabela 3 para o módulo LoRa E220-900T22D, que utiliza como parâmetro o air data rate em bps. Neste caso, quanto maior o bps, maior o número de mensagens recebidas e mais rápida é a transmissão, porém, menor é o alcance. Isto ocorre porque o cálculo de bits por segundo de transmissão é realizado através da equação 2 da fundamentação teórica (seção 2.3.3.1), onde um aumento em bps reflete na diminuição de Spreading Factor e/ou Bandwidth e Code Rate. Desta forma, pode-se supor que se os mesmos testes fossem realizados variando os parâmetros de Bandwidth e/ou Code Rate, os resultados apontariam para esta mesma relação entre velocidade de transmissão e alcance.

Por fim, analisando a tabela 2 em relação à tabela 1, pode-se observar o aumento de alcance atingido por adicionar um dispositivo intermediário no meio do percurso. Também é possível ressaltar que este aumento de alcance também resultou em um menor número de mensagens recebidas por intervalo de tempo, refletindo em um maior tempo de transmissão. Isto ocorre devido ao fato de que, neste caso, a comunicação é dividida em duas partes: uma entre o dispositivo no carro e o intermediário, e outra entre o dispositivo intermediário e o dispositivo no box. Isto significa que uma mesma mensagem é transmitida

duas vezes até que seja recebida. Como o algorítmo do dispositivo intermediário apenas recebe uma nova mensagem quando a anterior for retransmitida, o tempo de envio da mensagem deve levar em consideração, tanto o tempo de transmissão da primeira, quanto o da segunda parte da comunicação.

4.3.2 Alcance em função de Code Rate e TxPower

A partir dos testes ponto a ponto com duas placas Esp32 Heltec LoRa Wifi V1, foi possível observar que, para os testes com Spreading Factor 12, o alcance atingido foi diferente entre os testes iniciais e finais. Isto se deve ao fato de que o Code Rate e TxPower utilizados nos testes foi diferente. Nos testes iniciais com este sistema, o Code Rate utilizado foi de 4/8 e txPower foi o máximo suportado de 17 dB. Com esta configuração, foi possível atingir uma distância máxima de comunicação de aproximadamente 400 metros, enquanto nos testes finais, com o mesmo Spreading Factor, porém Code Rate igual a 4/5 e TxPower 14, foi atingido apenas 220 metros aproximadamente.

Isto indica a relevância de outros parâmetros na comunicação além do Spreading Factor.

4.3.3 RSSI em relação à distância de comunicação

Era esperado que o valor de RSSI se distanciasse cada vez mais de zero à medida que a distância do transmissor aumentasse em relação ao receptor. Isto pode ser observado para o sistema com duas Esp32 Heltec LoRa Wifi V1 na tabela 1. O mesmo, porém, não vale para os outros dois sistemas. Para o sistema com dispositivo intermediário, isto pode ser explicado pelo fato de que o RSSI medido foi o da comunicação entre o dispositivo intermediário e o dispositivo do box, onde ambos permaneceram imóveis durante os testes.

Para o sistema com o módulo E220-900T22D, representado pelos dados da tabela 3, o RSSI variou pouco, apresentando um desvio padrão de aproximadamente 2.7 para o 62.5 kps e 1,8 para 2.4 kbps. Isto significa que, apesar do aumento na distância, o RSSI permaneceu constante, o que aponta que a intensidade com que o sinal chegou ao receiver permaneceu a mesma durante todo o teste.

Também é possível observar a diferença de RSSI para cada um dos testes utilizando diferentes equipamentos para o recebimento das mensagens. Nos testes com Esp32 Heltec LoRa Wifi V1, o RSSI teve valor próximo a -100. Já nos testes utilizando a placa Esp32 Heltec LoRa Wifi V1, o RSSI teve valor em torno de -80. Por último, para os testes utilizando o módulo E220-900T22D, o RSSI teve valores em torno de -50. Isto exemplifica a dificuldade em comparar medidas de RSSI de equipamentos diferentes, uma vez que a escala de RSSI é determinada pelo fabricante do equipamento. Ou seja, um sinal que chega a 14 dB pode ter diferentes valores de RSSI para diferentes fabricantes. Isto pode

indicar a insuficiência da medida de RSSI para a medição de desempenho de sistemas de comunicação LoRa.

4.3.4 Medição do tempo de envio para o módulo LoRa E220-900T22D

Durante os testes finais com o módulo LoRa E220-900T22D, o tempo de envio em milissegundos para a transmissão das mensagens foi de 119 ms para a bit rate de 62.5 kbps e de 429 para 2.4 kbps. Isto deveria significar um envio de 25,2 mensagens a cada 3 segundos e 14 mensagens a cada 6 segundos para cada valor de air data rate, respectivamente. Porém, a partir da tabela 3, é possível observar que estas taxas de envio de mensagens foram muito diferentes da taxa de recebimento durante os mesmos intervalos. Isto pode ser atribuído ao fato de que a biblioteca LoRaE220 utilizada na implementação da comunicação com este módulo não bloqueia o fluxo do algorítmo escrito na IDE Arduino enquanto realiza a transmissão de mensagens.

Para medir de forma mais adequada a taxa de envio, é necessário calcular o tempo de envio de uma mensagem a partir da air data rate. Uma data rate de 62.5 kbps significa 7,8 kb/s. Sabendo que a mensagem tem, por padrão da biblioteca, o tamanho de 200 bytes, a transmissão de uma mensagem com este air data rate dura aproximadamente 26 ms, muito menor do que o tempo de envio de 119 medido a partir da diferença de tempo entre o começo e final do loop de execução do algorítmo. Porém, para este caso, como foi inserido um delay de 100 ms entre a chamada de função de envio de mensagens no algorítmo, é suposto que o tempo de envio de mensagens tenha um valor próximo do delay. Logo, a diferença entre a taxa de envio e a taxa de recebimento para este caso, significa uma grande perda de pacotes por este sistema implementado.

Para o caso do data rate de 2.4 kbps, ou seja 300 bytes por segundo, e tamanho de mensagem padrão igual a 200 bytes, as mensagens seriam enviadas em uma taxa de 1,5 por segundo. Uu seja, a taxa de envio de uma mensagem é de aproximadamente 667 ms. Desta forma, o delay inserido no algorítmo não é o bastante para cobrir o tempo de duração do envio de cada mensagem, resultando em um gargalo no módulo de envio de mensagens.

Comparando o número de mensagens recebidas durante os intervalos na tabela 3 com os reais valores de tempo de envio de mensagens calculados, é possível observar que, para ambos os valores do parâmetro air data rate, o número de mensagens recebidas permanece muito menor que o de enviadas. Isto significa uma perda de pacote de aproximadamente 92% para o air data rate de 62,5 kbps e 67% para o air data rate de 2,4 kbps.

Para comparar a perda de pacotes dos testes com o módulo LoRa E220-900T22D em relação aos testes com as placas Esp32 Heltec LoRa Wifi V1 e V3, as tabelas 4 e 5 foram montadas calculando a porcentagem de mensagens recebidas para cada medida de distância e valor de Spreading Factor em relação a taxa de envio em cada intervalo.

Pode-se observar que, embora a taxa de recebimento de mensagens durante os testes com LoRa E220-900T22D permaneça constante ao longo das diferentes medidas de distâncias, a porcentagem de pacotes perdidos é maior do que a mesma porcentagem nos testes utilizando as outras placas, que por sua vez, tem uma taxa de mensagens recebidas e de perda de pacote, em geral, relacionada a distância da comunicação. Isto pode revelar uma menor eficiência na comunicação utilizando o módulo LoRa E220.

Apesar da alta porcentagem de perda de pacotes, pode-se observar que, para os testes que atingiram a distância máxima de 550 metros, a taxa de pacotes recebidos foi muito parecida (em torno de 2 a 3 pacotes a cada 6 segundos) nos diferentes sistemas. Ou seja, embora a eficiência de consistência de pacotes do sistema utilizando as placas Esp32 Heltec LoRa Wifi V1 e V3 com intermediário seja maior, o desempenho relacionado a taxa de mensagens recebidas permanece o mesmo.

Distância (m)	**SF 8**	**SF 10**	**SF 12**
0	34%	10%	0%
55	28%	0%	0%
110	21%	55%	5%
165	34%	40%	0%
220	–	10%	0%
275	–	10%	–
330	–	10%	–
385	–	86%*	–
440	–	–	–
495	–	–	–
550	–	–	–

Tabela 4 – Valores de porcentagem de perda de pacotes para diferentes medidas de distância e Spreading Factor durante os testes finais ponto a ponto com as placas Esp32 Heltec LoRa Wifi V1

4.3.5 Seleção do sistema de comunicação

À partir dos testes, foi possível observar que apenas dois sistemas foram capazes de atingir o alcance de 535 metros exigido para cobrir a distância do Box ao ponto mais extremo da pista da FATEC em São José dos Campos: O sistema com as placas Esp32 Heltec LoRa Wifi V1 e V3 com um dispositivo intermediário retransmitindo as mensagens, e o sistema com o módulo LoRa E220-900T22D. Nesta subseção, será discutido os prós e contras de cada um destes sistemas.

Distância (m)	Spreading Factor 8	Spreading Factor 10	Spreading Factor 12
0	47%	55%	25%
55	47%	40%	25%
110	47%	40%	25%
165	41%	40%	25%
220	41%	40%	25%
275	47%	40%	25%
330	47%	40%	25%
385	87%	40%	25%
440	93%	85%	25%
495	87%	55%	25%
550	87%	85%	67%

Tabela 5 – Valores de porcentagem de perda de pacotes para diferentes medidas de distância e Spreading Factor durante os testes finais de comunicação LoRa com intermediário

Em relação à velocidade de transmissão, ambos os sistemas tiveram aproximadamente o mesmo número de mensagens recebidas a cada intervalo de 6 segundos nas configurações que atingiram o alcance desejado, atingindo 3 mensagens a cada 6 segundos, ou seja, uma taxa de 0,5 mensagem por segundo. Embora, a taxa de pacotes perdidos foi muito diferente: 25% no sistema com dispositivo intermediário versus 67% para o com módulo LoRa E220, para melhorar a taxa de envio e recebimento de mensagens para o sistema com o módulo LoRa E220, seria possível ajustar o tamanho do pacote com a configuração subPacketSetting da biblioteca "LoRa_E220.h"para o valor de 32 bytes, que se encaixa melhor com o tamanho das mensagens enviadas na aplicação. Também poderia ser ajustado o delay entre chamadas de envio de mensagens de modo a aguardar que o módulo termine de enviar uma mensagem antes de solicitar outro envio. Para o air data rate de 2.4 kbps, o valor teórico do delay seria de 35 ms.

Comparando em relação ao consumo de energia, principalmente nos componentes que são embarcados no veículo, ambas as soluções apresentam consumo nominal semelhante. Como foi observado no capítulo 3, estudando os manuais dos componentes, o Esp32 DevKit V1 opera a aproximadamente 80mA, enquanto as placas Esp32 Heltec Wifi Lora 32 V1 e V3 operam em torno de 110mA para o txPower de 14, assim como o módulo LoRa E220-900T22D, que consome 110 mA no valor padrão de potência de transmissão de 22dB. Visto que a bateria utilizada no veículo é de 7A, em um cenário que apenas estes dispositivos consomem esta bateria, ela duraria cerca de 36 horas. Visto que as provas mais longas da competição tem duração de 4 horas corridas, este consumo aparenta ser satisfatório. Para um passo futuro, seria interresante realizar testes medindo o consumo

real de energia de cada um dos sistemas de comunicação. A análise destes testes permitiria selecionar um valor mais adequado de TxPower do dispositivo, visando o melhor balanço entre consumo e alcance de transmissão.

Além do consumo de energia no veículo, a solução utilizando o dispositivo intermediário também necessita de uma fonte de energia. O que leva a um consumo de energia total do sistema mais elevado.

Por fim, a praticidade também é um fator a ser levado em consideração. Para o sistema com intermediário, um lado negativo é que durante as competições, seria necessário buscar um local adequado para deixar o dispositivo. Também seria necessário que algum membro da equipe ficasse responsável pela segurança do dispositivo, garantindo que ele não fosse movido ou manuseado indevidamente. Além disso, o módulo LoRa E220-900T22D abstrai os parâmetros de Spreading Factor, Largura de Banda e Code Rate da tecnologia de comunicação LoRa, fazendo com que a configuração do dispositivo seja mais simples e prática, apenas selecionando o air data rate.

Visto que os principais critérios de análise de desempenho do sistema de telemetria: 1 - alcance, 2 - consumo de energia e 3 - velocidade de transmissão dos sistemas implementados apresentaram resultados semelhantes, devido a maior praticidade do sistema com o módulo LoRa E220-900T22D, este sistema se mostra a melhor escolha para a aplicação de sistema de telemetria para o veículo Baja. Este sistema também pode ser facilmente configurado para provas em locais com distâncias menores entre o Box e o ponto mais extremo da pista, como no local onde é realizada a etapa regional que a equipe Baja UFSCar participa. Observando a figura 40, pode-se observar que a distância máxima é de aproximadamente 116 metros. Para se adequar a esta pista, basta configurar o módulo, selecionando um valor maior de air data rate, como o 62 kbps, diminuindo o alcance e aumentando a velocidade de transmissão de pacotes.

Figura 40 – Distância entre o ponto extremo da pista e os box das equipes na pista da Etapa Regional que a equipe Baja UFSCar participa - fonte: O Autor.

5 Conclusão

Este trabalho envolveu o estudo da tecnologia de comunicação de longo alcance e baixo consumo LoRa, seus parâmetros de configuração e a implementação de três sistemas de telemetria de um veículo de competição da modalidade SAE Baja, com quatro dispositivos diferentes: Esp32 DeviKit V1, Esp32 Heltec Wifi Lora 32 V1 e V3 e módulo LoRa E220-900T22D.

Foram realizados diferentes testes com cada um dos sistemas projetados visando medir o alcance e velocidade de transmissão da comunicação LoRa em cada um deles e, com base na análise dos resultados, foi selecionado o sistema utilizando os módulos LoRa E220-900T22D como o mais adequado para a telemetria do veículo da equipe Baja UFSCar, suportando mais de 550m de alcance, consumo em torno de 190mA e taxa de recepção de 0,5 mensagem por segundo. Embora a taxa de perda de pacotes foi relativamente alta para este sistema (superior a 50%), para a aplicação de telemetria do veículo Baja, este fator não apresenta tanta importância se comparado com a taxa de recepção de mensagens para o monitoramento.

Visto que o tempo de implementação durante a disciplina de TCC foi curto, foram estabelecidas algumas melhorias a se fazer no sistema selecionado para garantir um melhor funcionamento durante as competições futuras da equipe, como ajustar o intervalo entre medições dos sensores afim de diminuir o gargalo no dispositivo de transmissão, ajustar o parâmetro de tamanho do pacote de dados transmitido, que resulta em uma maior velocidade de transmissão de mensagens, e realizar medições de consumo de energia do sistema para analisar qual o consumo real de bateria do sistema a ser embarcado no veículo.

Durante os testes finais, foi possível observar algumas anomalias nos gráficos gerados a partir das medições recebidas. Isto pode ser relacionado ao mal contato dos componentes eletrônicos, principalmente nesta etapa em que foram utilizados protoboards e jumpers para testar os sistemas, realizando modificações constantes no arranjo e ligações dos componentes. O programa em Python utilizado para o armazenamento e display dos dados também pode ser um alvo de melhoria, realizando o tratando dados recebidos corrompidos e melhorando a interface para o usuário final.

Devido a necessidade de se enquadrar no calendário da disciplina de TCC - Trabalho de Conclusão de Curso de Engenharia de Computação, não foi possível investigar profundamente estas anomalias, assim como o problema da leitura dos sensores utilizando a placa Heltec Esp32 WiFi LoRa 32 V1 e V3 ao longo do trabalho. Também não foi possível realizar os testes no carro da equipe da universidade, visto que este também se encontrava em fases iniciais de desenvolvimento, e não estaria pronto para a implantação do sistema eletrônico.

Por fim, apesar dos desafios descritos anteriormente, este trabalho foi bem-sucedido em atingir seu objetivo de projetar e implementar um sistema de telemetria para veículos da modalidade Baja, se adequando aos desafios que a modalidade proporciona, como auseência de cobertura de redes celular, facilidade de implementação e baixo consumo de energia. Também cumprindo a proposta de desenvolvimento de um projeto de Engenharia de Computação. O projeto envolveu tanto a implementação de hardware quanto a de software, além de aplicar conceitos de telecomunicação, proporcionando uma solução eficiente e de baixo custo para monitoramento e comunicação dos sistemas do veículo.

Referências

ALMEIDA, A. et al. A comparative study between lora and zigbee transmission for racing car telemetry. In: . [S.l.: s.n.], 2023.

AUGUSTIN, A. et al. A study of lora: Long range low power networks for the internet of things. **Sensors**, v. 16, n. 9, 2016. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/16/9/1466>.

DEVALAL, S.; KARTHIKEYAN, A. Lora technology - an overview. In: **2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)**. [S.l.: s.n.], 2018. p. 284–290.

DIAS, R. N. **Solução de comunicação de baixo custo utilizando LoRa e ESP32**. Dissertação (Monografia (Graduação em Engenharia de Computação)) — Universidade Federal de São Carlos, São Carlos, Brasil, 2023.

GENYO. 2024. https://genyo.com.br/telemetria/. Acessado em: 12 fev. 2025.

GISI, A. et al. **A Review of Telemetric Subsystems Tracking Relevant Data in a Solar Powered Car**. International Foundation for Telemetering, 2021. Disponível em: <http://hdl.handle.net/10150/666303>.

GOMES, M. G. **Telemetria Veicular**. 26 p. Dissertação (Monografia (Graduação em Engenharia de Controle e Automação)) — Universidade Federal de Ouro Preto, Escola de Minas, Ouro Preto, Brasil, 2022.

HELERBROCK, R. **O que é efeito Doppler?** https://brasilescola.uol.com.br/o-que-e/fisica/o-que-e-efeito-doppler.htm. Acessado em: 05 mar. 2025.

IKPEHAI, A. et al. Low-power wide area network technologies for internet-of-things: A comparative review. **IEEE Internet of Things Journal**, v. 6, n. 2, p. 2225–2240, 2019.

NETO, A. S. **Implementação de telemetria no veículo da equipe cerrado Baja Sae**. Dissertação (Monografia (Graduação em Engenharia Elétrica)) — Universidade Federal de Uberlândia, Uberlândia, Brasil, 2021.

RENZONE, G. D. et al. Lorawan for vehicular networking: Field tests for vehicle-to-roadside communication. **Sensors**, v. 24, n. 6, 2024. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/24/6/1801>.

SAE-BRASIL. **BAJA Nacional**. https://saebrasil.org.br/programas-estudantis/baja-sae-brasil-2/. Acessado em: 01 fev. 2025.

SINHA, R. S.; WEI, Y.; HWANG, S.-H. A survey on lpwa technology: Lora and nb-iot. **ICT Express**, v. 3, n. 1, p. 14–21, 2017. ISSN 2405-9595. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405959517300061>.

SOUZA, F. **Sistemas Embarcados**. 2022. https://embarcados.com.br/o-que-sao-sistemas-embarcados/. Acessado em: 01 fev. 2025.

SUNDARAM, J. P. S.; DU, W.; ZHAO, Z. A survey on lora networking: Research problems, current solutions, and open issues. **IEEE Communications Surveys Tutorials**, v. 22, n. 1, p. 371–388, 2020.

TIER, E. G. **Desenvolvimento de sistemas de monitoramento empregando tecnologia LoRa em um protótipo tipo formula SAE**. Dissertação (Monografia (Graduação em Engenharia Elétrica)) — Universidade Federal de Controle e Automação, Santa Maria, Brasil, 2019.

Apêndices

```
/*
 *  C digo utilizado para enviar o valor de um contador por LoRa
 *  Autor: Jo o Paulo Sampaio G es Costa
 */

#include "heltec.h"

int cont = 0;
void setup() {
  Heltec.begin(true /*DisplayEnable Enable*/, false /*LoRa Disable*/,
      false /*Serial Enable*/);

  // Configura  o do LoRa
  LoRa.setSpreadingFactor(12);
  LoRa.begin(915.2E6, true); // Frequ ncia de 915MHz
  LoRa.setSignalBandwidth(125E3);
  LoRa.setTxPower(17,PA_OUTPUT_PA_BOOST_PIN);
  LoRa.setCodingRate4(8);

  Serial.begin(9600);
}

void loop() {

  LoRa.beginPacket();
  LoRa.print(cont);
  LoRa.endPacket();

  Heltec.display->drawString(64, 40, String(cont));
  Heltec.display->display();
  Serial.println(cont);

  cont++;

  delay(100);
}
```

Código 1 – Código para enviar mensagens LoRa com a placa Heltec Wifi Lora 32 V1

```
/*
 *  C digo utilizado para receber mensagens LoRa e imprimir elas no
    display e porta serial
 *  Autor: Jo o Paulo Sampaio G es Costa
 */

#include "heltec.h"

```

```
8  void setup() {
9    Heltec.begin(true /*DisplayEnable Enable*/, false /*LoRa Disable*/,
         true /*Serial Enable*/);
10   Heltec.display->init(); // Inicializa o display
11   Heltec.display->clear(); // Limpa o display
12   Heltec.display->setTextAlignment(TEXT_ALIGN_CENTER);
13   Heltec.display->setFont(ArialMT_Plain_10);
14   Heltec.display->drawString(64, 0, "Receiver");
15
16   // Configura  o do LoRa
17   LoRa.setSpreadingFactor(12);
18   LoRa.begin(915.2E6, true); // Frequ ncia de 915MHz
19   LoRa.setSignalBandwidth(125E3);
20   LoRa.setTxPower(17,PA_OUTPUT_PA_BOOST_PIN);
21   LoRa.setCodingRate4(8);
22   Serial.begin(115200);
23 }
24
25 void loop() {
26   //Heltec.display->clear();
27   Heltec.display->drawString(64, 0, "Receiver");
28   Heltec.display->display();
29   int packetSize = LoRa.parsePacket();
30   if (packetSize) { //verifica se recebeu mensagem
31     Heltec.display->clear(); //limpa a tela do oled
32     String receivedMessage = "";
33     while (LoRa.available()) { // salva a mensagem recebida
34       receivedMessage += (char)LoRa.read();
35     }
36     Heltec.display->drawString(64, 20, receivedMessage); //imprime a
           mensagem no oled
37     Heltec.display->display();
38     Serial.println(receivedMessage); //imprime a mensagem na porta
           serial
39   }
40
41   delay(500);
42 }
```

Código 2 – Código receber mensagens LoRa com a placa Heltec Wifi Lora 32 V1

```
1  /*
2   *  C digo para receber mensagens LoRa e retransmiti-las com outra sync
        word.
3   *  Autor: Jo o Paulo Sampaio G es Costa
4   */
5
6  #include <LoRa.h>
```

```
7
8  #define Frequency 917E6 // Hz
9  #define SignalBandwidth 125E3 // Hz
10 #define SpreadingFactor 10 // SF7..SF12
11 #define CodingRateDenominator 5
12 #define PreambleLength 8
13 #define SyncWord 0x12
14 #define SyncWordReceive 0x21
15 #define OutputPower 14 // dBm
16
17 void setup() {
18     Serial.begin(115200);
19     while (!Serial);
20     delay(500);
21
22     Serial.println("[setup] Configurando LoRa...");
23     LoRa.setPins(SS, RST_LoRa, DIO0);
24
25     if (!LoRa.begin(Frequency)) {
26         Serial.println("[setup] Erro ao iniciar LoRa!");
27         while (true);
28     }
29
30     LoRa.setSpreadingFactor(SpreadingFactor);
31     LoRa.setSignalBandwidth(SignalBandwidth);
32     LoRa.setPreambleLength(PreambleLength);
33     LoRa.setCodingRate4(CodingRateDenominator);
34     LoRa.setSyncWord(SyncWordReceive);
35     LoRa.disableCrc();
36
37     Serial.println("[setup] LoRa inicializado.");
38 }
39
40 void loop() {
41     int packetSize = LoRa.parsePacket();
42     if (packetSize) {
43         Serial.print("Pacote recebido: ");
44         String received = LoRa.readString();
45         Serial.println(received);
46
47         Serial.println("Alterando Sync Word...");
48         LoRa.setSyncWord(SyncWord);
49         delay(30);
50
51         Serial.println("Transmitindo mensagem...");
52         LoRa.beginPacket();
53         LoRa.print(received);
```

```
54          LoRa.endPacket();
55
56          Serial.println("Mensagem enviada. Restaurando Sync Word...");
57          delay(30);
58          LoRa.setSyncWord(SyncWordReceive);
59      }
60  }
```

Código 3 – Código Arduino para retransmissão LoRa

```
1  /*
2   *  C digo baseado no exemplo https://github.com/ropg/
       heltec_esp32_lora_v3/tree/main/examples/LoRa_rx_tx da biblioteca
       heltec_unnoficial.
3   *  utilizado para receber mensagens de comunica  o LoRa e imprimi-las
        na porta serial.
4   *  Autor: Jo o Paulo Sampaio G es Costa
5   */
6
7  #include <heltec_unofficial.h>
8
9  #define Frequency 917.0// Hz
10 #define SignalBandwidth 125.0// Hz
11 #define SignalBandwidthIndex 0// 0: 125 kHz,
12 // 1: 250 kHz,
13 // 2: 500 kHz,
14 // 3: Reserved
15 #define SpreadingFactor 10// SF7..SF12
16 #define CodingRate 1// 1: 4/5,
17 // 2: 4/6,
18 // 3: 4/7,
19 // 4: 4/8
20 #define CodingRateDenominator 5
21 #define PreambleLength 8// Same for Tx and Rx
22 #define SyncWord 0x12
23 #define OutputPower 14// dBm
24 #define SymbolTimeout 0// Symbols
25 #define FixedLengthPayload false
26 #define IQInversion false
27
28 String rxdata;
29 volatile bool rxFlag = false;
30 long cont = 0; //utilizado para armazenar o tempo em milissegundos em
       que se recebeu mensagens pela ultima vez.
31
32 void setup() {
33   // put your setup code here, to run once:
34   heltec_setup();
```

```
35    both.println("Radio init");
36    RADIOLIB_OR_HALT(radio.begin());
37    // Set the callback function for received packets
38    radio.setDio1Action(rx);
39    // Set radio parameters
40    both.printf("Frequency: %.2f MHz\n", Frequency);
41    RADIOLIB_OR_HALT(radio.setFrequency(Frequency));
42    both.printf("Bandwidth: %.1f kHz\n", SignalBandwidth);
43    RADIOLIB_OR_HALT(radio.setBandwidth(SignalBandwidth));
44    both.printf("Spreading Factor: %i\n", SpreadingFactor);
45    RADIOLIB_OR_HALT(radio.setSpreadingFactor(SpreadingFactor));
46    both.printf("TX power: %i dBm\n", OutputPower);
47    RADIOLIB_OR_HALT(radio.setOutputPower(OutputPower));
48    both.printf("preamble length: %i dBm\n", PreambleLength);
49    RADIOLIB_OR_HALT(radio.setPreambleLength(PreambleLength));
50    // Start receiving
51    RADIOLIB_OR_HALT(radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF));
52  }
53
54  void loop() {
55    // put your main code here, to run repeatedly:
56    heltec_loop(); //necessario para a biblioteca heltec_unnoficial
57    long int timer = millis();
58    if(timer - cont > 3000) { //se ficar mais de tres segundos sem receber
          mensagem, apaga o led
59      heltec_led(0);
60    }
61    if (rxFlag) { // mensagem recebida
62      rxFlag = false;
63      radio.readData(rxdata); //le a mensagem
64      if (_radiolib_status == RADIOLIB_ERR_NONE) { //caso nao haja nenhum
            erro na leitura, imprime o valor na porta serial
65        Serial.println(rxdata.c_str());
66        heltec_led(50); // acende o led
67        cont = millis(); // salva o tempo de recebimento da mensagem
68      }
69      RADIOLIB_OR_HALT(radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF))
            ;
70    }
71  }
72
73  // Can't do Serial or display things here, takes too much time for the
      interrupt
74  // fun   o chamada pela biblioteca quando uma mensagem LoRa    recebida
75  void rx() {
76    rxFlag = true;
77  }
```

Código 4 – Código para receber mensagens LoRa na placa Heltec Wifi Lora 32 V3,

```
/*
 *  C digo utilizado para envio do valor de um contador por mensagens
    de comunica  o LoRa com o m dulo E220
 *  Autor: Jo o Paulo Sampaio G es Costa
 */

#define FREQUENCY_915
#include "LoRa_E220.h"
int cont = 0;
LoRa_E220 e220ttl(&Serial2, 18, 21, 19); //  RX AUX M0 M1; // WeMos RX
    --> e220 TX - WeMos TX --> e220 RX

void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, 16, 17); // necessario para a
      comunicaca com o m dulo
  delay(500);

  // Startup all pins and UART
  e220ttl.begin();

}

void loop() {
  cont++;
  ResponseStatus rs  = e220ttl.sendMessage(String(cont)); //envia o
      valor do contador
  Serial.println("sending:" + String(cont));
  Serial.println(rs.getResponseDescription()); //retorna se obteve
      sucesso no envio da mensagem
  delay(100);
}
```

Código 5 – Código para envio de mensagens LoRa utilizando o módulo LoRa E220-900T220D

```
/*
 *  C digo utilizado para receber por mensagens de comunica  o LoRa
    com o m dulo E220
 *  Autor: Jo o Paulo Sampaio G es Costa
 */

#include "LoRa_E220.h"

```

```
8  #define FREQUENCY_915
9  LoRa_E220 e220ttl(&Serial2, 18, 21, 19); //  RX AUX M0 M1
10
11 #define LED_PIN 2
12
13 long int cont = 0; // armazena o tempo em que foi recebida a ultima
       mensagem
14
15 void setup() {
16   Serial.begin(9600);
17   Serial2.begin(9600, SERIAL_8N1, 16, 17);
18   delay(500);
19
20   // Startup all pins and UART
21   e220ttl.begin();
22
23   Serial.println("Start receiving!");
24
25   pinMode(LED_PIN, OUTPUT);
26   digitalWrite(LED_PIN, LOW);
27 }
28
29 void loop() {
30   long int timer = millis();
31   if(timer - cont > 3000) { // se nenhuma mensagem chegou nos ultimos 3
       segundos, apaga o LED.
32     digitalWrite(LED_PIN, LOW);
33   }
34   // If something available
35   if (e220ttl.available()>1) { // Se recebeu alguma mensagem
36       // read the String message
37     ResponseContainer rc = e220ttl.receiveMessage(); //le a mensagem
38
39     if (rc.status.code!=1){ // caso de um erro, imprime ele
40         Serial.println(rc.status.getResponseDescription());
41     }else{ // caso contrario, imprime a mensagem na porta serial
42           String mensagem = rc.data;
43           cont = millis();
44           digitalWrite(LED_PIN, HIGH);
45           Serial.println(mensagem);
46        }
47     }
48   }
49 }
```

Código 6 – Código para receber mensagens LoRa com o módulo LoRa E220-900T220D,

```
1  sor/*
```

```
2  *  C digo utilizado para ler o sensor de temperatura e enviar a medida
       pela porta serial
3  *  Autor: Jo o Paulo Sampaio G es Costa
4  */
5
6  #include <Adafruit_Sensor.h>
7  #include <Wire.h>
8  #include "max6675.h"
9
10 //Inicializa  o para o sensor termopar
11 int thermoSO = 4;
12 int thermoCS = 2;
13 int thermoCLK = 15;
14 MAX6675 thermocouple(thermoCLK, thermoCS, thermoSO);
15
16 void setup(void) {
17   Serial.begin(9600);
18 }
19
20 void loop() {
21
22   float tempmax = thermocouple.readCelsius();
23   Serial.println(tempmax);
24   delay(100);
25 }
```

Código 7 – Código para a leitura do sensor de temperatura e envio na prota serial

```
1  /*
2  *  C digo utilizado para receber medi oes da porta serial e envia-las
       por comunicacao LoRa.
3  *  Autor: Jo o Paulo Sampaio G es Costa
4  */
5
6
7  #include <LoRa.h>
8  #include <HardwareSerial.h>
9
10 #define Frequency 917E6// Hz
11 #define SignalBandwidth 125E3// Hz
12 #define SignalBandwidthIndex 0// 0: 125 kHz,
13 // 1: 250 kHz,
14 // 2: 500 kHz,
15 // 3: Reserved
16 #define SpreadingFactor 10// SF7..SF12
17 #define CodingRate 1// 1: 4/5,
18 // 2: 4/6,
19 // 3: 4/7,
```

```
20 // 4: 4/8
21 #define CodingRateDenominator 5
22 #define PreambleLength 8// Same for Tx and Rx
23 #define SyncWord 0x21
24 //#define SyncWordWait 0x12
25 #define OutputPower 14// dBm
26 #define SymbolTimeout 0// Symbols
27 #define FixedLengthPayload false
28 #define IQInversion false
29
30 void setup() {
31   Serial.begin(115200);
32   while (!Serial);        // If just the the basic function, must
       connect to a computer
33     delay(500);
34
35   LoRa.setPins(SS,RST_LoRa,DIO0);
36   Serial.println("[setup] Activate LoRa Sender");
37   if (!LoRa.begin(Frequency)) {
38     Serial.println("[setup] Starting LoRa failed!");
39     while (true);
40   }
41
42   Serial.print("[setup] LoRa Frequency: ");
43   Serial.println(Frequency);
44
45   LoRa.setSpreadingFactor(SpreadingFactor);
46   LoRa.setSignalBandwidth(SignalBandwidth);
47   LoRa.setPreambleLength(PreambleLength);
48   LoRa.setCodingRate4(CodingRateDenominator);
49   LoRa.setSyncWord(SyncWord);
50   LoRa.disableCrc();
51
52   Serial.println("[setup] LoRa initialisation complete");
53   Serial.println();
54
55   Serial2.begin(9600, SERIAL_8N1, 16, 17); // Configura UART1 (TX=17, RX
       =16) para receber medicoes serial
56 }
57
58 void loop() {
59     if (Serial2.available()) {       // Verifica se h  dados
         dispon veis
60     String data = Serial2.readStringUntil('\n'); // L  os dados
         recebidos
61     Serial.println("Received: " + data); // Exibe no monitor serial
62
```

```
63      // envia medicao por LoRa no formato esperado
64      Serial.println("Enviando mensagem...");
65      LoRa.beginPacket();
66      LoRa.print("pass,t," + data);
67      LoRa.endPacket();
68      delay(100);
69    }
70
71  }
```

Código 8 – Código utilizado para ler as medidas do sensor na porta serial e envia-los por LoRa

```
1   /*
2    *  C digo utilizado para receber mensagens LoRa, verificar se a
         mensagem cont m a senha definida e retransmitir a mensagem com
         outra syncword.
3    *  Autor: Jo o Paulo Sampaio G es Costa
4    */
5
6   #include <LoRa.h>
7
8   #define Frequency 917E6// Hz
9   #define SignalBandwidth 125E3// Hz
10  #define SignalBandwidthIndex 0// 0: 125 kHz,
11  // 1: 250 kHz,
12  // 2: 500 kHz,
13  // 3: Reserved
14  #define SpreadingFactor 10// SF7..SF12
15  #define CodingRate 1// 1: 4/5,
16  // 2: 4/6,
17  // 3: 4/7,
18  // 4: 4/8
19  #define CodingRateDenominator 5
20  #define PreambleLength 8// Same for Tx and Rx
21  #define SyncWord 0x12
22  #define SyncWordReceive 0x21
23  #define OutputPower 14// dBm
24  #define SymbolTimeout 0// Symbols
25  #define FixedLengthPayload false
26  #define IQInversion false
27
28  int sf = 0;
29
30  void setup() {
31    Serial.begin(115200);
32    while (!Serial);        // If just the the basic function, must
        connect to a computer
```

```
33      delay(500);
34    Serial.print("[setup] ");
35
36    LoRa.setPins(SS,RST_LoRa,DIO0);
37    if (!LoRa.begin(Frequency)) {
38      Serial.println("[setup] Starting LoRa failed!");
39      while (true);
40    }
41
42    Serial.print("[setup] LoRa Frequency: ");
43    Serial.println(Frequency);
44
45    LoRa.setSpreadingFactor(SpreadingFactor);
46    LoRa.setSignalBandwidth(SignalBandwidth);
47    LoRa.setPreambleLength(PreambleLength);
48    LoRa.setCodingRate4(CodingRateDenominator);
49    LoRa.setSyncWord(SyncWordReceive);
50    LoRa.disableCrc();
51
52    Serial.println("[setup] LoRa initialisation complete");
53    Serial.println();
54  }
55
56  void loop() {
57    // verifica se recebeu mensagem.
58    int packetSize = LoRa.parsePacket();
59    if (packetSize) {
60
61      Serial.print("Recebido pacote: ");
62      while (LoRa.available()) {
63        String received = LoRa.readString(); //armazena a mensagem
              recebida
64        Serial.println(received);
65        int posicao = received.indexOf("pass");
66        if (posicao != -1) {  // Se a mensagem tiver a senha determinada
67          delay(30);
68
69          Serial.println("trocando para sync word: "); //troca para a
              syncword de envio
70          long int timer = millis();
71          LoRa.setSyncWord(SyncWord);
72          Serial.println("syncword trocada em: " + String(millis() - timer
              ));
73
74          Serial.println("enviando msg no lora");
75          timer = millis();
76          LoRa.beginPacket();
```

```
77          LoRa.print(received);
78          LoRa.endPacket(); // Finalizar pacote
79          Serial.println("enviado: " + received + " em: " + String(millis
                () - timer));
80
81          Serial.println("trocando para receive sync word: ");
82          timer = millis();
83          delay(30);
84          LoRa.setSyncWord(SyncWordReceive); // retorna a syncword para o
                valor de recebimento de mensagens
85          Serial.println("syncword trocada em: " + String(millis() - timer
                ));
86          delay(30);
87        }
88
89      }
90        LoRa.setSyncWord(SyncWordReceive);
91        delay(50);
92    }
93  }
```

Código 9 – Código utilizado para retransmitir mensagens verificando a palavra chave nos testes finais com intermediário

```
1  /*
2   *  C digo baseado no exemplo https://github.com/ropg/
       heltec_esp32_lora_v3/tree/main/examples/LoRa_rx_tx da biblioteca
       heltec_unnoficial.
3   *  C digo utilizado para receber mensagens LoRa com a placa ESP32
       Heltec WIFI LoRa 32 V3, verificar se a mensagem cont m a senha
       definida e imprimir elas no display e porta serial.
4   *  Adicionalmente, o programa tamb m l  a porta serial. Caso um
       n mero seja inserido, ele altera o SpreadingFactor para este valor.
5   *  Autor: Jo o Paulo Sampaio G es Costa
6   */
7
8  #include <heltec_unofficial.h>
9
10 #define Frequency 917.0// Hz
11 #define SignalBandwidth 125.0// Hz
12 #define SignalBandwidthIndex 0// 0: 125 kHz,
13 // 1: 250 kHz,
14 // 2: 500 kHz,
15 // 3: Reserved
16 #define SpreadingFactor 10// SF7..SF12
17 #define CodingRate 1// 1: 4/5,
18 // 2: 4/6,
19 // 3: 4/7,
```

```
// 4: 4/8
#define CodingRateDenominator 5
#define PreambleLength 8// Same for Tx and Rx
#define SyncWord 0x12
#define OutputPower 14// dBm
#define SymbolTimeout 0// Symbols
#define FixedLengthPayload false
#define IQInversion false

int sf = 0;

String rxdata;
volatile bool rxFlag = false;
long int cont = 0; //utilizado para armazenar o tempo em milissegundos
    em que se recebeu mensagens pela ultima vez.

// Can't do Serial or display things here, takes too much time for the
    interrupt
void rx() {
  rxFlag = true;
}

void setup() {
  // put your setup code here, to run once:
  heltec_setup();
  both.println("Radio init");
  RADIOLIB_OR_HALT(radio.begin());
  // Set the callback function for received packets
  radio.setDio1Action(rx);
  // Set radio parameters
  both.printf("Frequency: %.2f MHz\n", Frequency);
  RADIOLIB_OR_HALT(radio.setFrequency(Frequency));
  both.printf("Bandwidth: %.1f kHz\n", SignalBandwidth);
  RADIOLIB_OR_HALT(radio.setBandwidth(SignalBandwidth));
  both.printf("Spreading Factor: %i\n", SpreadingFactor);
  RADIOLIB_OR_HALT(radio.setSpreadingFactor(SpreadingFactor));
  both.printf("TX power: %i dBm\n", OutputPower);
  RADIOLIB_OR_HALT(radio.setOutputPower(OutputPower));
  both.printf("preamble length: %i dBm\n", PreambleLength);
  RADIOLIB_OR_HALT(radio.setPreambleLength(PreambleLength));
  // Start receiving
  RADIOLIB_OR_HALT(radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF));
  heltec_led(0);
}

void loop() {
  // put your main code here, to run repeatedly:
```

```
65    heltec_loop(); //necessario para a biblioteca heltec_unnoficial
66
67    if (Serial.available()) { // trecho respons vel por ler a porta
          serial e configurar o spread factor dinamicamente
68      String sfstring = Serial.readString();  // L   a string enviada pelo
            monitor serial
69      Serial.print("Mensagem recebida: ");
70      sf = sfstring.toInt();
71      Serial.println(sf);
72      RADIOLIB_OR_HALT(radio.setSpreadingFactor(sf)); // configura o
            spread factor
73      RADIOLIB_OR_HALT(radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF))
            ;
74    }
75
76    long int timer = millis();
77    if(timer - cont > 3000) { //se ficar mais de tres segundos sem receber
          mensagem, apaga o led
78      heltec_led(0);
79    }
80
81    if (rxFlag) { // mensagem recebida
82      rxFlag = false;
83      radio.readData(rxdata); //le a mensagem
84      if (_radiolib_status == RADIOLIB_ERR_NONE) {
85        String mensagem = rxdata.c_str(); //le a mensagem
86        int posicao = mensagem.indexOf("pass");
87        if (posicao != -1) {  // Se a mensagem contem a senha definida
88          Serial.println(mensagem); //caso nao haja nenhum erro na leitura
                , imprime o valor na porta serial
89          heltec_led(50); // acende o led
90          cont = millis();  // salva o tempo de recebimento da mensagem
91          Serial.println("xcs,i," + String(radio.getRSSI())); //imprime o
                RSSI da mensagem
92        }
93      }
94      RADIOLIB_OR_HALT(radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF))
            ;
95    }
96  }
```

Código 10 – Código para ler mensagens LoRa na placa Heltec Wifi Lora 32 V3 verificando a palavra chave e imprimindo o RSSI para os testes finais com LoRa

```
1   /*
2    *  C digo utilizado para realizar a leitura do sensor de temperatura
       max6675 e enviar por mensagens de comunica  o LoRa com o m dulo
       E220
```

```
 *  Autor: Jo o Paulo Sampaio G es Costa
 */

#include <Wire.h>
#include "max6675.h"

#define FREQUENCY_915
#include "LoRa_E220.h"
LoRa_E220 e220ttl(&Serial2, 18, 21, 19); //  RX AUX M0 M1; // WeMos RX
    --> e220 TX - WeMos TX --> e220 RX


int thermoSO = 4;
int thermoCS = 2;
int thermoCLK = 15;
MAX6675 thermocouple(thermoCLK, thermoCS, thermoSO);

void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, 16, 17);
  delay(500);

  // Startup all pins and UART
  e220ttl.begin();

}

void loop() {
  float tempmax = thermocouple.readCelsius();
  ResponseStatus rs  = e220ttl.sendMessage("pass,r," + String(tempmax));
      //envia mensagem no formato
  Serial.println("sending: pass,r," + String(tempmax));
  Serial.println(rs.getResponseDescription());
  delay(100);
}
```

Código 11 – Código utilizado para ler o sensor de temperatura e enviar a medição por LoRa com o módulo LoRa E220-900T22D

Código 12 – Código Arduino para retransmissão LoRa

```
/*
 *  C digo utilizado para receber as mensagens LoRa com o m dulo E220,
     verificando se elas cont m a palavra chave correta e imprimindo
    tamb m o RSSI da mensagem.
 *  Autor: Jo o Paulo Sampaio G es Costa
 */
```

```
5
6  #include "LoRa_E220.h"
7  #define FREQUENCY_915
8  LoRa_E220 e220ttl(&Serial2, 18, 21, 19); //  RX AUX M0 M1
9
10 #define LED_PIN 2
11 #define ENABLE_RSSI true
12
13 long int cont = 0; // armazena o tempo em que foi recebida a ultima
       mensagem
14
15 void setup() {
16   Serial.begin(115200);
17   Serial2.begin(9600, SERIAL_8N1, 16, 17);
18   delay(500);
19
20   // Startup all pins and UART
21   e220ttl.begin();
22
23   Serial.println("Start receiving!");
24
25   pinMode(LED_PIN, OUTPUT);
26   digitalWrite(LED_PIN, LOW);
27 }
28
29 void loop() {
30   long int timer = millis();
31   if(timer - cont > 3000) { // se nenhuma mensagem chegou nos ultimos 3
         segundos, apaga o LED.
32     digitalWrite(LED_PIN, LOW);
33   }
34
35   if (e220ttl.available()>1) { // Se recebeu alguma mensagem
36
37     #ifdef ENABLE_RSSI //se deseja ler o RSSI, utiliza este m todo para
           ler a mensagem
38     ResponseContainer rc = e220ttl.receiveMessageRSSI();
39     #else
40     ResponseContainer rc = e220ttl.receiveMessage();
41     #endif
42
43     if (rc.status.code!=1){ //caso de um erro, imprime ele
44         Serial.println(rc.status.getResponseDescription());
45     }else{ // caso contrario, imprime a mensagem na porta serial
46         String mensagem = rc.data;
47         int posicao = mensagem.indexOf("pass"); //verifica se a mensagem
               recebida cont m a senha esperada
```

```
48          if (posicao != -1) {
49            // imprime a mensagem na porta serial, assim como o RSSI
50            cont = millis();
51            digitalWrite(LED_PIN, HIGH);
52            Serial.println(mensagem);
53            Serial.println("pass,i," + String(rc.rssi));
54          }
55      }
56    }
57  }
```

Código 13 – Código para a leitura de mensagens LoRa com o módulo LoRa E220-900T22D, verificando a palavra chave e imprimindo o RSSI

```
1  /*
2   * LoRa E220
3   * Get configuration.
4   * C digo baseado no exemplo de:
5   * by Renzo Mischianti <https://www.mischianti.org>
6   *
7   * https://www.mischianti.org
8   *
9   */
10
11 #include "LoRa_E220.h"
12
13
14 LoRa_E220 e220ttl(16, 17, &Serial2, 18, 21, 19, UART_BPS_RATE_9600); //
       esp32 RX <-- e220 TX, esp32 TX --> e220 RX AUX M0 M1
15
16 void printParameters(struct Configuration configuration);
17 void printModuleInformation(struct ModuleInformation moduleInformation);
18
19 void setup() {
20   Serial.begin(9600);
21   //Serial2.begin(9600, SERIAL_8N1, 16, 17);
22   while(!Serial){};
23   delay(500);
24
25   Serial.println();
26
27
28   // Startup all pins and UART
29   e220ttl.begin();
30
31   ResponseStructContainer c;
32   c = e220ttl.getConfiguration();
33   // It's important get configuration pointer before all other operation
```

```
34    Configuration configuration = *(Configuration*) c.data;
35    Serial.println(c.status.getResponseDescription());
36    Serial.println(c.status.code);
37
38    printParameters(configuration);
39
40    ResponseStructContainer cMi;
41    cMi = e220ttl.getModuleInformation();
42    // It's important get information pointer before all other operation
43    ModuleInformation mi = *(ModuleInformation*)cMi.data;
44
45    Serial.println(cMi.status.getResponseDescription());
46    Serial.println(cMi.status.code);
47
48    printModuleInformation(mi);
49
50    c.close();
51    cMi.close();
52  }
53
54  void loop() {
55
56  }
57  void printParameters(struct Configuration configuration) {
58    Serial.println("----------------------------------------");
59
60    Serial.print(F("HEAD : "));  Serial.print(configuration.COMMAND, HEX);
        Serial.print(" ");Serial.print(configuration.STARTING_ADDRESS, HEX)
        ;Serial.print(" ");Serial.println(configuration.LENGHT, HEX);
61    Serial.println(F(" "));
62    Serial.print(F("AddH : "));  Serial.println(configuration.ADDH, HEX);
63    Serial.print(F("AddL : "));  Serial.println(configuration.ADDL, HEX);
64    Serial.println(F(" "));
65    Serial.print(F("Chan : "));  Serial.print(configuration.CHAN, DEC);
        Serial.print(" -> "); Serial.println(configuration.
        getChannelDescription());
66    Serial.println(F(" "));
67    Serial.print(F("SpeedParityBit      : "));  Serial.print(configuration.
        SPED.uartParity, BIN);Serial.print(" -> "); Serial.println(
        configuration.SPED.getUARTParityDescription());
68    Serial.print(F("SpeedUARTDatte      : "));  Serial.print(configuration.
        SPED.uartBaudRate, BIN);Serial.print(" -> "); Serial.println(
        configuration.SPED.getUARTBaudRateDescription());
69    Serial.print(F("SpeedAirDataRate    : "));  Serial.print(configuration.
        SPED.airDataRate, BIN);Serial.print(" -> "); Serial.println(
        configuration.SPED.getAirDataRateDescription());
70    Serial.println(F(" "));
```

```
71    Serial.print(F("OptionSubPacketSett:␣"));  Serial.print(configuration.
          OPTION.subPacketSetting, BIN);Serial.print("␣->␣"); Serial.println(
          configuration.OPTION.getSubPacketSetting());
72    Serial.print(F("OptionTranPower␣␣␣␣:␣"));  Serial.print(configuration.
          OPTION.transmissionPower, BIN);Serial.print("␣->␣"); Serial.println
          (configuration.OPTION.getTransmissionPowerDescription());
73    Serial.print(F("OptionRSSIAmbientNo:␣"));  Serial.print(configuration.
          OPTION.RSSIAmbientNoise, BIN);Serial.print("␣->␣"); Serial.println(
          configuration.OPTION.getRSSIAmbientNoiseEnable());
74    Serial.println(F("␣"));
75    Serial.print(F("TransModeWORPeriod␣:␣"));  Serial.print(configuration.
          TRANSMISSION_MODE.WORPeriod, BIN);Serial.print("␣->␣"); Serial.
          println(configuration.TRANSMISSION_MODE.
          getWORPeriodByParamsDescription());
76    Serial.print(F("TransModeEnableLBT␣:␣"));  Serial.print(configuration.
          TRANSMISSION_MODE.enableLBT, BIN);Serial.print("␣->␣"); Serial.
          println(configuration.TRANSMISSION_MODE.getLBTEnableByteDescription
          ());
77    Serial.print(F("TransModeEnableRSSI:␣"));  Serial.print(configuration.
          TRANSMISSION_MODE.enableRSSI, BIN);Serial.print("␣->␣"); Serial.
          println(configuration.TRANSMISSION_MODE.
          getRSSIEnableByteDescription());
78    Serial.print(F("TransModeFixedTrans:␣"));  Serial.print(configuration.
          TRANSMISSION_MODE.fixedTransmission, BIN);Serial.print("␣->␣");
          Serial.println(configuration.TRANSMISSION_MODE.
          getFixedTransmissionDescription());
79
80
81    Serial.println("----------------------------------------");
82  }
83  void printModuleInformation(struct ModuleInformation moduleInformation)
        {
84    Serial.println("----------------------------------------");
85    Serial.print(F("HEAD:␣"));  Serial.print(moduleInformation.COMMAND,
          HEX);Serial.print("␣");Serial.print(moduleInformation.
          STARTING_ADDRESS, HEX);Serial.print("␣");Serial.println(
          moduleInformation.LENGHT, DEC);
86
87    Serial.print(F("Model␣no.:␣"));  Serial.println(moduleInformation.
          model, HEX);
88    Serial.print(F("Version␣␣:␣"));  Serial.println(moduleInformation.
          version, HEX);
89    Serial.print(F("Features␣:␣"));  Serial.println(moduleInformation.
          features, HEX);
90    Serial.println("----------------------------------------");
91  }
```

Código 14 – Código utilizado para ler as configurações do módulo LoRa E220-900T22D

```
/*
 * LoRa E220
 * Set configuration.
 * C digo baseado no exemplo de:
 * by Renzo Mischianti <https://www.mischianti.org>
 *
 * https://www.mischianti.org
 *
 */


#define FREQUENCY_915
#define LoRa_E220_DEBUG  // for printing the settings
#include "LoRa_E220.h"

LoRa_E220 e220ttl(16, 17, &Serial2, 18, 21, 19, UART_BPS_RATE_9600); //
    esp32 RX <-- e220 TX, esp32 TX --> e220 RX AUX M0 M1

void printParameters(struct Configuration configuration);
void printModuleInformation(struct ModuleInformation moduleInformation);

void setup() {
  Serial.begin(9600);
  //Serial2.begin(9600, SERIAL_8N1, 16, 17);
  while(!Serial){};
  delay(500);

  Serial.println();


  // Startup all pins and UART
  e220ttl.begin();

  ResponseStructContainer c;
  c = e220ttl.getConfiguration();
  // It's important get configuration pointer before all other operation
  Configuration configuration = *(Configuration*) c.data;
  Serial.println(c.status.getResponseDescription());
  Serial.println(c.status.code);

  printParameters(configuration);

//  ----------------------- DEFAULT TRANSPARENT -----------------------
  configuration.ADDL = 0x02;  // Low byte of address
```

```
44    configuration.ADDH = 0x00; // High byte of address
45
46    configuration.CHAN = 23; // 868 MHz for Exxx-900 modules, choose 23
          for Exxx-400 to set 433 MHz
47
48    configuration.SPED.uartBaudRate = UART_BPS_9600; // Serial baud rate
49    //configuration.SPED.airDataRate = AIR_DATA_RATE_010_24; // Air baud
          rate
50    configuration.SPED.airDataRate = AIR_DATA_RATE_111_625;
51    configuration.SPED.uartParity = MODE_00_8N1; // Parity bit
52
53    configuration.OPTION.subPacketSetting = SPS_200_00; // Packet size
54    configuration.OPTION.RSSIAmbientNoise = RSSI_AMBIENT_NOISE_DISABLED;
          // Need to send special command
55    configuration.OPTION.transmissionPower = POWER_22; // Device power
56
57    configuration.TRANSMISSION_MODE.enableRSSI = RSSI_DISABLED; // Enable
          RSSI info
58    configuration.TRANSMISSION_MODE.fixedTransmission =
          FT_TRANSPARENT_TRANSMISSION; // Transmission mode
59    configuration.TRANSMISSION_MODE.enableLBT = LBT_DISABLED; // Check
          interference
60    configuration.TRANSMISSION_MODE.WORPeriod = WOR_2000_011; // WOR
          timing
61
62    configuration.CRYPT.CRYPT_H = 0x00;  // encryption high byte, default:
           0x00
63    configuration.CRYPT.CRYPT_L = 0x00;  // encryption low byte, default:
          0x00
64
65    /* Set configuration changed and set to hold the configuration; chose
66  WRITE_CFG_PWR_DWN_LOSE to not save the configuration permanently */
67    ResponseStatus rs = e220ttl.setConfiguration(configuration,
          WRITE_CFG_PWR_DWN_SAVE);
68    Serial.println(rs.getResponseDescription());
69    Serial.println(rs.code);
70
71    c = e220ttl.getConfiguration();
72    // It's important get configuration pointer before all other operation
73    configuration = *(Configuration*) c.data;
74    Serial.println(c.status.getResponseDescription());
75    Serial.println(c.status.code);
76
77    printParameters(configuration);
78    c.close();
79  }
80
```

```
81 void loop() {
82
83 }
84 void printParameters(struct Configuration configuration) {
85   DEBUG_PRINTLN("----------------------------------------");
86
87   DEBUG_PRINT(F("HEAD : "));  DEBUG_PRINT(configuration.COMMAND, HEX);
       DEBUG_PRINT(" ");DEBUG_PRINT(configuration.STARTING_ADDRESS, HEX);
       DEBUG_PRINT(" ");DEBUG_PRINTLN(configuration.LENGHT, HEX);
88   DEBUG_PRINTLN(F(" "));
89   DEBUG_PRINT(F("AddH : "));  DEBUG_PRINTLN(configuration.ADDH, HEX);
90   DEBUG_PRINT(F("AddL : "));  DEBUG_PRINTLN(configuration.ADDL, HEX);
91   DEBUG_PRINTLN(F(" "));
92   DEBUG_PRINT(F("Chan : "));  DEBUG_PRINT(configuration.CHAN, DEC);
       DEBUG_PRINT(" -> "); DEBUG_PRINTLN(configuration.
       getChannelDescription());
93   DEBUG_PRINTLN(F(" "));
94   DEBUG_PRINT(F("SpeedParityBit     : "));  DEBUG_PRINT(configuration.
       SPED.uartParity, BIN);DEBUG_PRINT(" -> "); DEBUG_PRINTLN(
       configuration.SPED.getUARTParityDescription());
95   DEBUG_PRINT(F("SpeedUARTDatte     : "));  DEBUG_PRINT(configuration.
       SPED.uartBaudRate, BIN);DEBUG_PRINT(" -> "); DEBUG_PRINTLN(
       configuration.SPED.getUARTBaudRateDescription());
96   DEBUG_PRINT(F("SpeedAirDataRate    : "));  DEBUG_PRINT(configuration.
       SPED.airDataRate, BIN);DEBUG_PRINT(" -> "); DEBUG_PRINTLN(
       configuration.SPED.getAirDataRateDescription());
97   DEBUG_PRINTLN(F(" "));
98   DEBUG_PRINT(F("OptionSubPacketSett: "));  DEBUG_PRINT(configuration.
       OPTION.subPacketSetting, BIN);DEBUG_PRINT(" -> "); DEBUG_PRINTLN(
       configuration.OPTION.getSubPacketSetting());
99   DEBUG_PRINT(F("OptionTranPower    : "));  DEBUG_PRINT(configuration.
       OPTION.transmissionPower, BIN);DEBUG_PRINT(" -> "); DEBUG_PRINTLN(
       configuration.OPTION.getTransmissionPowerDescription());
100  DEBUG_PRINT(F("OptionRSSIAmbientNo: "));  DEBUG_PRINT(configuration.
       OPTION.RSSIAmbientNoise, BIN);DEBUG_PRINT(" -> "); DEBUG_PRINTLN(
       configuration.OPTION.getRSSIAmbientNoiseEnable());
101  DEBUG_PRINTLN(F(" "));
102  DEBUG_PRINT(F("TransModeWORPeriod : "));  DEBUG_PRINT(configuration.
       TRANSMISSION_MODE.WORPeriod, BIN);DEBUG_PRINT(" -> ");
       DEBUG_PRINTLN(configuration.TRANSMISSION_MODE.
       getWORPeriodByParamsDescription());
103  DEBUG_PRINT(F("TransModeEnableLBT : "));  DEBUG_PRINT(configuration.
       TRANSMISSION_MODE.enableLBT, BIN);DEBUG_PRINT(" -> ");
       DEBUG_PRINTLN(configuration.TRANSMISSION_MODE.
       getLBTEnableByteDescription());
104  DEBUG_PRINT(F("TransModeEnableRSSI: "));  DEBUG_PRINT(configuration.
       TRANSMISSION_MODE.enableRSSI, BIN);DEBUG_PRINT(" -> ");
```

```
      DEBUG_PRINTLN(configuration.TRANSMISSION_MODE.
      getRSSIEnableByteDescription());
105   DEBUG_PRINT(F("TransModeFixedTrans:␣")); DEBUG_PRINT(configuration.
      TRANSMISSION_MODE.fixedTransmission, BIN);DEBUG_PRINT("␣->␣");
      DEBUG_PRINTLN(configuration.TRANSMISSION_MODE.
      getFixedTransmissionDescription());
106
107
108   DEBUG_PRINTLN("----------------------------------------");
109 }
110 void printModuleInformation(struct ModuleInformation moduleInformation)
      {
111   Serial.println("----------------------------------------");
112   DEBUG_PRINT(F("HEAD:␣")); DEBUG_PRINT(moduleInformation.COMMAND, HEX)
      ;DEBUG_PRINT("␣");DEBUG_PRINT(moduleInformation.STARTING_ADDRESS,
      HEX);DEBUG_PRINT("␣");DEBUG_PRINTLN(moduleInformation.LENGHT, DEC);
113
114   Serial.print(F("Model␣no.:␣")); Serial.println(moduleInformation.
      model, HEX);
115   Serial.print(F("Version␣␣:␣")); Serial.println(moduleInformation.
      version, HEX);
116   Serial.print(F("Features␣:␣")); Serial.println(moduleInformation.
      features, HEX);
117   Serial.println("----------------------------------------");
118 }
```

Código 15 – Código utilizado para configurar o módulo LoRa E220-900T22D