Politécnico de Viseu
Tecnologia e Gestão Lamego

Trabalho prático Sistemas de Telecomunicações

Trabalho de avaliação à disciplina de Sistemas de Telecomunicações

Politécnico de Viseu
Tecnologia e Gestão Lamego

LAMEGO 2024

Projeto Lora Emergency Message Finder

Trabalho de avaliação à disciplina de Sistemas de Telecomunicações

Licenciatura: Engenharia Informática e Telecomunicações

Cadeira: Sistemas de Telecomunicações

Orientador: Prof. Jorge Duarte

Autor: Jorge Pinto nº26171

Introdução

Nos últimos anos, a busca por soluções eficientes para localizar pessoas em áreas remotas ou com pouca cobertura de rede móvel tem se intensificado. Este projeto propõe o desenvolvimento de um sistema de localização baseado em dispositivos LoRa, utilizando módulos de transmissão (TX) e receção (RX) para estabelecer uma comunicação robusta e de longo alcance. A solução é voltada para operações de busca e salvamento, como em desastres naturais, áreas florestais, ou locais onde a infraestrutura de comunicação convencional é limitada ou inexistente.

O sistema utiliza o RSSI (Received Signal Strength Indicator) como método primário para estimar a localização do transmissor, permitindo determinar a proximidade em relação ao recetor. Caso o transmissor esteja equipado com um módulo GPS, suas coordenadas podem ser transmitidas diretamente, oferecendo maior precisão e eficiência. Para maximizar o alcance e a cobertura em terrenos de difícil acesso, o recetor será instalado num drone, possibilitando a realização de buscas aéreas com maior agilidade e amplitude.

Este projeto combina a tecnologia LoRa, conhecida por seu baixo consumo de energia e capacidade de comunicação de longa distância, com recursos avançados de mobilidade aérea. O resultado é uma ferramenta acessível e eficiente para auxiliar em missões críticas de busca e salvamento, potencializando a capacidade de localizar indivíduos em situações de risco de maneira mais rápida e precisa.

Politécnico de Viseu
Tecnologia e Gestão Lamego

Índice

Politécnico de Viseu
Tecnologia e Gestão Lamego

Tema do projeto

O tema deste trabalho foi pensado com base de ser uma possível solução para encontrar pessoas perdidas em ambientes com fraca cobertura de sinal telefónico.

Uma das possíveis soluções da sua utilização poderá passar por grupos de desportistas de neve em que mesmo com ambientes controlados avalanches podem ocorrer e podem facilmente ficar subterrados 10 metros, mesmo as autoridades locais terem uma ampla vasta de gama para auxílio imediato como helicópteros, motas de neve, etc. Se o grupo contar com um sistema embarcado dedicado conseguimos adiantar muito a operação de buscas e diminuindo significativamente o seu raio.

Para a utilização deste projeto neste ambiente é necessário que cada utilizador possua um rádio TX (transmissor) em que podemos contar com duas versões:

- Versão 1: o transmissor irá ser equipado apenas com uma bateria.
- Versão 2: o transmissor irá ser equipado com um módulo GPS e duas baterias sendo que uma será inteiramente dedicada ao módulo GPS para garantir o máximo de precisão das coordenadas e evitar assim o *Cold Start* e enviar coordenadas com uma má precisão.

Quando os utilizadores se dirigirem para as pistas os mesmos deverão preparar os seus dispositivos e caso levem a versão 2 deverão ativar a alimentação do módulo de GPS. Os rádios estarão inteiramente adormecidos, caso ocorra qualquer emergência os dipositivos terão a sua alimentação bloqueada com uma patilha de plástico (como podemos encontrar quando compramos comandos novos) e que está conectada por um cordão ao peito do utilizador, a mesma deverá ser puxada ativando assim o equipamento. Como todos os utilizadores contam também com walkie-talkies é possível enviar um pré-alerta que fará com que outros companheiros se dirijam ao local o mais rápido possível e que possam ir buscar o rádio RX (recetor) em que este irá ser colocado num drone, e como o TX irá estar sempre a enviar dados podemos utilizar o RSSI das mensagens para ter uma estimativa da proximidade, ou caso o TX possuo-a o módulo GPS podemos receber as suas coordenadas e assim ir diretamente.

Teste de funcionamento

Para conseguir ter uma análise melhor e como só possuía 2 dispositivos LORA 868MHZ, coloquei o TX no drone e o RX ficou na base para realizar testes de distância e força de sinal.

Seguem algumas imagens de cada estado dos rádios:

- TX em modo Stand-by

Figura 1 - TX em modo Stand-by

- TX ativa com potência de 1 em 14 para evitar uma propagação muito grande pois dificulta mais e o raio aumenta consideravelmente

Figura 2 - TX ativado com Potência de sinal 1-14

- TX começa a sua transmissão em que irá enviar uma nova mensagem a cada 250ms ou seja 4 mensagens por segundo

Figura 3 - TX em modo de transmissão continua

- RX inicia com potência 14b para maior captação de sinal

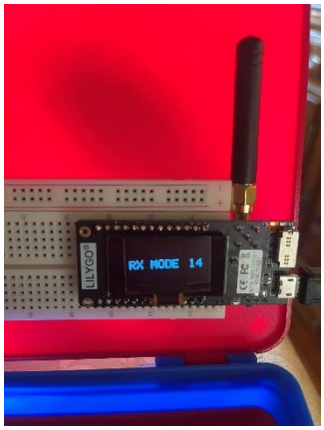

Figura 4 - TX ativado com Potência de sinal 14-14

- Quando o RX encontrar uma mensagem irá apresentar a mesma e irá realizar uma média de RSSI de 4 mensagens para assim evitar uma elevada variação do valor e cada atualização o valor antigo será apresentado na mesma para assim verificar-se se existe ou não uma melhoria

Figura 5 - RX encontra mensagem de SOS

Melhorias para este projeto

Um dos melhores pontos da tecnologia LoRa por vezes torna-se no ponto fraco deste projeto que se trata da distância, como falamos de uma tecnologia que está projetada para transmissões na casa dos KMs pode ser um problema quando queremos determinar proximidades entre RX-TX através do RSSI. Para contornar este problema será necessário criar um algoritmo em que tanto o TX como o RX iniciem com potências elevadas para primeira deteção e ajustarem se consoante a sua proximidade ao ponto de estarem ambos com a potência mínima para assim garantir uma boa variação do valor RSSI.

Linguagem de programação utilizadas

No desenvolvimento do projeto, foram empregues diferentes linguagens de programação, de acordo com as necessidades específicas de cada ambiente. O C++ foi utilizado para a programação do microcontrolador, garantindo eficiência e controle detalhado dos recursos embarcados.

C++

C++ é uma linguagem de programação de propósito geral conhecida por ser poderosa e versátil, derivada do C. Ela oferece suporte à programação orientada a objetos, programação genérica e programação procedural. Seu principal diferencial é a combinação de eficiência de baixo nível (próxima ao hardware) com recursos de alto nível, o que a torna ideal para sistemas de software complexos, como sistemas operacionais, drivers de dispositivos, aplicações em tempo real, jogos e aplicações de alta performance.

C++ oferece suporte a encapsulamento, herança e polimorfismo, permitindo modularidade e reuso de código, além de controle detalhado de recursos, como memória, por meio de ponteiros e gerenciamento manual. Sua flexibilidade também se estende ao desenvolvimento de sistemas embarcados, como o ESP32, onde é amplamente utilizado para criar aplicações eficientes e com recursos avançados, tirando proveito das capacidades de conectividade Wi-Fi e Bluetooth do microcontrolador.

Softwares utilizados

VsCode

É um editor de código leve, gratuito e de código aberto desenvolvido pela Microsoft. Ele é amplamente utilizado por desenvolvedores devido à sua flexibilidade, extensibilidade e suporte a múltiplas linguagens de programação.

As suas principais características são:

- Leve e rápido, projetado para ser ágil, mesmo em máquinas menos potentes.
- Extensões, possui uma vasta biblioteca de extensões, que permitem adicionar suporte para linguagens, frameworks e ferramentas, como Python, Java, Node.js, e PlatformIO.
- Depurador integrado inclui ferramentas para depuração de código, ajudando a identificar e corrigir erros rapidamente.
- IntelliSense oferece sugestões inteligentes de código, como o auto completar e informações contextuais.
- Terminal integrado, permite executar comandos sem sair do editor.
- Controle de versão integrado, compatível com Git e outros sistemas de controle de versão.
- Cross-platform, disponível para Windows, macOS e Linux.

PlatformIO

É um ecossistema de desenvolvimento integrado (IDE) para sistemas embarcados e Internet das Coisas (IoT). Ele oferece uma plataforma unificada para programar microcontroladores como ESP32, Arduino, STM32, Atmel AVR, e etc. Seu principal objetivo é facilitar o desenvolvimento, construção e a depuração de projetos em diversas plataformas e ambientes.

As suas principais características são:

- Compatibilidade com múltiplas plataformas e frameworks:
- Suporta uma ampla gama de plataformas (como Arduino, ESP-IDF, Zephyr, etc.).
- Permite criar projetos que podem ser executados em diferentes arquiteturas de hardware.
- Pode ser usado como por exemplo numa extensão do VS Code (Visual Studio Code).
- Oferece também suporte a uma interface de linha de comando (CLI), ideal para automação e fluxos de trabalho personalizados.
- Sistema de gestão de bibliotecas:

- Inclui um poderoso gerenciador de dependências para bibliotecas e frameworks, garantindo que todas as versões necessárias sejam instaladas automaticamente.
- Ambiente multiplataforma
- Inclui ferramentas para depuração de código em tempo real, algo que o Arduino IDE, por exemplo, não oferece nativamente.
- Permite testes unitários diretamente no microcontrolador ou em simulações.

Por que usar o PlatformIO?

- Flexibilidade: Você pode trabalhar em projetos complexos que requerem bibliotecas e frameworks diferentes.
- Produtividade: Ele automatiza tarefas, como download de dependências e configuração do ambiente.
- Integração com ferramentas modernas: É ideal para quem quer usar recursos avançados, como integração contínua (CI/CD) ou ferramentas como Git.

Figura 6 – PlatformIO

Git

Git é um sistema de controle de versão distribuído amplamente utilizado no desenvolvimento de software. Ele permite que desenvolvedores rastreiem alterações em seu código-fonte, colaborem de maneira eficiente em equipes e revertam para versões anteriores do projeto quando necessário. Criado por Linus Torvalds em 2005, o Git é conhecido por sua rapidez, confiabilidade e flexibilidade.

Com Git, as mudanças são registradas em um repositório que armazena todo o histórico de modificações, possibilitando o controle detalhado das versões do código. Cada desenvolvedor possui uma cópia local do repositório, permitindo o trabalho offline e a realização de experimentos sem impactar a versão principal do projeto. Ele utiliza conceitos como branches (ramificações) para que múltiplos desenvolvedores possam trabalhar em diferentes funcionalidades ou correções de forma isolada antes de mesclá-las ao projeto principal.

Além disso, Git facilita a colaboração por meio de plataformas de hospedagem como GitHub, GitLab e Bitbucket, que oferecem recursos adicionais como gerenciamento de issues, revisões de código, e integração contínua (CI/CD). Essas funcionalidades tornam Git uma ferramenta essencial para equipes de desenvolvimento modernas, ajudando a gerenciar o ciclo de vida dos projetos de forma eficiente e segura.

Figura 7 - Git

Microcontroladores

Os microcontroladores desempenham o papel de "cérebro" neste projeto, pois são responsáveis, por meio da programação em C++, pelo controle da lógica do sistema. Eles permitem a leitura de sinais de entrada e a gestão das saídas, viabilizando o funcionamento desejado do dispositivo. Atualmente, há diversos tipos de microcontroladores disponíveis no mercado, dos quais alguns dos principais serão mencionados mais adiante neste trabalho.

Componentes Utilizados no Projeto

Lilygo Lora32 v2.1.6

É uma placa de desenvolvimento baseada no ESP32 que integra conectividade LoRa, tornando-a ideal para projetos de IoT e redes de longa distância.

Processador e Memória

Microcontrolador ESP32-D0WDQ6 com processador dual-core de 32 bits

Clock de até 240 MHz

Flash, geralmente de 4 MB

RAM, 520 KB SRAM interna no ESP32

Módulo LoRa

Chip LoRa, semtech SX1276 com frequência de 868MHZ

Suporte para comunicação bidirecional LoRaWAN

Conectividade

Wi-Fi: IEEE 802.11 b/g/n

Bluetooth

LoRa, comunicação de longo alcance com baixo consumo.

Alimentação

Conector JST para baterias externas.

Circuito integrado de carregamento (geralmente usa-se o chip IP5306).

Alimentação USB

Recursos de Hardware

Display OLED integrado:

Suporte para I2C, SPI e UART.

Slot para armazenamento externo.

LEDs integrados

Dimensões e Forma

Compacta: Dimensões aproximadas de 25 mm x 75 mm, facilitando a integração em projetos portáteis.

Software e Suporte

Programável com:

- Arduino IDE
- PlatformIO
- Frameworks como ESP-IDF
- Protocolos: Suporte para LoRaWAN e comunicação ponto a ponto (P2P)

Figura 8 - Lilygo Lora32 v2.1.6

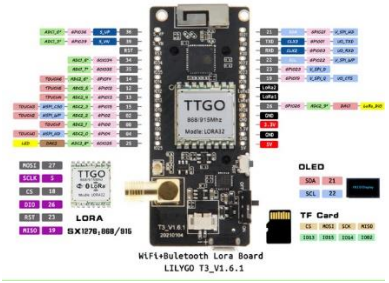

Figura 9 - Lilygo Lora32 v2.1.6 PINOUT

Código fonte

Sistema

```
#include"SendSignal\SendSignal.h"
#include"FoundSignal\FoundSignal.h"
#include"Lcd\lcd.h"
#include "HardwareSerial.h"

#define SELECTOR 34
#define FREQUENCIE 868E6
#define SS 18
#define RESET 23
#define DI0 26
#define SCL 5
#define MISO 19
#define MOSI 27
#define GPSTX 17
#define GPSRX 16
bool gpsEnable = false;

//Cria os objetos transmissor e emissor
FoundSignal foundSignal;
SendSignal sendSignal(gpsEnable, GPSTX, GPSRX, 9600);
//Cria UART
HardwareSerial SerialGps(1); //Ativa a UART 1

Lcd lcd;
//Declaração das funções
bool selectorChange(int);



void setup() {
    pinMode(SELECTOR, INPUT_PULLUP);
    Serial.begin(115200);
    if(sendSignal.gps) {
        SerialGps.begin(sendSignal.getBaud(),SERIAL_8N1,
sendSignal.getTx(), sendSignal.getRx());
    }
    lcd.init(128, 64, 21, 22);
}


void loop() {
    bool txMode = false;
    lcd.setMessageDisplay("WELCOME", 20, 30, 2);
    delay(2000);
```

```
    bool deviceSelected = false;
    //selectorChange(SELECTOR) ? txMode = false : txMode = true;
    String power;
    !txMode ? power = String(foundSignal.lora.setPower(14)) : power =
String(sendSignal.lora.setPower(1));
    txMode ? lcd.setMessageDisplay("TX MODE", 0, 30, 2, power, 100, 30,
2) : lcd.setMessageDisplay("RX MODE", 0, 30, 2, power, 100, 30, 2);
    delay(5000);

    //Loop para rádios TX
    while(txMode) {
        lcd.setMessageDisplay(sendSignal.sendEmergencyContacts(), 20, 30,
2);
        delay(500);
        selectorChange(SELECTOR) ? ESP.restart() : void();
    }

    //Loop para rádios RX
    while(!txMode) {
        foundSignal.FoundMessage();
        if(foundSignal.signal && !foundSignal.lostSignal) {
            lcd.setMessageDisplay(foundSignal.response, 0, 0, 2,
foundSignal.values, 0, 40, 1);
        }
        if(millis() - foundSignal.lastMessage >= 10000) {
            foundSignal.signal ?  lcd.lostSignal() : lcd.noSignal();
            foundSignal.lostSignal = true;
            foundSignal.count = 0;
            }
        selectorChange(SELECTOR) ? ESP.restart() : void();
    }
}

//Seletor para caso se queira inserir um botão para a alteração de TX
para RX ou vice versa
bool selectorChange(int selector){
    bool change;
    digitalRead(selector) == 1 ? change = true : change = false;
    return change;
}
```

Classe FoundSignal

Interface

```
#pragma once
#include<Arduino.h>
#include<Lora\Lora.h>
#include<SPI.h>


class FoundSignal{
    public:
        int tempRSSI;
        int count = 0;
        int lastRssi = 0;
        int newRssi = 0;
        bool signal = false;
        bool lostSignal = true;
        unsigned long lastMessage = millis();
        String response;
        String values;
        Lora lora;
        FoundSignal() = default;
        double getDistanceRssi(double, int);
        void FoundMessage();


    private:

};
```

Implementação

```
#include"FoundSignal.h"



double FoundSignal::getDistanceRssi(double avgValues, int rssi) {
    double distance;
    //Math.pow(10.0, ((Math.abs(this) - Math.abs(txPower)) / (10 *
2)).toDouble())

    return distance;
};


void FoundSignal::FoundMessage() {
    if(lora.getMessage()) {
```

```
        tempRSSI += lora.rssi;
        Serial.println(lora.response);
        lastMessage = millis();
        signal = true;
        lostSignal = false;
        count++;
        if(count == 4) {
            count = 0;
            lastRssi = newRssi;
            newRssi = tempRSSI/4;
            tempRSSI = 0;
            values = "last RSSI: " + String(lastRssi);
            values += "\nnew RSSI: " + String(newRssi);
        }
        response = lora.response;
    }
};
```

Classe SendSignal

Interface

```
#pragma once
#include"GPS\Gps.h"
#include"Lora\Lora.h"

class SendSignal {
    public:
        int count = 0;
        bool emergency;
        bool gps;
        String emergencyMessage;
        Gps gpsNeo;
        Lora lora;

        SendSignal(bool gps, int gpsTx, int gpsRx, unsigned long gpsBaud,
String emergencyMessage = "SOS") {
            this->emergency = false;
            this->emergencyMessage = emergencyMessage;
            this-> gps = gps;
            this-> gpsRx = gpsRx;
            this-> gpsTx = gpsTx;
            this->gpsBaud = gpsBaud;
            if(gps){gpsNeo.init(gpsTx, gpsRx, gpsBaud);}
            lora.init();
        };
```

```
        int getRx();
        int getTx();
        unsigned long getBaud();
        void sendSOS(int, String);
        void sendCoordinates();
        String sendEmergencyContacts();
        void powerSave();

    private:
        unsigned long gpsBaud;
        int gpsTx;
        int gpsRx;

};
```

Implementação

```
#include"SendSignal.h"


int SendSignal::getRx(){
    return gpsRx;
}

int SendSignal::getTx(){
    return gpsTx;
}

unsigned long SendSignal::getBaud(){
    return gpsBaud;
}

void SendSignal::sendSOS(int timePerMessage, String message){
    unsigned long lastTime = millis();
    while(true) {
        unsigned long newTime = millis();
        if((newTime -lastTime) >= timePerMessage) {
            sendCoordinates();
            sendEmergencyContacts();
        }
    }
};

void SendSignal::sendCoordinates() {

    if(gpsNeo.getData()) {
```

```
        Serial.println("I am lost please search me in these
coordinates:");
        Serial.print("Latitude: "); Serial.println(gpsNeo.latitude);
        Serial.print("Longitude: "); Serial.println(gpsNeo.longitude);
        Serial.print("My speed is Kmp/h: ");
Serial.println(gpsNeo.speed);
        Serial.print("This message has HDOP value: ");
Serial.println(gpsNeo.hdop);
    }
};

String SendSignal::sendEmergencyContacts() {
    String message = emergencyMessage + ": " + count;
    Serial.println(message);
    lora.message = message;
    lora.sendMessage();
    count++;
    if(count == 1000) {count = 0;}
    return message;
}
```

Classe Gps

Interface

```
#pragma once
#include<Arduino.h>
#include"TinyGPS++.h"

class Gps {
    public:
        int speed;
        double latitude;
        double longitude;
        int satelites;
        int hdop;
        TinyGPSPlus gps;

        Gps() {};
        void init(int TxPin, int RxPin, unsigned long baud);
        bool getData();


    private:
        unsigned long baud;
        int TxPin;
        int RxPin;
```

```
};
```

Implementação

```
#include"Gps.h"

void Gps::init(int TxPin, int RxPin, unsigned long baud) {
    this->TxPin = TxPin;
    this->RxPin = RxPin;
    this->baud = baud;
    if(Serial) {
        Serial.println("GPS inicialized.");
    }
};

bool Gps::getData() {
    if(gps.satellites.value() > 4) {
        speed = gps.speed.kmph();
        latitude = (gps.location.lat(), 6);
        longitude = (gps.location.lng(), 6);
        satelites = gps.satellites.value();
        hdop = gps.hdop.value();
        return true;
    } else { return false;}
};
```

Classe Lcd

Interface

```
#pragma once

#include<Arduino.h>
#include <memory.h>
#include <Wire.h>
#include<Adafruit_SSD1306.h>

#include <FS.h>
#include "SPIFFS.h"


class Lcd {
    public:
        String displayMessage = "";
        String message = "";
```

```
        int displayPosX = 0;
        int displayPosY = 0;
        void init(int width, int height, int sdaPin, int sclPin);
        void setMessageDisplay(String message, int posX, int posY, int
size, String message2 = "", int posX2 = 0, int posY2 = 0, int size2 = 0);
        void noSignal();
        void lostSignal();

    private:
        int width = -1;
        int height = -1;
        int sdaPin = -1;
        int sclPin = -1;
        int resetPin = -1;
        int address = 0x3C;
        Adafruit_SSD1306* lcd;
};
```

Implementação

```
#include"lcd.h"

void Lcd::init(int width, int height, int sdaPin, int sclPin) {
    this->width = width;
    this->height = height;
    this->sdaPin = sdaPin;
    this->sclPin = sclPin;
    this->lcd = new Adafruit_SSD1306(width, height, &Wire, resetPin);
    if(!lcd-> begin(SSD1306_SWITCHCAPVCC, address)) {
        Serial.println(F("SSD1306 allocation failed"));
        for(;;); // Don't proceed, loop forever
    }
    lcd-> clearDisplay();
    lcd-> display();
    delay(2000);
};

void Lcd::noSignal() {
    lcd-> clearDisplay();
    lcd-> setCursor(50, 20);
    lcd-> print("NO");
    lcd-> setCursor(45, 40);
    lcd-> print("SIGNAL");
    lcd-> display();
}

void Lcd::lostSignal() {
```

```
    lcd-> clearDisplay();
    lcd-> setCursor(50, 20);
    lcd-> print("LOST");
    lcd-> setCursor(45, 40);
    lcd-> print("SIGNAL");
    lcd-> display();
}


void Lcd::setMessageDisplay(String message, int posX, int posY, int size,
String message2, int posX2, int posY2, int size2) {
    lcd-> clearDisplay();
    lcd-> setTextColor(SSD1306_WHITE);
    lcd-> setCursor(posX, posY);
    lcd-> setTextSize(size);
    lcd-> print(message);
    lcd-> setCursor(posX2, posY2);
    lcd-> setTextSize(size2);
    lcd-> print(message2);
    lcd-> display();
}
```

Classe Lora

Interface

```
#pragma once
#include<Arduino.h>
#include<LoRa.h>
#include <SPI.h>

class Lora {
    public:
    String message;
    String response;
    int packetSize;
    int rssi;

    void init();
    void sendMessage();
    bool getMessage();
    int setPower(int power);

    private:

};
```

Implementação

```
#include"Lora\Lora.h"


void Lora::init(){

    SPI.begin(5, 19, 27, 18);
    Serial.println("SPI set pins!");
    LoRa.setPins(18,23,26);
    Serial.println("LoRa set pins!");
    if (!LoRa.begin(868E6)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
};


void Lora::sendMessage() {
    LoRa.beginPacket();
    LoRa.print(message);
    LoRa.endPacket();
    Serial.println("Lora send message!!");
};

bool Lora::getMessage() {
  packetSize = LoRa.parsePacket();
  String message = "";
  if (packetSize) {
    while(LoRa.available()) {
      char text = LoRa.read();
      message += text;
    }
    rssi = LoRa.rssi();
    response = message;
    return true;
  } else { return false;}
};



int Lora::setPower(int power) {
  LoRa.setTxPower(1);
  return power;
};
```

Conclusão

O projeto LoRa Emergency Message Finder demonstrou o potencial da tecnologia LoRa para resolver desafios em operações de busca e salvamento em ambientes com pouca ou nenhuma cobertura de rede móvel. A integração de dispositivos de transmissão e recepção LoRa, com a possibilidade de uso de RSSI ou coordenadas GPS para localização, oferece uma abordagem versátil e eficiente para localizar indivíduos em emergências.

A utilização de drones para o transporte do recetor mostrou-se uma solução prática e inovadora, permitindo ampliar a área de cobertura e reduzir significativamente o tempo necessário para encontrar um sinal. Este modelo pode ser ainda mais aprimorado com algoritmos de ajuste dinâmico de potência de transmissão e receção, melhorando a precisão das estimativas baseadas no RSSI.

O desenvolvimento do sistema apresentou não apenas viabilidade técnica, mas também um impacto potencial significativo em cenários reais, como resgates em áreas montanhosas ou regiões sujeitas a desastres naturais. Avanços futuros podem incluir o uso de tecnologias adicionais, como redes neuronais para análise de padrões de sinal e integração com sistemas de comunicação emergenciais globais.

O LoRa *Emergency Message Finder* é um exemplo de como a combinação de tecnologias modernas e criatividade pode levar a soluções práticas e acessíveis, salvando vidas em situações críticas.