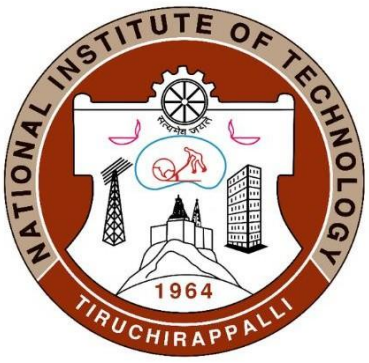
Thesis on

LoRa Transceiver based

Emergency Response System



Submitted to

CoE-ERSS CDAC Thiruvananthapuram

NIT Tiruchirappalli

Submitted by

107121006	**Akshaiy Thinakaran**
107121016	**Atharva Rathi**
107121028	**Diya S Dileep**

Faculty in charge

Dr. Sishaj P Simon

Dept. of EEE
National Institute of Technology, Tiruchirappalli

Objective

Motivation and Application:

The project is motivated by the critical need for efficient and reliable communication systems during emergency situations, such as natural disasters, accidents, or medical emergencies.

Traditional communication infrastructure may be compromised or unavailable in such scenarios, emphasizing the importance of alternative communication methods like LoRa technology.

Aim:

The aim of the project is to develop a LoRa Transceiver-based Emergency Response System capable of providing seamless and long-range communication for distress signaling and coordination during emergencies.

The system aims to overcome the limitations of existing communication systems by leveraging the advantages of LoRa modulation, including long-range communication, low-power operation, and robustness in challenging environments.

Approach:

Hardware and Sensor Integration: Implemented LoRa transceivers, microcontrollers, and sensors for data acquisition and transmission, ensuring compatibility with emergency response requirements.

Software Development: Developed firmware for microcontrollers to handle LoRa communication protocols and sensor data processing. Implement distress signal detection algorithms for encoding and transmission.

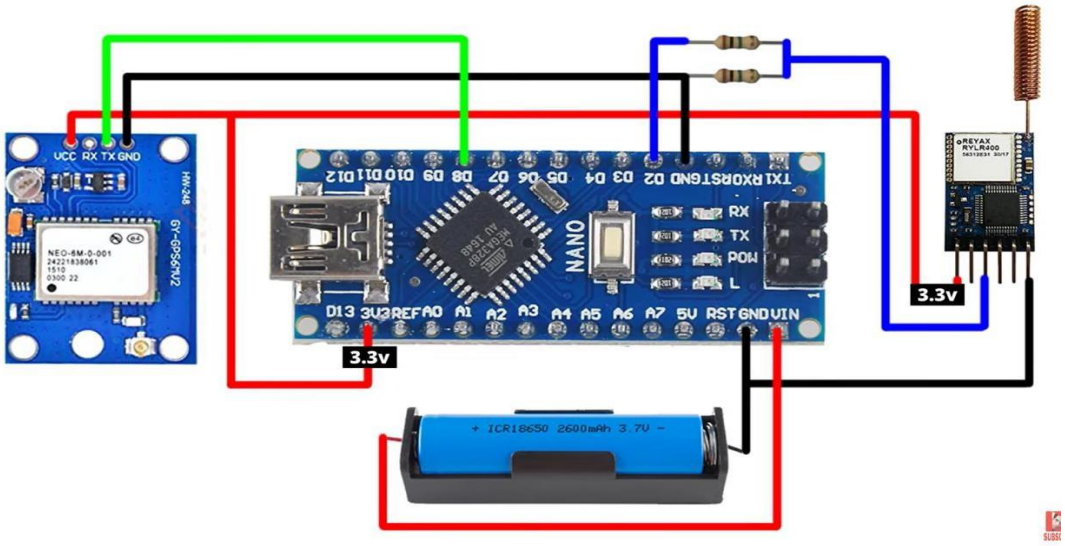
Ant Colony Optimization (ACO): Integrate ACO algorithm for finding the shortest path between two coordinates on a map or geographical area, optimizing the routing of emergency responders.

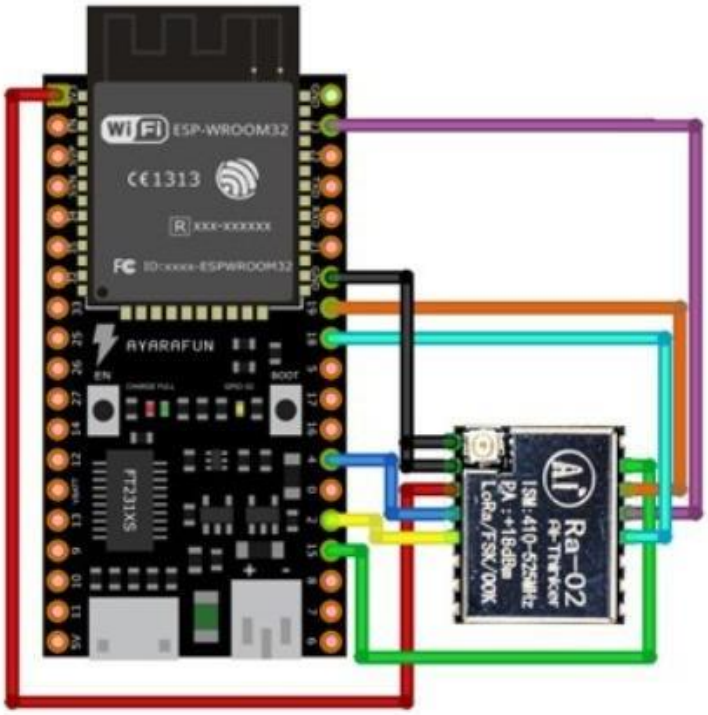
Integration and Testing: Integrate ACO algorithm with the overall software framework for emergency response. Conduct extensive testing to validate the functionality and performance of the emergency response system under various scenarios.

Deployment and Optimization: Deploy the system in simulated or real-world environments, gathering feedback for optimization and fine-tuning of ACO parameters and algorithms to enhance efficiency and reliability.

Keywords: LoRa, Emergency Response System, Distress Signal, Long-range Communication, Ant Colony Optimization.

Circuit Schematic



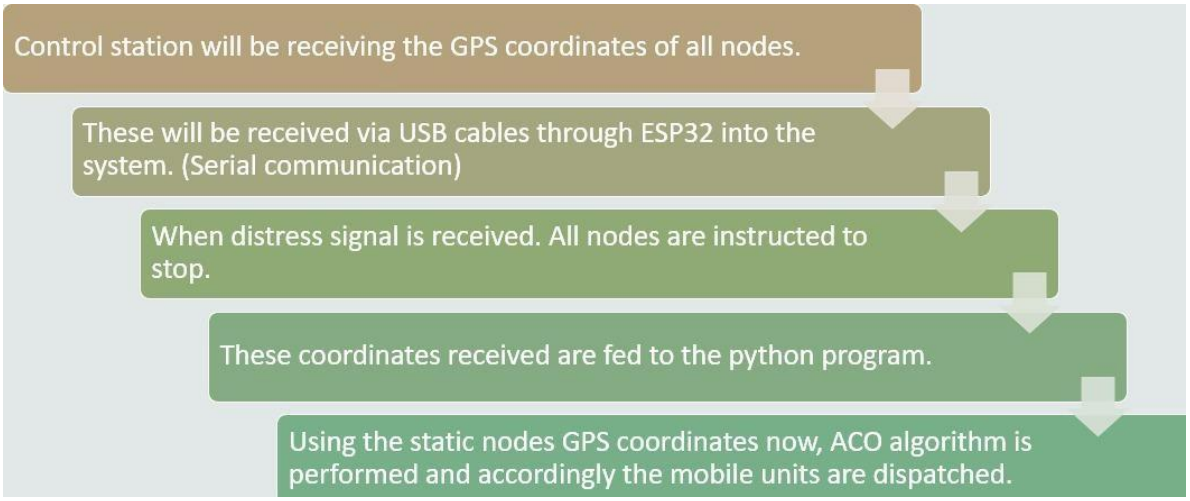


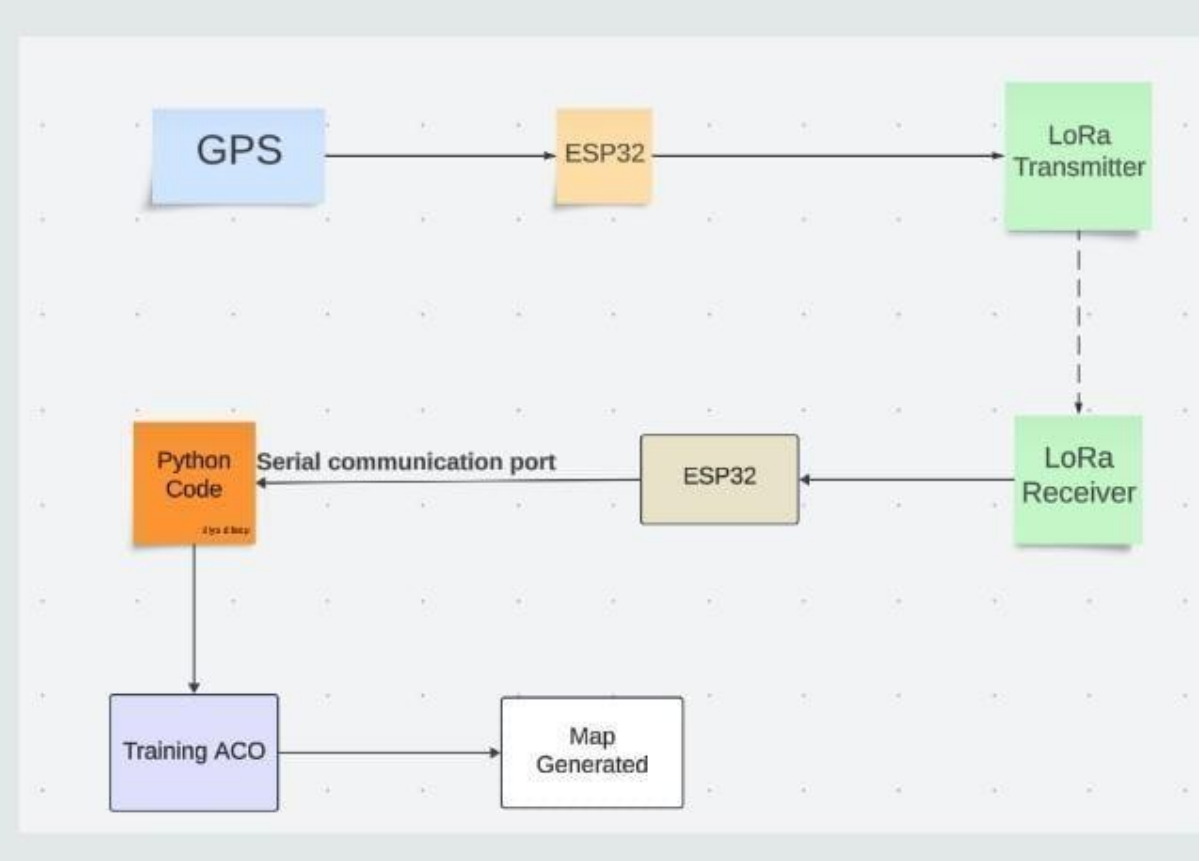
Working Principle

The emergency response system operates through the integration of hardware components and software algorithms, ensuring swift and effective communication and coordination during critical situations. Hardware components include LoRa transceivers, GPS modules, and microcontrollers, strategically deployed across the emergency response network. Each node in the system, comprising the LoRa transceiver, GPS module, and microcontroller (e.g., ESP32), continuously retrieves location coordinates and transmits them wirelessly to the central control station. Upon receiving distress signals from any node, the control station initiates a coordinated response by processing the received location coordinates. This is achieved through a Python program running on the control station, which efficiently manages data processing and route planning.

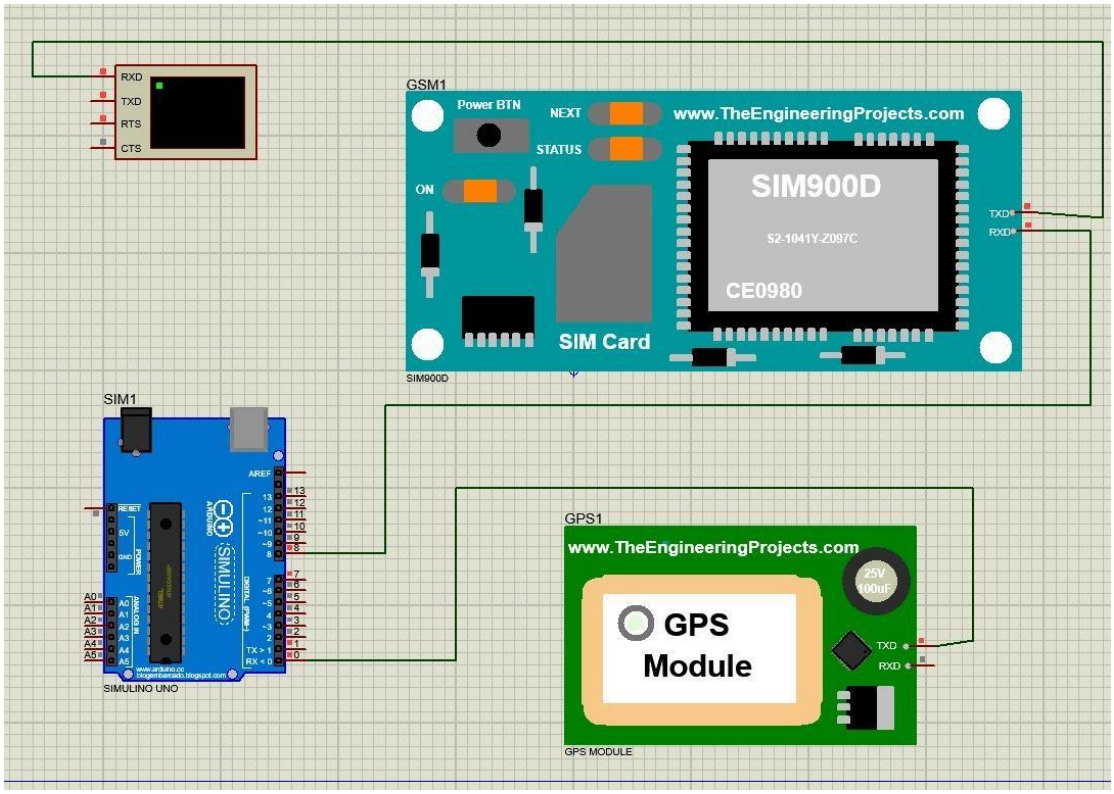
Utilizing the Ant Colony Optimization (ACO) algorithm, the Python program determines the optimal path from each node to the distress signal location. By integrating the static coordinates of the nodes with the dynamic distress signal location, the ACO algorithm calculates the shortest and most efficient routes for dispatching mobile response units. These routes are dynamically generated and updated in real-time to adapt to changing conditions and optimize response times. The control station coordinates the dispatch of response units, ensuring a rapid and coordinated response to the distress signal.

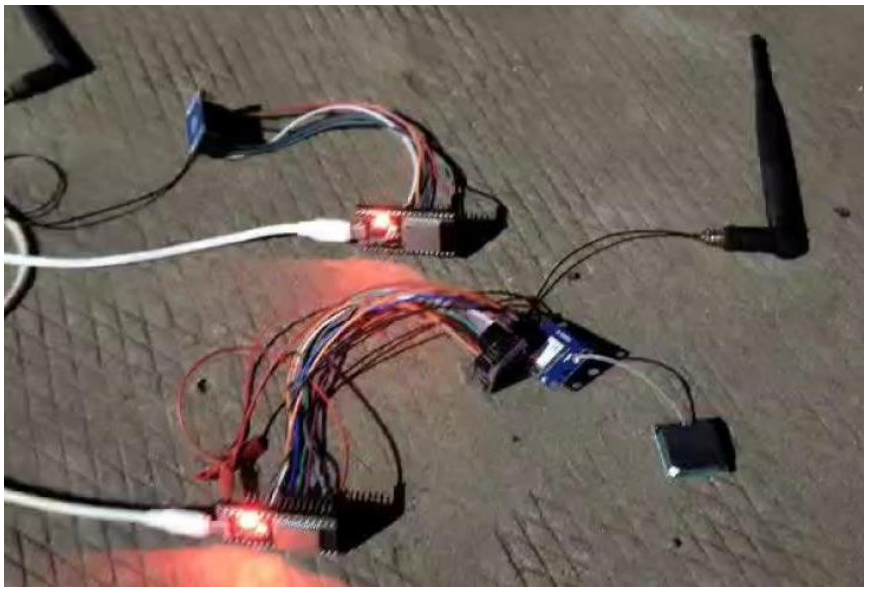
The system's operational flow is meticulously managed to ensure seamless coordination and communication. Nodes are instructed to halt operations upon receipt of a distress signal to prioritize the emergency response. Location coordinates are processed in real-time, enabling the timely execution of the ACO algorithm to determine optimal routes. Response units are dispatched based on the generated routes, facilitating prompt and efficient emergency response operations. Continuous monitoring and adaptation further enhance the system's responsiveness, allowing for dynamic adjustments to routing plans as needed. Overall, the integration of hardware components and software algorithms enables the emergency response system to effectively coordinate and respond to distress signals, ultimately improving outcomes during emergency situations.





Circuit Diagram





Resul

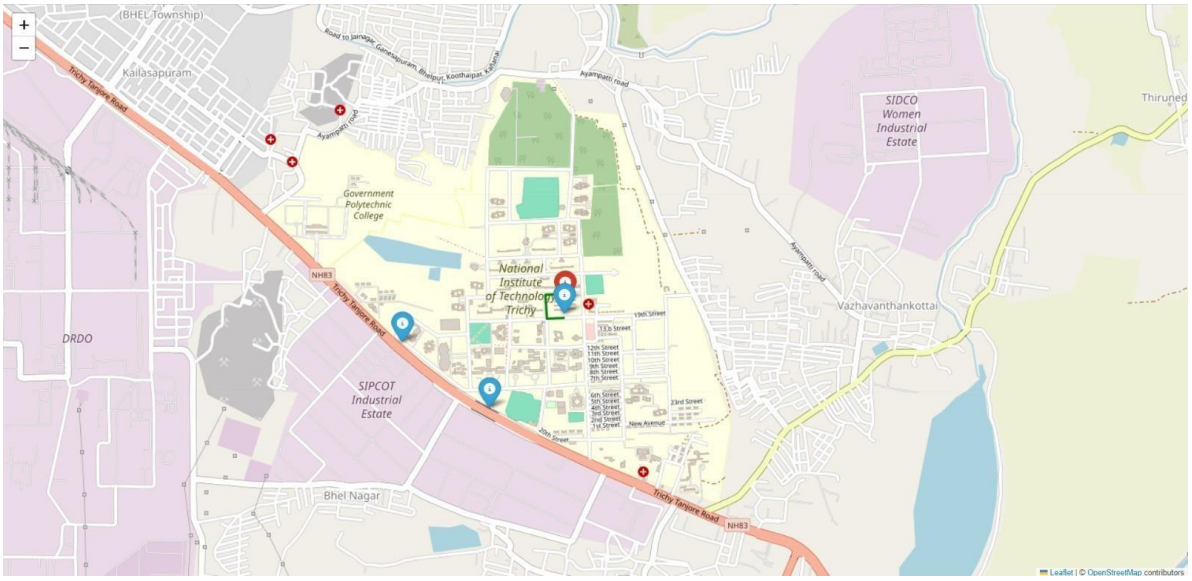
Virtual Terminal

```
Latitude = 30.240455 Longitude = -97.817710

Latitude = 30.240455 Longitude = -97.817710

Latitude = 30.239772 Longitude = -97.815689

Latitude = 30.240455 Longitude = -97.817710
```



Program

1. Code for ACO training algorithm

```
import requests
import folium
import random
import numpy as np


# Function to calculate distance between two points
def distance(point1, point2):
    return np.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)


def yolo():
    from serial.tools import list_ports
    import serial
    import time
    import csv

    # Identify the correct port
    ports = 'COM4'
    for port in ports: print(port)

    # Create CSV file
    # f = open("data.csv","w",newline='')
    # f.truncate()
```

```

    # Open the serial com
    serialCom = serial.Serial('COM4', 115200)

    # Toggle DTR to reset the Arduino
    serialCom.setDTR(False)
    time.sleep(1)
    serialCom.flushInput()
    serialCom.setDTR(True)

    # How many data points to record
    kmax = 20

    # Loop through and collect data as it is available
    for k in range(kmax):
        try:
            # Read the line
            s_bytes = serialCom.readline()
            decoded_bytes = s_bytes.decode("utf-8").strip('\r\n')
            print(decoded_bytes)

            # Parse the line
            if k == 0:
```

```
                print(decoded_bytes)
                values = str(decoded_bytes).split(",")
                # print(values)
                pass
            else:
                values = str(decoded_bytes).split(",")
                pass
        except:
            pass

        # f.close()
    print(values)
    lat = values[0]
    long = values[1]
    return {lat, long}


# Function to perform Ant Colony Optimization (ACO) to find the closest city
def ant_colony_optimization(points, central_point, n_ants, n_iterations, alpha):
    n_points = len(points)
    pheromone = np.ones(n_points)
    closest_city = None
    min_distance = np.inf



  for iteration in range(n_iterations):
      for ant in range(n_ants):
          visited = [False] * n_points
          current_point = np.random.randint(n_points)
          visited[current_point] = True

          while False in visited:
              unvisited = np.where(np.logical_not(visited))[0]
              probabilities = np.zeros(len(unvisited))

              for i, unvisited_point in enumerate(unvisited):
                  dist = distance(points[current_point], points[unvisited_point])
                  if dist == 0:  # Prevent division by zero
                      probabilities[i] = 0
                  else:
                      probabilities[i] = pheromone[unvisited_point] / dist

              total_prob = np.sum(probabilities)

              if total_prob == 0 or np.isnan(total_prob):  # Handle zero probabilities or NaN
                  probabilities = np.ones(len(unvisited))
                  probabilities /= len(unvisited)
```

```
                else:
                    probabilities /= total_prob

                next_point = np.random.choice(unvisited, p=probabilities)
                visited[next_point] = True
                current_point = next_point

            route_distance = distance(points[current_point], central_point)
            if route_distance < min_distance:
                closest_city = current_point
                min_distance = route_distance

        # Update pheromone levels based on the closest city found
        pheromone *= (1 - alpha)
        pheromone[closest_city] += alpha / min_distance

    return closest_city


# Function to fetch route information between locations using OpenRouteService API
def get_route_between_locations(api_key, start_coords, end_coords):
    url = f'https://api.openrouteservice.org/v2/directions/driving-car?api_key={api_key}&start={start_coords[1]},{start_coords[0]}&end={end_coords[1]},{end_coords[0]}'
    response = requests.get(url)

    if response.status_code == 200:
        route_data = response.json()
        return route_data
    else:
        print("Failed to fetch route data.")
        return None


# Replace 'your_api_key' with your actual API key from OpenRouteService
api_key = '5b3ce3597851110001cf624871201fb2606e49f58686268881b1ac15'

a, b = yolo()
a = float(a)
b = float(b)
print(a)
print(b)
# Latitude and longitude coordinates for Central City
```

```
central_location = (a, b)  # Central City (e.g., New York)
10
# Generate four random spots around the central location
random_spots = [
    (10.756953, 78.813263), (10.762093, 78.817408), (10.760537, 78.808403)
]

# Create a map centered on the central location
mymap = folium.Map(location=central_location, zoom_start=5)

# Add marker for the central location
folium.Marker(location=central_location, popup='Central City', icon=folium.Icon(color='red')).add_to(mymap)

# Add markers for all random spots
for idx, spot in enumerate(random_spots, start=1):
    folium.Marker(location=spot, popup=f'Random Spot {idx}', icon=folium.Icon(color='blue')).add_to(mymap)

# Run ACO to find the closest city to the central location
closest_city_idx = ant_colony_optimization(random_spots, central_location, n_ants=10, n_iterations=100, alpha=1.0)
closest_city = random_spots[closest_city_idx]

# Get route information from the closest city to the central location
route_info = get_route_between_locations(api_key, closest_city, central_location)
if route_info:
    # Extract route geometry (coordinates) from route_info
    route_geometry = route_info['features'][0]['geometry']['coordinates']

    # Add route polyline to the map (in green color)
    folium.PolyLine(locations=[(lat, lon) for lon, lat in route_geometry], color='green', weight=3).add_to(mymap)
else:
    print("Failed to fetch route data for the closest city to the central location")

# Save map as HTML
map_file = "final12.html"
mymap.save(map_file)

# Provide a link to the HTML file
print(f"Map saved as '{map_file}'. Click the link to view the map: ")
print(f'<a href="{map_file}" target="_blank">Open Route Map</a>')
```

2.Sender's Side-

```
#include <SPI.h>
#include <LoRa.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

#define ss 5
#define rst 14
#define dio0 2

TinyGPSPlus gps;
SoftwareSerial gpsSerial(16, 17);

void setup() {
  Serial.begin(115200);
  gpsSerial.begin(9600);
  while (!Serial);

  Serial.println("LoRa Sender");
  LoRa.setPins(ss, rst, dio0);
  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
}
```

```
void loop() {

 while (gpsSerial.available() > 0)
 {
    if (gps.encode(gpsSerial.read()))
    {
        if (gps.location.isValid())
        {
          delay(1000);
          Serial.println("Sending to LoRa");
          LoRa.beginPacket();
          //LoRa.print("Lat: ");
          LoRa.print(gps.location.lat(), 6);
          //LoRa.print(" Long: ");
          LoRa.print(",");
          LoRa.print(gps.location.lng(), 6);
          Serial.println("Sent via LoRa");
          LoRa.endPacket();
        }
    }
 }
}
```

3. Reciever's Side-

```
#include <SPI.h>
#include <LoRa.h>

#define ss 5
#define rst 14
#define dio0 2


void setup() {
  Serial.begin(115200);
  while (!Serial);
  delay(500);
  Serial.println("LoRa Receiver");
  LoRa.setPins(ss, rst, dio0);
  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("Starting LoRa failed!");
}

void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // received a packet
    Serial.println();
    Serial.print("Received packet ");

    // read packet
    while (LoRa.available()) {
     char incoming = (char)LoRa.read();
        Serial.print(incoming);

    }
  }
}
```

Outcomes

The outcomes of the project encompass both tangible deliverables and intangible impacts, reflecting the successful implementation and utilization of the LoRa Transceiver-based Emergency Response System:

Functional Emergency Response System: The primary outcome of the project is the development of a fully functional emergency response system capable of providing efficient and reliable communication and coordination during emergency situations. The system integrates LoRa transceivers, GPS modules, and microcontrollers, facilitating data acquisition, transmission, and processing. It demonstrates the successful integration of hardware components and software algorithms, including the implementation of the Ant Colony Optimization (ACO) algorithm for route planning.

Improved Response Time and Efficiency: By leveraging the capabilities of LoRa technology and intelligent routing algorithms, the emergency response system enhances response time and operational efficiency. Optimal routes generated by the ACO algorithm enable prompt dispatch of mobile response units to the distress signal location, minimizing delays and improving overall emergency response effectiveness.

Enhanced Resilience and Reliability: The utilization of LoRa technology enhances the resilience and reliability of communication channels during emergency situations. LoRa's long-range communication capabilities and low-power operation ensure robust communication links even in remote or challenging environments where traditional communication infrastructure may be compromised.

Scalability and Adaptability: The modular design of the emergency response system allows for scalability and adaptability to varying emergency scenarios and operational requirements. The system can be easily expanded or modified to accommodate additional nodes, integrate new sensors, or incorporate advanced algorithms for enhanced functionality and performance.

Impact on Emergency Management Practices: The successful implementation of the emergency response system demonstrates the potential impact of innovative technologies on emergency management practices. It serves as a practical example of how modern communication technologies, coupled with intelligent algorithms, can revolutionize emergency response strategies, ultimately saving lives and mitigating the impact of emergencies on affected communities.

Potential for Deployment and Adoption: The project outcomes lay the foundation for the deployment and adoption of similar emergency response systems in real-world scenarios. The demonstrated effectiveness and efficiency of the system showcase its potential utility in various emergency response applications, ranging from natural disasters to medical emergencies, thereby contributing to the advancement of emergency management practices on a broader scale.

Overall, the outcomes of the project reflect the successful development and implementation of a LoRa Transceiver-based Emergency Response System, with far-reaching implications for improving emergency response capabilities and enhancing resilience in the face of unforeseen crises.

Conclusion

In conclusion, the development and implementation of the LoRa Transceiver-based Emergency Response System represent a significant milestone in the realm of emergency management technology. This project has demonstrated the effectiveness of leveraging modern communication technologies, such as LoRa, and intelligent routing algorithms to enhance communication, coordination, and response efficiency during critical situations. By integrating hardware components and software algorithms, the system has shown promising results in improving response time, operational efficiency, and overall resilience in the face of emergencies. Moving forward, the outcomes of this project provide valuable insights and opportunities for further research, development, and deployment of similar systems in diverse emergency response scenarios. Ultimately, the impact of this project extends beyond its immediate scope, contributing to the advancement of emergency management practices and the safeguarding of communities worldwide.

Limitations

While the LoRa Transceiver-based Emergency Response System shows promise in enhancing emergency management practices, it also presents some limitations and challenges:

Limited Coverage Range: Despite LoRa's long-range communication capabilities, the system's coverage range may still be limited in certain geographical areas with obstacles such as buildings, mountains, or dense vegetation, potentially hindering communication reliability and response effectiveness in such areas.

Dependency on Infrastructure: The system relies on the availability of infrastructure such as power sources and communication networks to operate effectively. In remote or disaster-stricken areas with limited infrastructure, maintaining system functionality may pose challenges and require additional measures for power supply and network connectivity.

Accuracy of GPS Data: The accuracy of GPS data retrieved from GPS modules may vary depending on factors such as signal interference, satellite visibility, and atmospheric conditions. Inaccurate or inconsistent GPS data could affect the precision of distress signal localization and routing decisions, leading to suboptimal response outcomes.

Complexity of Algorithm Implementation: Implementing and fine-tuning algorithms such as the Ant Colony Optimization (ACO) algorithm for route planning may be complex and resource-intensive, requiring expertise in algorithm design and optimization. Managing the computational load and ensuring real-time responsiveness of the system may present challenges, particularly in large-scale deployment scenarios.

Interference and Congestion: In densely populated areas or during high-demand periods, the LoRa communication channel may experience interference and congestion, potentially leading to packet loss or delays in data transmission. Mitigating interference and optimizing

channel utilization to maintain communication reliability are ongoing challenges in LoRa-based systems.

Security and Privacy Concerns: Ensuring the security and privacy of transmitted data is crucial, especially in sensitive emergency response scenarios. The LoRa communication protocol may be susceptible to eavesdropping and unauthorized access, highlighting the need for robust encryption and authentication mechanisms to protect sensitive information.