Public Transport Optimization

Phase 5: Project Documentation & Submission

In this part you will document your project and prepare it for submission.

Document the Public Transportation Optimization project and prepare it for submission.

Documentation

Describe the project's objectives, IoT sensor deployment, platform development, and code implementation.

Include diagrams, schematics, and screenshots of the IoT sensors, transit information platform, and real-time data display.

Explain how the real-time transit information system can improve public transportation services and passenger experience

1. Objectives:

The primary objectives of the project are to improve the efficiency, safety, and passenger experience in public transport systems. This can be achieved by deploying IoT sensors and developing a platform that collects and analyzes real-time data from the sensors. The project aims to optimize fleet management, route planning, capacity utilization, and provide timely information to passengers.

2. IoT Sensor Deployment:

IoT sensors need to be strategically deployed in public transport vehicles and infrastructure to capture relevant data. Some examples of sensor deployment include:

- Vehicle-mounted sensors to track location, speed, and route information.

- Passenger counting sensors at entry and exit points to monitor occupancy levels.

- Environmental sensors to measure air quality, temperature, and humidity.

- Sensors to monitor vehicle components for predictive maintenance purposes.

The sensors should be capable of collecting data accurately and transmitting it wirelessly to a central platform for analysis.

3. Platform Development:

A central platform needs to be developed to collect, process, and analyze the data from the deployed IoT sensors. The platform should have the following features:

- Data ingestion and storage: It should be able to receive and store real-time data from the sensors.

- Data processing and analytics: The platform should perform data analytics to extract meaningful insights, such as fleet utilization patterns, passenger flow, and environmental conditions.

- Visualization and reporting: The platform should provide intuitive dashboards and reports to present the analyzed data in a user-friendly manner.

- Integration: The platform should integrate with existing public transport systems, traffic management systems, and other relevant data sources to enhance the effectiveness of optimization efforts.

4. Code Implementation:

The code implementation involves the development of software components for data collection, processing, analytics, and visualization. This typically includes:

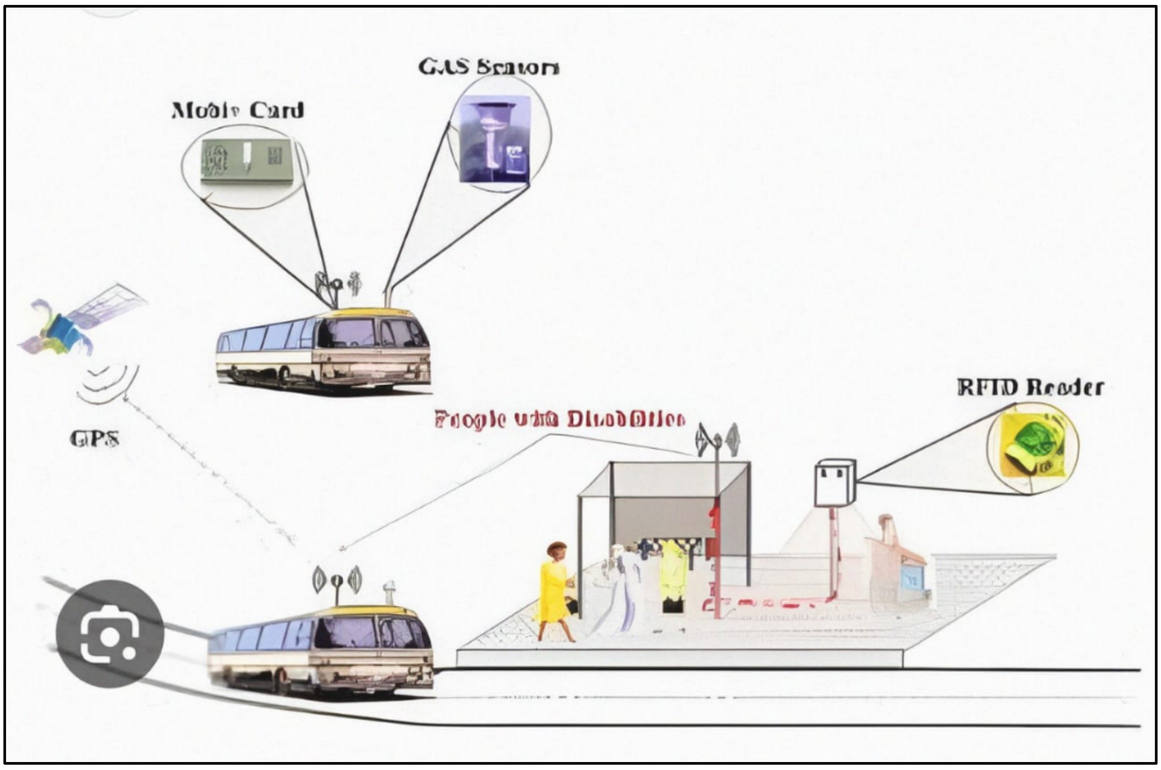
- Sensor data collection: Writing code to interface with the IoT sensors and collect data in real-time.

- Data processing: Implementing algorithms to clean, filter, and transform the raw sensor data into a suitable format for analysis.

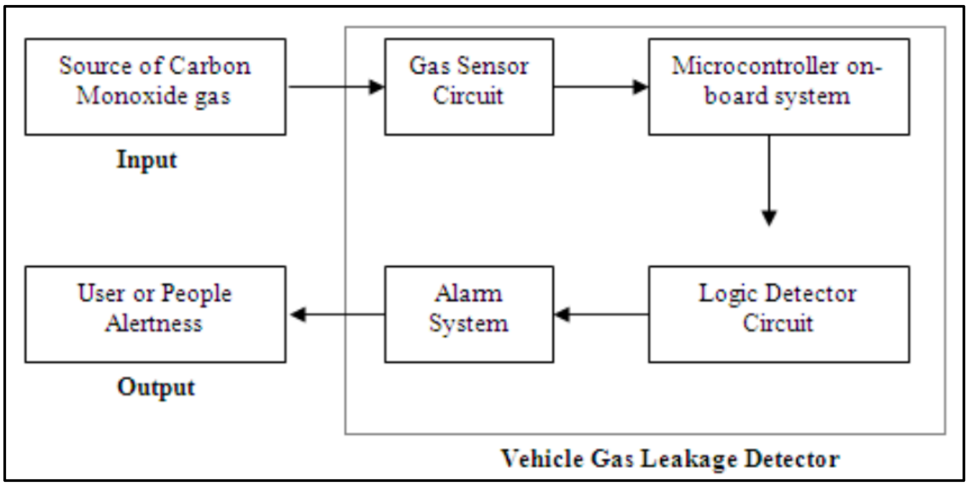
- Analytics: Developing code to perform various analyses, such as route optimization, capacity planning, predictive maintenance, and passenger behavior analysis.

- Visualization: Creating code to generate interactive dashboards, charts, and reports to present the analyzed data in a visually appealing and informative manner.

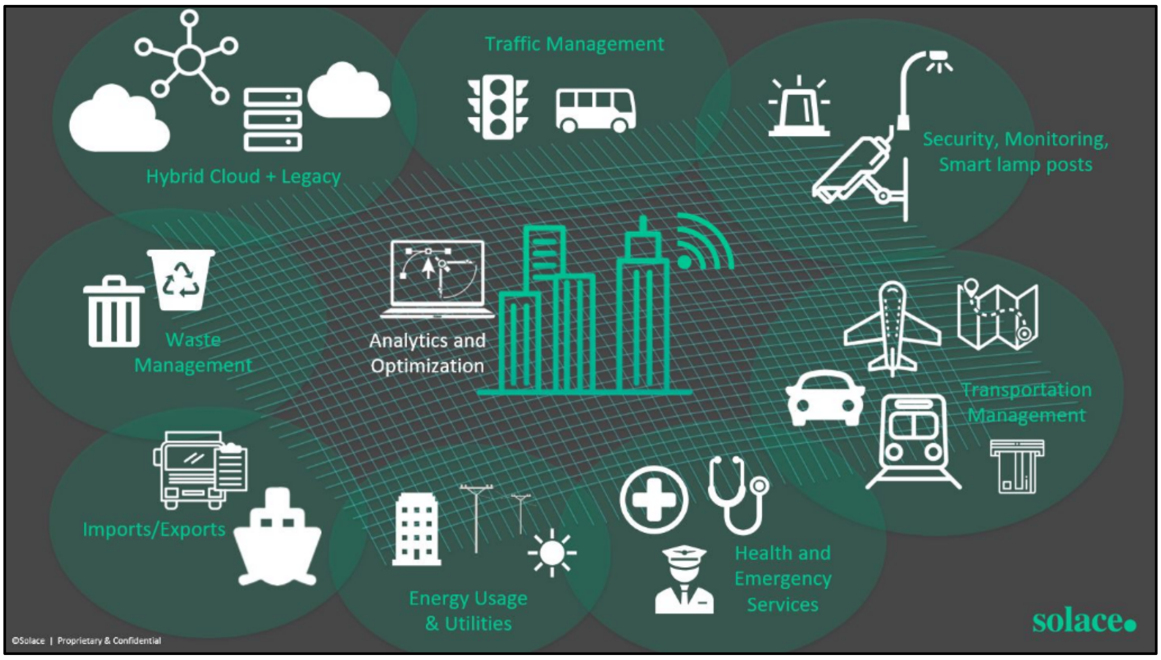
Diagram, Schematic diagram,Screenshots of IOT sensors in Public Transport Optimization



Schematics diagram


Source of Carbon Monoxide gas
Input
Gas Sensor Circuit
Microcontroller on-board system
Logic Detector Circuit
Alarm System
User or People Alertness
Output
Vehicle Gas Leakage Detector

Screenshot of IOT


Hybrid Cloud + Legacy
Traffic Management
Security, Monitoring, Smart lamp posts
Waste Management
Analytics and Optimization
Transportation Management
Imports/Exports
Energy Usage & Utilities
Health and Emergency Services
solace
©Solace | Proprietary & Confidential

A transit information platform coupled with real-time data display is an essential component of public transport optimization. Here's a description of these aspects:

1. Transit Information Platform:

A transit information platform serves as a centralized system that collects, processes, and disseminates relevant information to passengers, operators, and other stakeholders. The platform integrates various data sources, including IoT sensors, to provide comprehensive and up-to-date information about public transport services. The key features of a transit information platform include:

- Real-time updates: The platform gathers real-time data from IoT sensors, vehicle tracking systems, and other sources to provide accurate information about the current location, arrival/departure times, and delays of vehicles.

- Trip planning: Passengers can access the platform to plan their journeys, check routes, find the nearest stops/stations, and obtain estimated travel times. The platform can suggest alternative routes in case of disruptions or congestion.

- Service alerts and notifications: The platform can send alerts and notifications to passengers regarding service disruptions, changes in schedules, or other relevant information via mobile apps, SMS, or email.

- Fare information: The platform can display fare details, payment options, and any special discounts or promotions available to passengers.

- Accessibility features: The platform should consider accessibility requirements, providing information for passengers with disabilities, such as wheelchair-accessible routes and facilities.

2. Real-time Data Display:

Real-time data display refers to the visual presentation of real-time information to passengers at various touchpoints, such as stations, bus stops, and on-board displays. The objective is to keep passengers informed and improve their overall experience. Some aspects of real-time data display

Submission

GitHup Respository   Link: https://github.com/kanagaveera

Instruction

1. Hardware Setup:

   - Choose the IoT sensors suitable for your transit information project. This can include GPS sensors, accelerometers, temperature sensors, etc.

   - Connect the IoT sensors to a microcontroller or development board capable of collecting sensor data and transmitting it over a network. Examples of popular boards include Arduino, Raspberry Pi, or ESP32.

2. IoT Sensor Data Transmission:

   - Write firmware or software code for the microcontroller to read data from the connected sensors.

   - Establish a communication protocol (e.g., MQTT, HTTP, or custom protocols) for transmitting the sensor data to the cloud or a server.

   - Implement the code on the microcontroller to send sensor data to the designated endpoint using the chosen communication protocol.

3. Cloud/Server Setup:

   - Set up a cloud platform or server to receive and process the incoming sensor data. You can use platforms like AWS IoT, Azure IoT, or set up your own server using frameworks like Flask or Django.

   - Configure the cloud platform to handle incoming data from the IoT sensors and store it in a database or process it in real-time.

4. Transit Information Platform Development:

- Choose a web framework like Flask, Django, or a front-end framework like React, Vue.js to develop the transit information platform.

- Design and implement the user interface (UI) for the platform. This can include features like real-time data visualization, maps, search functionality, etc.

- Integrate the platform with the database or real-time data processing components to fetch and display the relevant transit information.

5. Integration using Python:

- Write Python scripts or modules to interact with the cloud/server APIs and retrieve the IoT sensor data.

- Process the sensor data as required and integrate it into the transit information platform.

- Implement functions or APIs to fetch real-time data updates and display them on the platform.

```
python

import math


class BusStop:

    def __init__(self, name, latitude, longitude):

        self.name = name

        self.latitude = latitude

        self.longitude = longitude


    def distance_to(self, other_stop):

        # Calculate the distance between two bus stops using the Haversine formula

        earth_radius = 6371   # Earth's radius in kilometers

        lat1, lon1 = math.radians(self.latitude), math.radians(self.longitude)

        lat2, lon2 = math.radians(other_stop.latitude), math.radians(other_stop.longitude)

        dlat = lat2 - lat1

        dlon = lon2 - lon1

        a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2) ** 2

        c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
        distance = earth_radius * c

        return distance


def optimize_bus_route(bus_stops):

    # Nearest Neighbor algorithm for optimizing bus routes

    optimized_route = []

    remaining_stops = bus_stops.copy()

    current_stop = remaining_stops.pop(0)   # Start from the first bus stop

    optimized_route.append(current_stop)


    while remaining_stops:

        nearest_stop = min(remaining_stops, key=lambda stop: current_stop.distance_to(stop))

        optimized_route.append(nearest_stop)

        current_stop = nearest_stop

        remaining_stops.remove(nearest_stop)


    return optimized_route


# Example usage

bus_stops = [

    BusStop("A", 51.5074, -0.1278),   # London

    BusStop("B", 48.8566, 2.3522),     # Paris

    BusStop("C", 55.7558, 37.6176),   # Moscow

    BusStop("D", 40.7128, -74.0060)   # New York

]


optimized_route = optimize_bus_route(bus_stops)


# Print the optimized route
```

```
for stop in optimized_route:
    print(stop.name)
```

In this example, we define a `BusStop` class to represent each bus stop with its name, latitude, and longitude. The `distance_to` method calculates the distance between two bus stops using the Haversine formula.

The `optimize_bus_route` function takes a list of bus stops as input and applies the Nearest Neighbor algorithm to optimize the bus route. It iteratively selects the nearest remaining bus stop from the current stop and adds it to the optimized route until all stops are visited.

Finally, we create a list of bus stops and call the `optimize_bus_route` function to obtain the optimized route. The program then prints the names of the bus stops in the optimized route.

Note that this is a simplified example, and in practice, you may need to consider additional that such as traffic conditions, passenger demand, or time constraints when optimizing public transport routes.