RoadSense - Requirements and Specifications

Overview

The **RoadSense** project aims to develop an IoT-based system to detect and map road anomalies such as bumpiness and potholes. By installing sensor nodes on multiple vehicles, the system collects and analyzes road vibration data to create a detailed, interactive heatmap of road conditions. This information is invaluable for road maintenance planning, improving driver safety, and providing real-time alerts for hazardous conditions.

Team Members

- **Luca Di Bello**

- **Georg Mayer**
- **Paolo Deidda**

Objectives

- **Develop an Arduino-based IoT device** capable of detecting road vibrations and recording positional data.
- **Implement noise reduction algorithms** to account for different vehicle baselines and reduce data inaccuracies.
- **Enable data transmission** from the IoT device to a centralized server via Wi-Fi when in range of dedicated hotspots.
- **Create a centralized server** to collect, aggregate, and analyze data from multiple devices.
- **Visualize the data** by overlaying a heatmap on a map to display road bumpiness levels.
- **Enhance data precision** by increasing the number of participating vehicles.
- **Distinguish between different types of road anomalies** such as speed bumps, manholes, road markings, and potholes.

Requirements

- **Up-to-Date Road Map**: Create an interactive map with detailed information on road bumpiness and hazards.
- **Visualization**: Display road conditions through a heatmap overlay, highlighting areas with significant anomalies.
- **Interactivity**: Allow stakeholders to engage with the map, view alerts, and mark issues as resolved.
- **Anomaly Differentiation**: Distinguish between various road features like speed bumps, potholes, and manholes.
- **Data Collection**: Rely on multiple vehicles for comprehensive data and enhanced accuracy.
- **Scalability**: Design a cost-effective solution suitable for widespread adoption.
- **Centralized Data Management**: Use a server to collect, aggregate, and analyze data from all devices.
- **Optimized Data Transmission**: Ensure efficient communication between IoT devices and the server.
- **Noise Reduction**: Implement algorithms to minimize inaccuracies due to different vehicle characteristics.
- **Intermittent Connectivity**: Transmit data via Wi-Fi when in range, storing data locally when not connected.
- **Power Supply**:
  - **Primary**: Utilize the vehicle's power source.
  - **Backup**: Include a battery to maintain operation when the vehicle is off.

- **Durable Casing**: Securely enclose all components, protect against external elements, and include status LEDs.

System design

System components

- **Sensor Nodes**: IoT devices installed in vehicles, responsible for collecting vibration data using an Inertial Measurement Unit (IMU) sensor and location data via a GPS module.
- **Actuator Nodes**: Handle data transmission to the server and manage device power.
- **Server-Side Application**: Collects, aggregates, and analyzes data from multiple devices, and visualizes road quality using heatmaps.
- **Control Logic**: Defines the behavior of the IoT device in terms of data collection, processing, and communication.
- **User Interface**: An interactive web application allowing stakeholders to visualize road conditions and manage alerts.

Sensor Nodes

Requirements

1. **Cost Restriction per node**: XXX CHF
   - To keep cost of installation and parts low, one single node/sensor package will be installed in the drivers cabine.
2. **Quantify felt RoadState for Driver**:
   - Node has to be close to the driver and mounted securely to the chassis to minimize errors.
   - Roadstate will be quantified in a range of 0 (very good) to 14 (very bad) with a value of 15 for hazardous condition.
   - The road state is assigned for 3m of road at a time (reduce communication).
3. **Adapt Quantification to different cars and driving states**:
   - A simple linear Mass-Spring-Damper Model is chosen to model the cars factor on the transduced shocks. (While keeping computational effort low.)
   - A first calibration phase coupled to a initial parameter set aims to fit Mass-Spring-Damper Model parameters.
   - Measured data will be fit to quantified values during calibration phase.
   - Further physical quatities other than z-axis acceleration have to be considered to decouple driving induced accelerations from the road state.
4. **Sensing of physical quantities**:
   1. **Acceleration in z-Axis** to determine road state and potholes. (Adapt pollingrate to vehicle velocity/ must be high enough)

2. **Acceleration in x,y-Axis and rotational acceleration** to minimize errors induced from driving scenarios.
3. **Driving Velocity** to coupple shock amplitudes to velocity (through Spring-Damper Model).
4. **Geographical Position** to reference qualification to current position.
5. **Driving Direction** to deduce road lane (use gps data).

5. **Transmit Data at established Gatepoints**
   1. Transmitted information Format:
      - (Node ID (2 Byte)) | Position (2 * 4 Byte (SP)) | RoadState (0.5 Byte)
   2. Preprocess and save (Position, Quality)-Tuples locally on Node
   3. Only save and transmit date every 3 meters
   4. Automatically establish connection at gatepoints and transmit new gathered data

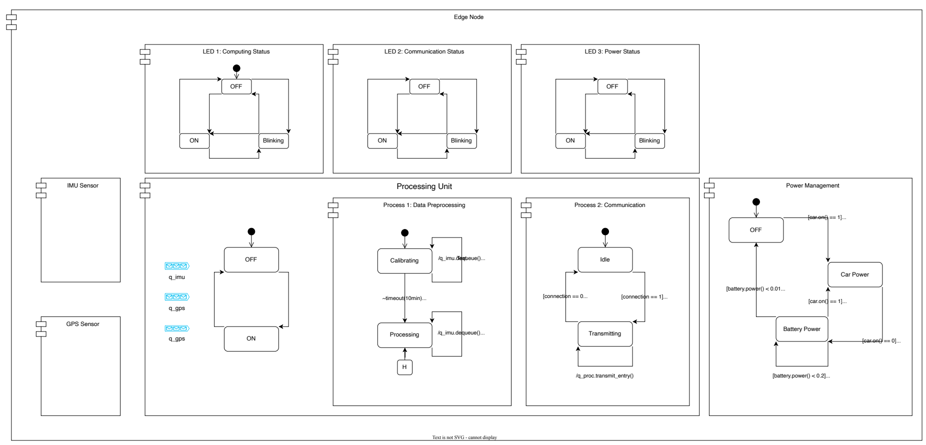

Figure 1: RT-UML

RT-UML with State Charts

Hardware Components

1. **Microcontroller**:
   - **Arduino Nano 33 IoT** or similar with built-in Wi-Fi capability.
2. **IMU Sensor**:
   - **MPU-6050** or **MPU-9250** accelerometer and gyroscope module.
3. **GPS Module**:
   - **Ublox NEO-6M** GPS module for accurate positioning.
4. **Power Supply**:
   - **Primary**: Connected to the vehicle's power supply.
   - **Backup**: Rechargeable Li-Po battery with voltage regulation.
5. **Enclosure**:

    - Durable casing with LEDs for status indication (e.g., transmission activity, errors).
6. **Connectivity**:
    - Wi-Fi module (if not integrated) like **ESP8266** or **ESP32**.

Firmware Development

- **Sensor Calibration**:
    - Implement routines to calibrate the IMU sensor for accurate readings.
- **Data Sampling**:
    - Sample sensor data at appropriate intervals (e.g., 50 Hz).
- **Noise Reduction**:
    - Apply Kalman Filter or Complementary Filter to fuse sensor data.
    - Use moving averages or median filters to smooth out transient spikes.
- **Baseline Adjustment**:
    - Establish a baseline for the vehicle's normal vibrations.
    - Adjust subsequent readings by subtracting the baseline values.
- **Wi-Fi Connectivity**:
    - Configure to connect to known Wi-Fi networks.
    - Implement setup mode for inputting Wi-Fi credentials.
- **Data Packaging**:
    - Format data (timestamp, GPS coordinates, vibration metrics) for transmission.
- **Data Transmission**:
    - Use HTTP/HTTPS protocols to send data to the server.
    - Implement error handling and retries for network issues.
- **Power Management**:
    - Monitor power source and switch between vehicle power and backup battery as needed.
    - Implement sleep modes when the vehicle is not in motion.

Data Processing and Noise Reduction

- **Calibration Period**:
    - Collect initial data to establish the vehicle's baseline vibration patterns.
    - Dynamically adjust the baseline to account for changes (e.g., vehicle load).
- **Filtering Techniques**:
    - **Low-Pass Filter**: Remove high-frequency noise unrelated to road conditions.
    - **High-Pass Filter**: Eliminate low-frequency movements like vehicle tilts.
    - **Band-Pass Filter**: Focus on frequencies corresponding to road-induced vibrations.
- **Statistical Methods**:

- **Standard Deviation and Variance**: Measure dispersion of vibration data.
    - **Peak Detection**: Identify significant deviations indicating bumps or potholes.
    - **Thresholding**: Categorize road conditions based on vibration intensity thresholds.
- **Edge Computing**:
    - **Data Compression**: Reduce data size by transmitting only significant events.
    - **Event Detection**: Implement on-device logic to detect and report anomalies.
    - **Power Efficiency**: Optimize code to reduce processor load and conserve battery life.

Actuator Nodes

The **actuator nodes** are integrated within the sensor nodes, handling data transmission and power management:

- **Data Transmission**:
    - Manage communication protocols and ensure secure data transfer to the server.
    - Queue data for transmission when connectivity is unavailable.
- **Power Management**:
    - Switch between vehicle power and backup battery seamlessly.
    - Monitor battery levels and optimize power consumption.

System Architecture

The **RoadSense system** consists of multiple IoT devices installed in vehicles, communicating with a central server designed to be highly scalable to handle data from thousands of devices.

IoT Data Pipeline

1. **Data Acquisition**: Sensor nodes collect vibration and positional data using IMU and GPS modules.
2. **On-Device Processing**: Apply noise reduction and adjust for the vehicle's baseline bumpiness using algorithms like Kalman filters.
3. **Data Transmission**: Processed data is sent to the centralized server via Wi-Fi when in range of dedicated hotspots.
4. **Data Ingestion**: The server receives data through a scalable, high-throughput data pipeline.
5. **Data Aggregation and Analysis**: The server aggregates data from multiple devices, applying further filtering and analysis.
6. **Data Storage**: An optimized database stores raw and processed data for efficient retrieval.

7. **Visualization**: Generate heatmaps and overlay them on maps to display road bumpiness levels.

Note: The backend is designed with scalability in mind, utilizing distributed computing and cloud services to handle the influx of data from numerous IoT devices.

User Interaction Flow

1. **Data Visualization**: Stakeholders access the web interface to view the heatmap of road conditions.
2. **Alert Management**: Users can view, acknowledge, and mark alerts as resolved.
3. **Interactive Map**: Features like zooming, panning, and filtering by date or severity enhance usability.
4. **Feedback Loop**: Stakeholders can provide feedback on detected anomalies to improve system accuracy.

Server-Side Application

API Development

- **RESTful APIs**:
  - For data ingestion from IoT devices.
  - For data retrieval by the web interface.
- **Authentication and Security**:
  - Implement token-based authentication.
  - Secure data transmission with HTTPS.
- **Rate Limiting**:
  - Prevent server overload by controlling the rate of incoming requests.

Database Design

- **Data Storage**:
  - Use scalable databases like PostgreSQL or MongoDB.
  - Define schemas for raw sensor data, processed data, and aggregated results.
- **Spatial Indexing**:
  - Utilize geospatial indexing for efficient geographical queries.
- **Optimization**:
  - Optimize queries for real-time data access and analysis.

Data Aggregation and Analysis

- **Data Processing Pipeline**:
  - Aggregate data from multiple devices.
  - Apply further filtering and anomaly detection.
- **Scalability**:

  - Design the backend to handle high volumes of data.
  - Use message queues and microservices architecture.
- **Analysis Techniques**:
  - Machine learning algorithms to improve anomaly detection.
  - Predictive analytics for road degradation.

Visualization

- **Web Interface**:
  - Develop a responsive web application for data visualization.
  - Integrate mapping APIs like Google Maps or OpenStreetMap.
- **Heatmap Generation**:
  - Calculate bumpiness scores for road segments.
  - Overlay heatmaps on the map interface.
- **Interactive Features**:
  - Enable filtering by time ranges, severity levels, or specific areas.
  - Provide statistical summaries and trends.
- **User Engagement**:
  - Allow stakeholders to add comments or additional data.

Control Logic

- **Data Collection**:
  - Continuously collect IMU and GPS data when the vehicle is in motion.
- **Data Processing**:
  - Apply noise reduction and baseline adjustments in real-time.
- **Communication**:
  - Transmit data to the server when connectivity is available.
  - Implement retry mechanisms for failed transmissions.
- **Power Management**:
  - Switch between power sources as needed.
  - Enter low-power modes when idle.
- **Error Handling**:
  - Monitor system health and report errors to the server.
  - Indicate status through LEDs on the device casing.

Testing and Validation

- **Field Testing**:
  - Install devices in various vehicle types (sedans, SUVs, trucks).
  - Collect data over different road conditions (urban, rural, highways).
- **Algorithm Tuning**:
  - Adjust filtering parameters based on test results.
  - Validate bumpiness scores against known road conditions.
- **User Feedback**:

    - Gather feedback from stakeholders to improve system accuracy and usability.
- **Simulation**:
    - Use simulated data to test the system under various scenarios.

Possible Problems and Solutions

Network Connectivity

- **Wi-Fi Availability**:
    - **Challenge**: Continuous Wi-Fi connectivity may not be available during travel.
    - **Solution**: Store data locally and transmit when connectivity is available.
    - **Alternative**: Use mobile hotspots or integrate GSM modules for cellular data.
- **Data Security**:
    - Encrypt data transmissions to prevent interception or tampering.
- **Error Handling**:
    - Implement robust error handling for network disruptions.

Vehicle Variability

- **Machine Learning Models**:
    - Train models to adjust for different vehicle characteristics.
- **User Input**:
    - Allow users to specify vehicle type during device setup for better calibration.
- **Adaptive Algorithms**:
    - Continuously learn and adapt to the vehicle's behavior over time.

Power Management

- **Sleep Modes**:
    - Implement low-power modes when the vehicle is stationary.
- **Efficient Components**:
    - Use low-power sensors and microcontrollers.
- **Power Monitoring**:
    - Provide alerts when battery levels are low.

Data Volume Management

- **Data Sampling**:
    - Optimize sampling rates to balance data quality and volume.
- **Selective Reporting**:
    - Transmit only aggregated or significant data points.
- **Data Compression**:

– Compress data before transmission to reduce bandwidth usage.