GNANAMANI COLLEGE OF TECHNOLOGY(Pachal,Namakkal)

DEPARTMENT: BIO MEDICAL ENGINEERING

YEAR: THIRD YEAR

PROJECTNAME- Noise pollution monitoring

Team members :K.Thirisha

C.Swathi

K.Nivetha

J.Sathya

D.Vijayalakshmi

BY,

J.sathya

NOISE POLLUTION MONITORING

Problem Statement:

Create a noise pollution monitoring system using IoT and Arduino to measure and analyze noise levels in a specific area.

Solution:

Hardware Setup:

- Use an Arduino board (e.g., Arduino Uno) as the central controller.
- Connect a sound sensor (e.g., a microphone sensor) to the Arduino to capture ambient noise levels.
- Add a Wi-Fi or Ethernet shield/module to enable internet connectivity.

Data Collection:

- Program the Arduino to continuously read data from the sound sensor.
- Convert the analog data into decibel (dB) values to measure noise levels.
- Collect timestamped noise data at regular intervals.

Data Transmission:

- Send the collected noise data to a cloud server or a web-based platform using Wi-Fi or Ethernet connectivity.
- Consider using MQTT or HTTP protocols for Data transmission.

Data Storage and Analysis:

- Store the received data in a database for historical analysis.
- Implement algorithms to calculate noise averages, peak levels, and trends over time.
- Apply noise threshold limits to identify noise pollution incidents.

User Interface:

- Create a web or mobile application to display real-time noise data and historical trends.
- Provide visualizations and alerts for noise pollution events.

Alerts and Notifications:

- Set up alert mechanisms to notify authorities or users when noise levels exceed predefined thresholds.
- Send notifications via email, SMS, or push notifications.

Power Management:

Implement power-saving techniques to prolong the Arduino's battery life if using battery power.

Maintenance:

- Regularly calibrate and maintain the sensors to ensure accurate measurements.
- Monitor the health of the Arduino and connectivity to address any issues promptly.

Data Privacy and security:

- Implement encryption and authentication measures to protect the data being transmitted and stored.

Regulatory Compliance:

Ensure that your noise monitoring system complies with local noise pollution regulations and standards.

Data Visualization and Reporting:

Generate reports and visualizations for analysis and decision-making by stakeholders and authorities.

Scalability:

- Design the system to be scalable, allowing for the addition of more sensors in different locations if needed.
- Remember that this is a simplified overview, and the actual implementation may vary depending on your specific requirements and constraints. Additionally, consider using low- power components and optimizing data transmission to make the system more energy-efficient and cost-effective.

INNOVATIONS

- Innovation in noise pollution monitoring has been advancing with development of technology. Some key innovations include
- IoT Sensors: Internet of Things (IoT) devices and sensors can be deployed throughout urban areas to continuously monitor noise levels. These sensors can transmit data in real-time to centralized systems for analysis
- Noise Mapping: Advanced mapping software can create real-time noise maps of cities, helping authorities identify noisy areas and plan mitigation strategies
- Noise Apps: Smartphone apps allow citizens to report noise complaints and collect data, contributing to crowd-sourced noise monitoring efforts
- Machine Learning: AI and machine learning algorithms can process vast amounts of noise data to identify patterns and sources of noise pollution more efficiently
- Acoustic Cameras: These cameras can visualize noise sources in real-time, providing a clear picture of where noise pollution originates.
- Noise-Cancelling Technologies: Innovations in noise-cancelling technology can help reduce noise pollution in specific environments,

Development Part 1

- Development of create a noise pollution monitoring system using IoT and Arduino to measure and analyze noise level in a specific area

- Creating a noise pollution monitoring system using IoT and Arduino involves several steps:

Components Needed:

- Arduino board (e.g., Arduino Uno or Arduino Mega)

- Sound sensor (e.g., a microphone or sound level sensor)

- IoT module (e.g., ESP8266 or ESP32 for Wi-Fi connectivity)

- Power source (e.g., batteries or a power adapter)

- Internet connection (Wi-Fi or cellular)Data storage and visualization

- platform (e.g., cloud service like AWS or Azure)

Enclosure and casing for outdoor use (if necessary)

Hardware Setup:

Connect the sound sensor to the Arduino.Connect the Arduino to the IoT module for data transmission.Ensure proper power supply and consider weatherproofing if used outdoors.

Programming:

Write Arduino code to read data from the sound sensor and send it to the IoT module.Program the IoT module to establish an internet connection and transmit the data to a cloud server.

Cloud-Based Data Storage:

Set up a cloud-based database to store the noise level data.

Data Visualization:

Use a dashboard or web application to visualize the noise data.Implement data analysis to track noise trends and trigger alerts when noise levels exceed predefined thresholds

Alerting Mechanism:

Implement notifications or alerts through email, SMS, or other means when noise levels exceed acceptable limits.

Power Management:

Optimize power usage to ensure the system can run for an extended period, especially in remote or outdoor locations.

User Interface:

Create a user-friendly interface for users to access and analyze noise data.

Calibration and Testing:

Calibrate the system to ensure accurate noise measurements.Thoroughly test the system in real-world conditions.

Data Analysis and Reporting:

Analyze the collected data to identify noise patterns and trends.Generate reports or visualizations for stakeholders.

Maintenance and Updates:

Regularly maintain and update the system to ensure its reliability and accuracy.Keep in mind that you'll need a good understanding of Arduino programming, IoT, data handling, and possibly cloud services to create a robust noise pollution monitoring system. Additionally, consider any legal and ethical considerations, such as data privacy and regulations related to noise pollution monitoring in your specific area.

FEATURE ENGINEERING OF NPM :

Feature engineering for noise pollution involves creating relevant input variables for machine learning models to better understand and predict noise pollution levels. Here are some feature engineering ideas for noise pollution:

Temporal Features:

Time of day: Splitting the day into time periods (morning, afternoon, evening, night) and representing them as categorical variables.

Day of the week:

Encode weekdays and weekends separately as binary variables.

Seasonality:

Incorporate seasonal variations as binary flags or numeric values.

Geospatial Features:

Location:

Use geographic coordinates (latitude and longitude) or divide the area into zones or regions.

Proximity to noise sources: Calculate the distance to major noise sources like highways, airports, industrial zones, or train tracks.

Land use: Categorize land use types (residential, commercial, industrial) in the vicinity.

Environmental Factors:

Weather conditions: Include weather data (e.g., wind speed, temperature) which can affect noise propagation.Air quality: Pollution levels may be correlated with noise pollution.Demographic and Socioeconomic

Factors:

Population density:

The number of people living in an area can impact noise levels.

Income levels:

Income data can help identify areas with more or less noise insulation.

Acoustic Features:

Audio data:

If available, extract acoustic features from audio recordings, such as spectral features, loudness, or dominant frequency.

Training a model for noise pollution monitoring typically involves the following steps:

Data Collection:

Gather a dataset of audio recordings along with corresponding noise level measurements. These recordings should cover various types of noise sources and environments.

Data Preprocessing:

Clean and preprocess the audio data, which may include noise reduction, feature extraction (e.g., spectrograms or Mel-frequency cepstral coefficients), and segmentation.

Labeling:

Annotate the data with noise level labels, indicating the noise intensity or class (e.g., quiet, moderate, loud).

Model Selection:

Choose an appropriate machine learning model, such as a convolutional neural network (CNN), recurrent neural network (RNN), or a combination of both.

Model Architecture:

Design the model architecture, including the input layer, hidden layers, and output layer. Consider using transfer learning if applicable.

Training:

Split the data into training, validation, and test sets. Train the model on the training data, and use the validation set to fine-tune hyperparameters and monitor for overfitting.

Evaluation:

Assess the model's performance using appropriate metrics like mean squared error (MSE), accuracy, or other noise-related metrics. Evaluate the model on the test set.