BME68X Project Documentation

Version 1.1
Date: YYYY-MM-DD

Table of Contents

Introduction

The **BME68X Project** is a comprehensive solution designed for environmental sensing using the BME68X sensor. The project integrates embedded firmware (written in C++) for sensor configuration, data collection, and real-time control with a Python-based data processing and machine learning system for odor classification.

The primary goal was to achieve a reliable classification of different odors. Although initial trials with alternative approaches (labeled as "BSEC" and various classification versions) were explored to differentiate sensor responses based on concentration levels, these attempts did not yield reliable differentiation for concentrations. Instead, the system successfully distinguishes between different odors.

System Overview and Architecture

The project comprises two major components:

- **Firmware (Embedded C++ - BME68X):**
  The firmware is responsible for sensor initialization, dynamic configuration via an SD card, applying heater profiles, adjusting data intervals, and handling various sensor commands. The trial code under the BSEC directory was explored to differentiate concentration levels, but the final solution focuses solely on odor classification.

- **Data Processing and GUI (Python):**
  Python modules manage data collection, feature extraction, and classification. The primary model employed is a RandomForest classifier from scikit-learn, which has been tuned for odor classification based on the sensor data.

A high-level system architecture diagram is shown below:

 System Architecture
Insert your system architecture diagram here.

Firmware Code Overview

Sensor Initialization and Error Handling

The firmware (located in the BME68X directory) initializes multiple BME68X sensors and includes robust error handling routines. Key functionalities include:

- **Error Reporting:**
  The function `getBmeErrorMessage()` returns descriptive error messages corresponding to different error codes. Visual indicators such as LED blinking are used to signal warnings or errors.

- **Sensor Status Report:**
  The firmware cycles through each sensor to verify their operational status and prints diagnostic messages while triggering LED alerts for detected issues.

Dynamic Configuration, Heater Profiles, and Command Handling

The firmware dynamically loads configuration data from an SD card. If the SD card is unavailable or the configuration file is empty, it falls back to hardcoded configurations.

Heater Profiles

- **Heater Profile Application:**
  The firmware supports multiple heater profiles to control the sensor's operation. Functions like `setHeaterProfile()` assign specific heater configurations, while button presses allow for cycling between profiles.

Sensor Commands and Button Actions

The system supports a variety of commands sent via the serial interface and button presses on the hardware. Each command and button action is designed to control and monitor the sensor's behavior:

- **Serial Commands:**

    - `START`: Begins data collection.
    - `STOP`: Ends data collection.
    - `SEC_<number>`: Adjusts the data interval (e.g., `SEC_3000` sets the interval to 3000 milliseconds).
    - `GETHEAT`: Requests the current heater profile configuration from the sensor.
    - `START_CONFIG_UPLOAD`: Initiates the process to upload a new JSON configuration via the serial interface.
    - `STATUS_REPORT`: Triggers a full sensor status report.

- **Button Press Actions:**

    - **Single Button Press:**
      When one button is pressed, it increments or decrements a counter (used for tracking or as a label tag).
    - **Simultaneous Button Press:**
      Pressing both buttons together triggers the cycling of heater profile assignments across sensors.

These commands and actions allow users to monitor and control the sensor behavior effectively, including adjusting heater profiles and data intervals in real time.

Data Handling and Classification

Data Collection and Processing Pipeline

Python scripts in the BME68X Data Handler module manage the acquisition and processing of sensor data:

- **Data Acquisition:**
  Raw sensor data is logged in CSV format, including measurements such as temperature, pressure, humidity, and gas resistance.

- **Feature Extraction:**
  The `DataProcessor` class uses a sliding window approach to compute statistical features (mean, standard deviation, minimum, and maximum) from the sensor data, preparing the data for classification.

Model Training Using RandomForest (scikit-learn)

The primary model used for odor classification is a RandomForest classifier from scikit-learn. The training process involves:

- **Preprocessing:**
  Reading the raw data and processing it to extract relevant features.

- **Label Encoding:**
  Converting sensor labels into numerical values suitable for training.

- **Model Training:**
  Training the RandomForest classifier with balanced class weights to account for any data imbalances.

- **Metrics Generation:**
  Generating performance metrics such as accuracy, confusion matrix, and classification reports, which are saved in a JSON file (`model_metrics.json`).

Graphical User Interfaces

Model Trainer & Predictor GUI

A dedicated GUI (developed with Tkinter) provides the following functionalities:

- **File Selection:**
  Users can browse and select raw data files and processed feature files.
- **Feature Extraction and Model Training:**
  The GUI includes controls for setting the window size and stride for feature extraction. It shows training progress and status updates.
- **Model Prediction:**
  After training the model (using the RandomForest classifier), the GUI allows users to run batch predictions and view model metrics.

Placeholder for Trainer GUI screenshot:
Model Trainer GUI
Insert screenshot of the Model Trainer GUI here.

Data Logger GUI

The Data Logger GUI enables real-time monitoring and logging of sensor data. Its key features include:

- **Serial Connection Management:**
  It provides controls to select and connect to available serial ports.
- **Real-Time Data Display and Plotting:**
  The GUI displays live sensor data in a scrolling text area and updates sensor data plots dynamically.
- **Command Controls:**
  Buttons on the GUI allow sending of serial commands (such as `START`, `STOP`, and `GETHEAT`) to the firmware. It also reflects button actions and sensor command responses.
- **Interactive Plotting:**
  Uses Matplotlib to plot sensor measurements over time, with options to adjust the displayed time window and select parameters.

Placeholder for Data Logger GUI screenshot:
Data Logger GUI
Insert screenshot of the Data Logger GUI here.

Results

- **Odor Classification:**
  The system is capable of reliably classifying different odors based on sensor data using the RandomForest classifier. Although initial trials attempted to differentiate different concentration levels (using the BSEC trial code and various classification versions), these approaches did not yield the desired results. The final system distinguishes between odor types but not concentration differences.

- **Dynamic Heater Profiles:**
  The firmware successfully applies and cycles through different heater profiles, and the GUI reflects these changes in real time.

- **Sensor Commands and Controls:**
  All sensor commands (e.g., `START`, `STOP`, `SEC_<number>`, `GETHEAT`) and button actions function as intended, allowing comprehensive control over sensor operation and data logging.

Conclusion and Future Work

This project presents a full-stack solution for environmental sensing using the BME68X sensor. The embedded firmware efficiently manages sensor operation, while the Python-based data processing system performs feature extraction and odor classification using a RandomForest classifier. Although the system did not achieve reliable concentration differentiation, it successfully distinguishes between various odors.

Future Enhancements:

- **Advanced Concentration Detection:**
  Further research may focus on refining sensor calibration and signal processing techniques to differentiate concentration levels.
- **Extended Machine Learning Models:**
  Exploring deep learning models could provide improved classification performance.
- **Cloud Integration:**
  Integrating cloud services for remote monitoring and data analysis.
- **Enhanced User Interfaces:**
  Improving the GUI interactivity and adding additional customization options for real-time data visualization and control.

Appendices

Appendix A: Directory Structure

```
BME688-PROJECT
├── BME68X
│   ├── BME688_CPP_Code
│   │   ├── .pio
│   │   ├── config
│   │   │   ├── config.json
│   │   ├── include
│   │   │   ├── main.h
│   │   │   ├── commMux
```

```
|   |   |-- lib
|   |   |-- logs
|   |   |-- src
|   |   |   |-- main.cpp
|   |   |-- platformio.ini
|   |   |-- BME688_Data_Handler
|   |   |   |-- Bulk Training Data
|   |   |   |-- DataClassification
|   |   |   |   |-- data_classifier_model.joblib
|   |   |   |   |-- dataClassification.py
|   |   |   |   |-- dataProcessor.py
|   |   |   |   |-- model_metrics.json
|   |   |   |-- DataClassification_Versions
|   |   |   |   |-- DataClassification_linearsvc
|   |   |   |   |-- DataClassification_MLP
|   |   |   |   |-- DataClassification_RandomForest
|   |   |   |   |-- DataClassification_SGDClassifier
|   |   |   |   |-- DataClassification_XgBoost
|   |   |   |-- DataCollection
|   |   |   |   |-- DataCollection.py
|   |   |   |-- Testing_CSV
|   |   |   |-- Training_Data
|   |   |   |-- Label_Encoder.csv
|-- (BSEC folder omitted - trial code for concentration differentiation)
|-- Screenshots
|-- Report.pdf
```

Appendix B: Command and Button Action Summary

- **Serial Commands:**
  - `START`: Initiates sensor data collection.
  - `STOP`: Stops sensor data collection.
  - `SEC_<number>`: Sets the data interval (e.g., `SEC_3000` sets a 3000 ms interval).
  - `GETHEAT`: Retrieves and displays the current heater profile configuration.
  - `START_CONFIG_UPLOAD`: Begins the process to upload a new configuration file via serial.
  - `STATUS_REPORT`: Generates and prints a complete sensor status report.
- **Button Press Actions:**
  - **Single Button Press:** Increments or decrements a counter value used as a label tag.
  - **Simultaneous Button Press:** Cycles through different heater profile assignments across the sensors.

End of Report
