BME688 Project Documentation

Author: Omar Elgendy

Affiliation: Carleton University

Date: March 2025

Supervisor: Dr. Adrian Chan

Table of Contents

Introduction

The **BME68X Project** is a comprehensive solution designed for environmental sensing using the BME68X sensor. It integrates embedded firmware 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 reliable classification of different odors. Although initial trials using 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 final system successfully distinguishes between different odors.

System Overview and Architecture

The project comprises two major components:

- **Firmware (Embedded C++ – BME68X):**
  Responsible for sensor initialization, dynamic configuration via an SD card, applying heater profiles, adjusting data intervals, and handling various sensor commands.
- **Data Classification and Collection (Python):**
  Python modules manage data collection, feature extraction, and classification. The primary model is a RandomForest classifier (from scikit-learn), tuned for odor classification based on sensor data.

Firmware Code Overview

Sensor Initialization and Error Handling

The firmware initializes multiple BME68X sensors and includes robust error handling routines. Key functionalities include:

- **Error Reporting:**
  The `reportBmeStatus(Bme68x &sensor, uint8_t sensorIndex)` function assesses each sensor's operational status and provides descriptive error messages for various fault conditions. LED indicators differentiate between critical errors and non-critical warnings, ensuring clear and immediate visual feedback.
- **Sensor Status Report:**
  The firmware cycles through each sensor, verifies operational status, and prints diagnostic messages while triggering LED alerts for detected issues.

Heater Profiles

- **Dynamic Configuration and Heater Profiles:**
  The firmware dynamically loads configuration data from an SD card and falls back to hardcoded configurations if the SD card is unavailable or the configuration file is empty.
- **Heater Profile Application:**
  Functions such as `setHeaterProfile()` configure specific heater settings. Profiles can be loaded via the serial monitor or from the SD card (requiring the rest of the board to be active).

Sensor Commands and Button Actions

The system supports a variety of commands sent via the serial interface and hardware button presses:

- **Serial Commands:**
  - **START:** Begins data collection.

  - **`STOP`:** Ends data collection.
  - **`MS_<number>`:** Adjusts the data interval (e.g., `MS_3000` sets the interval to 3000 milliseconds).
  - **`GETHEAT`:** Requests the current heater profile and duty cycle configuration.
  - **`START_CONFIG_UPLOAD`:** Initiates the process to upload a new JSON configuration.
  - **`END_CONFIG_UPLOAD`:** Marks the end of the configuration upload process.
  - **`STATUS_REPORT`:** Triggers a full sensor status report.
- **Button Press Actions:**
  - **Single Button Press:**
    Increments (Button 1) or decrements (Button 2) a counter used for tracking the current odor label.
  - **Simultaneous Button Press:**
    Triggers the cycling of heater profile assignments across sensors.

Data Handling and Classification

Data Acquisition and Processing Pipeline

This section outlines the roles of various modules in the BME68X system:

Firmware Data Acquisition

- The firmware collects real-time sensor readings (temperature, pressure, humidity, and gas resistance) and transmits them for further processing.
- It initializes sensors at startup, applies heater profiles, and loops until a command (such as `START`) is received.

Data Collection Module (Python)

- Establishes a serial connection via the GUI.
- Initiates data logging through the Data Logger GUI, allowing users to specify file names and storage locations.
- Monitors system activity through a data transfer window and provides options to visualize real-time sensor data.

Data Processing Module

- Extracts key statistical features (mean, standard deviation, minimum, and maximum) from sensor readings.
- Segments data into overlapping time windows to capture short-term variations.
- Uses a configurable stride length for efficient feature extraction and dynamic mapping of sensor data.

Model Training Using RandomForest

The classification module employs a RandomForest classifier to distinguish between different odors: - **Training:**
The model is trained using either raw data files or processed feature files. Users can configure window length and stride parameters. - **Prediction:**
Supports two modes:
- **Real-time Prediction:** Connects to the serial interface, sets a batch length greater than the window, and begins predictions. - **File-based Prediction:** Loads a data file and uses the **Predict** button for classification.

Graphical User Interfaces

Data Logger GUI

The Data Logger GUI provides real-time monitoring and logging of sensor data. Key features include:

- **Serial Connection Management:**
  Selection and connection to available serial ports.

- **Real-Time Data Display and Plotting:**
  Displays live sensor data in a scrolling text area and updates sensor data plots dynamically.
- **Command Controls:**
  Allows sending of serial commands (such as `START`, `STOP`, and `GETHEAT`) and reflects command responses.
- **Interactive Plotting:**
  Uses Matplotlib to plot sensor measurements over time with options to adjust the time window and select parameters.

Model Trainer & Predictor GUI

The Data Classifier GUI provides real-time prediction and model training. Key features include:

- **File Selection:**
  Browsing and selecting raw data files and processed feature datasets.
- **Feature Extraction and Model Training:**
  Controls for setting the window size and stride, with visible training progress and status updates.
- **Model Prediction:**
  Allows users to run both real-time and file-based predictions after training the RandomForest model.

Test Procedure and GUI Operation

This section details the step-by-step procedure for validating the performance of the BME68X sensor system through various test scenarios.

1. Preparation

- **Hardware Setup:**
  - **Assembly:** Ensure the BME68X sensor board is properly set up and that the SD card, containing the configuration file for dynamic configuration, is inserted.
  - **Firmware:** Confirm that the firmware (BME688_CPP_CODE) is flashed successfully and that the sensor is operational.
  - **Test Environment:** Set up a controlled environment for systematic odor introduction.
- **Software Setup:**
  - **Data Logger GUI:** Launch the GUI to monitor live sensor data.
  - **Serial Connection:** Connect the sensor to your computer and select the appropriate serial port.
  - **Model Trainer & Predictor GUI:** Open the GUI for feature extraction, model training, and predictions.

2. Running Tests

Baseline Stability Test

- **Objective:** Verify sensor stability under a constant odor.
- **Procedure:**
  - Introduce a single, constant odor.
  - Use the `Start logging` button to begin data collection.
  - Monitor the sensor output for 5–10 minutes, observing the consistency in temperature, humidity, and gas resistance as they respond to the odor, noting both the initial rise and stabilization.
  - Record any unexpected fluctuations.

Odor Transition Test

- **Objective:** Observe sensor response when transitioning between odors.

- **Procedure:**
  - Sequentially expose the sensor to different odors.
  - Monitor live data plots during each transition.
  - Note sensor response time and any abrupt or gradual changes.
  - Document the transition behavior for analysis.

Repeated Exposure Test

- **Objective:** Assess the repeatability of sensor responses.
- **Procedure:**
  - Expose the sensor repeatedly to the same odor at defined intervals.
  - Record sensor measurements for each cycle.
  - Compare data to evaluate consistency and reliability.

Control and Configuration Test

- **Objective:** Verify heater profile functionality and sensor command handling.
- **Procedure:**
  - Cycle through heater profiles using onboard buttons and observe LED indicators.
  - Send serial commands (e.g., `GETHEAT`, `SEC_<number>`, `STATUS_REPORT`) to adjust sensor settings. This can be done using the serial monitor terminal within PlatformIO.
  - Confirm that firmware responses are accurately reflected in sensor logs and GUIs.

3. Using the Graphical User Interfaces

Data Logger GUI

- **Connection:**
  Launch the GUI, select the correct serial port, and connect to the sensor to stream live data.
- **Monitoring:**
  Observe sensor data in the scrolling text area and real-time plots. Use GUI buttons to send commands such as `START`, `STOP`, and `GETHEAT`.
- **Data Logging:**
  Save or export logged data for later analysis.

Model Trainer & Predictor GUI

- **Data Loading:**
  Load raw data files or processed feature datasets using the file selection feature.
- **Feature Extraction:**
  Set the window size and stride to extract statistical features.
- **Model Training:**
  Train the RandomForest classifier and monitor progress (accuracy metrics, confusion matrix, etc.).
- **Real-Time Prediction:**
  Run predictions on live or recorded sensor data and compare predicted odor labels with expected outcomes.

4. Documentation and Analysis

- **Logging Results:**
  Maintain a detailed log that captures:
  - Sensor readings for each test phase.
  - GUI screenshots (live plots, command responses, model metrics).
  - Observations during odor stability, transitions, and repeated exposures.
- **Post-Test Analysis:**
  Analyze saved CSV data and JSON model metrics to assess sensor performance, classifier accuracy, and response times. Identify any anomalies for troubleshooting.

5. Iteration and Optimization

- **Refinement:**
  Fine-tune sensor settings or update model parameters based on test results. Re-run tests to verify improvements.
- **Future Enhancements:**
  - Explore additional machine learning models for enhanced odor classification.
  - Enhance GUI interactivity for more intuitive control and visualization.

Results

- **Odor Classification:**
  The system reliably classifies different odors using a RandomForest classifier. While initial trials aimed to differentiate concentration levels with various approaches (including BSEC trial code), these methods did not yield the desired results. The final system distinguishes odor types effectively.

- **Dynamic Heater Profiles:**
  The firmware successfully applies and cycles through different heater profiles, with real-time reflections on the GUI.

- **Sensor Commands and Controls:**
  All sensor commands (e.g., `START`, `STOP`, `GETHEAT`) and button actions operate as intended, providing 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 operations while the Python-based data processing system performs feature extraction and odor classification using a RandomForest classifier. Although reliable concentration differentiation was not achieved, the system successfully distinguishes between various odors.

Future Enhancements: - **Concentration Detection:**
Refine sensor calibration and signal processing techniques to enable concentration differentiation. - **Extended Machine Learning Models:**
Explore deep learning and other models to improve classification performance. - **Cloud Integration:**
Incorporate cloud services for remote monitoring and data analysis. - **Enhanced User Interfaces:**
Improve GUI interactivity and add more customization options for real-time visualization and control.

Appendices

Appendix A: Directory Structure

```
BME688-PROJECT
├── BME68X
│   ├── BME688_CPP_Code
│   │   ├── .pio
│   │   ├── .vscode
│   │   ├── config
│   │   ├── include
│   │   │   ├── commMux
│   │   │   └── main.h
│   │   ├── lib
│   │   ├── logs
│   │   ├── src
│   │   │   └── main.cpp
│   │   ├── test
│   │   └── platformio.ini
├── BME688_Data_Handler
│   ├── Bulk Training Data
│   ├── DataClassification
│   │   ├── data_classifier_model.joblib
│   │   ├── dataClassification.py
│   │   ├── dataProcessor.py
│   │   └── model_metrics.json
│   ├── DataCollection
│   │   ├── dataCollection.py
│   │   └── model.pkl
│   ├── Testing_CSV
│   ├── Training_Data
│   └── Label_Encoder.csv
└── Old files
```

Appendix B: Command and Button Action Summary

- **Serial Commands:**
  - **`START`**: Initiates sensor data collection.
  - **`STOP`**: Stops sensor data collection.
  - **`GETHEAT`**: Retrieves the current heater profile configuration.
  - **`GETDUTY`**: Retrieves the current duty cycle profile configuration.
  - **`START_CONFIG_UPLOAD`**: Begins the configuration file upload process via serial input.
  - **`END_CONFIG_UPLOAD`**: Ends the configuration file upload process.
  - **`STATUS_REPORT`**: Generates a complete sensor status report.
  - **`MS_<number>`**: Updates the data collection interval (e.g., `MS_5000` for 5000 ms).
- **Button Press Actions:**
  - **Single Button Press:**
    Increments (`Button 1`) or decrements (`Button 2`) the label counter used for odor classification.
  - **Simultaneous Button Press:**
    Cycles through different heater profile assignments across sensors.

Appendix C: Program Interface Screenshots