BME68X Project Documentation

Author: Omar Elgendy
Affiliation: Carleton University
Date: March 2025
Supervisor: Dr. Adrain 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:**
  The function setHeaterProfile() configures specific heater settings. Profiles can be loaded either via the serial monitor or from the SD card (which requires the rest of the board to be active). Additionally, these heater profiles can be accessed and modified using the Python collection interface for convenience.

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 a Python module.
- Initiates data logging, allowing users to specify file names and storage locations.
- Monitors system activity through data transfer routines and provides options to visualize 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.

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.

- **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 Data Visualization:**
Improve data visualization methods for more intuitive control and analysis.

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:**
    Pressing both buttons simultaneously cycles through the heater profile assignments across sensors. The assignment array's numbers represent the heater profile, while the index denotes the corresponding sensor. With each simultaneous press, the assignment rotates as follows:
    - First press: [1, 1, 2, 2, 3, 3, 0, 0]
    - Second press: [2, 2, 3, 3, 0, 0, 1, 1]
    - Third press: [3, 3, 0, 0, 1, 1, 2, 2]
    - Fourth press: [0, 0, 1, 1, 2, 2, 3, 3]

Appendix C: Program Interface Screenshots