Coding Document

Cement Mixer Monitoring System

1. Introduction

This document provides a complete technical walkthrough of the three main codebases powering the Cement Mixer Monitoring System. These include two Arduino-based ESP8266 sensor circuits and a Python Flask server script that receives and logs real-time sensor data.

All files are located in the `/Codes` folder of the GitHub repository:
- `MPU6050_ESP8266.ino`
- `SCT013_ESP8266.ino`
- `flask_server.py`

2. Code Functionality Overview

- The MPU6050 circuit measures angular velocity and linear acceleration, and transmits these via Wi-Fi to a local server.
- The SCT013 circuit measures the RMS current drawn by the mixer motor.
- The Flask server logs incoming sensor data into CSV files and manages time-stamping and data validation.

All codes contain debug prints and logic blocks for testing, which are helpful during development but do not interfere with real-time deployment.

3. Arduino Code (MPU6050 + ESP8266)

Wi-Fi Setup: Connects to a specified SSID and password. The ESP8266 blinks its internal LED during data transfer.

MPU6050 Initialization: I2C scan is performed to detect the sensor. The gyroscope range is configured to ±1000 deg/s.

Sensor Reading Loop: Raw gyroscope and accelerometer values are read. Acceleration is converted to a Euclidean norm (`accel_euclidean_sum`) representing linear acceleration.

Data Transmission: A JSON payload with 4 parameters is posted to the Flask server every 5 seconds using `HTTPClient`.

Debugging & Resilience: Wi-Fi reconnect logic is embedded in the loop. Serial logs confirm sensor connection and POST status codes.

4. Arduino Code (SCT013 + ESP8266)

Wi-Fi Setup: Same as MPU6050 setup. Manages LED blinking and reconnection.

Current Sensor Initialization: Uses the EmonLib library with a calibration constant of `30.0`. (Note: This value must be experimentally calibrated based on setup and current range).

Sensor Reading Loop: Calculates Irms current using 1480 samples per loop (appropriate for 50Hz power systems).

Data Transmission: Sends JSON data to the same Flask endpoint. The field used is `field5`.

Debugging: Serial output includes real-time current measurements and HTTP response status.

5. Flask Server Code

Routes: The server has a single endpoint `/update` that listens for POST requests.

Input Parsing: The server inspects the incoming JSON data to determine whether it's from MPU6050 or SCT013.

Logging: Based on the fields present, the server logs data into two different files (`MPU6050_Log.csv` and `SCT013_Log.csv`).

Note: Currently, these are saved to fixed Windows directory paths. Consider using relative paths for better portability.

Recommendations:
- Use a unified logging system if both data types can arrive synchronously.
- Include millisecond precision timestamps for improved temporal analysis.
- Consider securing the endpoint using token-based authentication.
- Host on cloud (e.g., Heroku/Render) if central logging is needed.

6. Coding Best Practices & Notes

Maintainability Tips:
- Place Wi-Fi credentials and IP in a separate config file (if scaling)
- Comment debug blocks clearly for easy enable/disable
- Use relative paths and modularize code for wider deployment

Deployment Considerations:
- IP addresses and Wi-Fi must be updated in the code before uploading via USB
- Flask server must run before circuits are powered
- Avoid editing CSV logs in Excel while writing
- Use LED indicators for fast troubleshooting:
  - External Red LED: Power
  - Internal Blue LED: Data Transmission

Testing & Reliability:
- The code is stable; most issues arise due to wiring or power inconsistencies
- Trial and error is required to calibrate the SCT013 constant

7. Code Reviews & Logic Overview

7.1 Arduino Code: MPU6050 + ESP8266

This code interfaces the MPU6050 IMU with an ESP8266 board to measure angular velocity and linear acceleration and transmit the data to a local Flask server over Wi-Fi.

Key Logic Segments:
- Wi-Fi Setup: Establishes connection using `WiFi.begin()` with visual feedback through an external LED. Handles reconnection logic gracefully within the loop.
- MPU6050 Initialization: Performs an I2C scan and sets up the sensor with clock source and gyroscope range. Ensures device readiness using `mpu.testConnection()`.
- **Sensor Data Acquisition: Retrieves raw gyro and accelerometer readings and converts them into degrees/sec and g units, respectively. Computes a Euclidean sum of acceleration to simplify linear motion representation.
- Data Transmission: Constructs a JSON object using `String()` and sends a POST request to the `/update` endpoint using `HTTPClient`. Includes HTTP response logging and error handling.
- Debugging: Includes serial print statements at every key step (connection, sensor read, data sent), aiding real-time diagnostics.

7.2 Arduino Code: SCT013 + ESP8266

This sketch measures the RMS current using a non-invasive SCT013 current sensor and sends the result to a Flask server over Wi-Fi.

Key Logic Segments:
- Wi-Fi Setup: Similar to the MPU6050 sketch with a dedicated LED for connection status.
- Sensor Calibration: Uses EmonLib's `current()` method with a calibration constant (`30.0`). Though functional, this constant should be tuned per deployment.
- Current Measurement: Calls `calcIrms()` over 1480 samples, optimized for 50Hz power systems. Provides accurate RMS current values for industrial monitoring.
- Data Transmission: Sends the current value via JSON to the same `/update` endpoint. Uses `field5` as a marker for the Flask server to differentiate.
- Debugging: Serial monitor outputs both current readings and HTTP statuses for network diagnostics.

7.3 Python Code: Flask Server

This Flask-based server listens for incoming POST requests from the ESP8266 devices and logs them into appropriate CSV files.

Key Logic Segments:
- Request Handling: The `/update` route parses JSON payloads and checks for keys specific to either the MPU6050 or SCT013 sensor.
- Dynamic Routing: Logic branches based on JSON keys (`gyroX_dps` for MPU6050 and `field5` for SCT013) to determine the CSV structure.

- CSV Logging: Appends new entries to their respective files. If the file doesn't exist, it creates a new DataFrame with the correct structure.
- Error Handling: All exceptions are caught and logged, with corresponding HTTP status codes returned.
- Recommendations: Should use relative paths instead of `D:/...` and consider restructuring to allow both data types in a single combined CSV file. In production, basic authentication and HTTPS should also be implemented.