# WEB MONITORING SERVER

## Short textual description of the application

This project implements a real-time monitoring web server based on Python Flask that acts as an intermediary between an STM32 Controlled Board and web browser clients.

The application periodically retrieves motor control data from the Controlled Board via HTTP requests every 100ms. The data includes motor position (actual and requested) and PWM duty cycle, sampled at 5ms intervals (20 samples per batch).

The retrieved data is cached in a thread-safe manner and served to web clients through a REST API endpoint. A frontend web page built with Chart.js displays real-time graphs showing position tracking and PWM output over a 2-second window.

This architecture allows monitoring the motor control system from any web browser without requiring direct access to the embedded board, bypassing browser CORS restrictions through the Flask proxy server.

## Application data flow

The data flow follows a producer-consumer pattern with caching for performance optimization.

A dedicated data fetcher thread runs continuously as a daemon, executing HTTP GET requests to the Controlled Board's /data endpoint every 100ms. The request has a 500ms timeout to prevent blocking. Upon successful response, the JSON payload containing 20 samples is parsed and stored in a global cache dictionary.

Access to the data cache is protected by a threading.Lock() to ensure thread safety between the fetcher thread (writer) and Flask request handlers (readers). The cache stores arrays of actual positions, requested positions, PWM values, the sample interval (dt_ms), and a timestamp of the last update.

When a browser client requests data through the /api/data/batch endpoint, the Flask handler acquires the cache lock, copies the current cache contents, and returns them as a JSON response. If the cache is empty (e.g., during startup or connection failure), an HTTP 503 error is returned.

The frontend JavaScript code uses the Fetch API to request data every 100ms. Received samples are appended to local arrays maintaining a 2-second history (400 samples at 5ms resolution). Old samples are removed in FIFO order. The Chart.js library renders two graphs: one for position (requested vs actual) and one for PWM duty cycle (normalized to ±100%).

## Description of Global Functions and Variables

This section describes the main global variables and functions implemented in the web server, including their purpose and behavior.

### Global Variables

#### controlled_board_IP (string)
IP address of the Controlled Board. Default: "192.168.1.2".

#### controlled_board_port (int)
HTTP port of the Controlled Board. Default: 80.

#### data_cache (dict)
Dictionary storing the most recent batch of samples from the Controlled Board.

Contains keys: "actual" (list of 20 floats), "requested" (list of 20 floats), "pwm" (list of 20 floats), "dt_ms" (int, sample interval), "timestamp" (float, Unix timestamp of last update). Protected by cache_lock.

#### cache_lock (threading.Lock)
Lock protecting concurrent access to data_cache.

Acquired by the data fetcher thread when writing and by Flask handlers when reading. Uses Python's context manager pattern (with statement) for automatic release.

### Data Fetcher Thread

#### data_fetcher()
Thread function implementing periodic data retrieval from the Controlled Board.

- Description:

Runs an infinite loop executing HTTP GET requests every 100ms. On success, acquires cache_lock and updates data_cache with the received JSON data. Handles Timeout, ConnectionError, and generic exceptions gracefully, logging errors without stopping the thread.

- Return value: None (infinite loop, daemon thread).

### Flask Endpoints

#### index() - Route: /
Serves the main HTML page (index.html) for the monitoring interface.

- Return value: Rendered HTML template.

#### api_data_batch() - Route: /api/data/batch
Returns the most recent batch of samples in JSON format.

- Description:

Acquires cache_lock, checks if data is available, and returns a JSON object containing status, dt_ms, actual[], requested[], pwm[], timestamp, and num_samples. Returns HTTP 503 if cache is empty.

- Return value: JSON response with status code 200 or 503.

## Frontend Functions (JavaScript)

### updateGraphs()

Async function fetching data from /api/data/batch and updating the charts.

- Description:

Called every 100ms via setInterval. Fetches JSON data, calls addSamplesToGraph to update arrays, updates current value labels, recalculates Y-axis ranges for auto-scaling, and calls chart.update() on both charts.

### addSamplesToGraph(actualArray, requestedArray, pwmArray, dtMs)

Adds received samples to the display buffers and manages buffer size.

- Parameters: arrays of position and PWM values, sample interval.
- Description:

Iterates through samples, generates time labels, appends to dataActual, dataRequested, and dataPwm arrays (normalizing PWM to percentage). Removes oldest samples when buffer exceeds MAX_SAMPLES (400).