Technical Project Abstract

Medical Device Monitoring System — Embedded Linux + Flutter + OpenCV

Prepared for Antigravity · February 2026

Project Overview

This project is a self-contained simulation of a medical device monitoring platform, built from scratch to demonstrate practical knowledge of the technology stack used in modern IVD (in vitro diagnostic) instruments. It is structured around two independent subsystems — a thermal monitoring backend and a computer vision cell analysis pipeline — both running inside Docker containers on a local machine, with a Flutter application serving as the unified user interface.

The architecture intentionally mirrors what a real embedded medical device would look like on a Torizon-managed Raspberry Pi or similar hardware platform. Swapping the target from localhost to a device IP is the only change required to deploy the full system on hardware.

What Was Built

1. C++ Temperature Sensor Server

A C++ service that simulates a thermal sensor drifting toward a target temperature with realistic gaussian noise. It exposes a REST API over HTTP using the cpp-httplib header-only library, returning JSON payloads on GET /api/temperature. The service runs inside a gcc Docker container exposed on port 8080.

2. C++ OpenCV Cell Analysis Pipeline

A second C++ service that simulates what a microfluidic imaging instrument does on each captured frame. The pipeline proceeds as follows:

- Synthetic frame generation: a 640×480 grayscale image is produced with randomised cell-like blobs — bright blobs represent live (fluorescently labelled) cells, dim blobs represent dead cells, with realistic gaussian falloff and sensor noise.
- OpenCV blob detection: cv::SimpleBlobDetector is configured with area, circularity, convexity, and brightness filters to detect each cell in the frame — the same algorithmic approach used in real cytometry software.
- Cell classification: for each detected blob, the mean pixel intensity inside the blob radius is measured. Cells above a brightness threshold are classified as alive; those below as dead.
- Frame annotation: OpenCV draws green circles around live cells and red circles around dead cells, overlaying counts on the image.
- REST delivery: the annotated frame is encoded as a PNG, converted to base64, and returned alongside cell counts and viability percentage in a JSON response on GET /api/analyze at port 8081.

3. Flutter User Interface

A Flutter/Dart desktop application providing a unified dashboard across two tabs. The temperature tab displays a live scrolling area chart (using fl_chart) with MIN/MAX statistics, polling the backend every second. The cell analysis tab displays the live OpenCV-annotated microscope frame, total/alive/dead cell counts, a circular viability gauge, and a viability trend chart, refreshing every three seconds. Both panels include connection status indicators and pause/resume controls.

Technology Stack

Component	Technology	Purpose
Sensor backend	C++17, cpp-httplib	REST API, sensor simulation
Cell analysis	C++17, OpenCV 4	Computer vision pipeline
Containerisation	Docker (gcc / ubuntu:22)	Isolated, deployable services
User interface	Flutter / Dart	Cross-platform dashboard
Charting	fl_chart	Live time-series visualisation
Communication	REST API / JSON / HTTP	Frontend-backend data exchange
Target platform	Torizon OS / Raspberry Pi	Embedded Linux deployment

Relation to TheraMe! IVD Platform

Each component of this project maps directly to what TheraMe! is building for their microfluidic diagnostic instrument:

- The OpenCV blob detection pipeline mirrors how a real instrument would image a microfluidic chip, detect cells or assay spots, and classify results — the core of their computer vision layer.
- The C++ backend architecture matches their instrument supervision layer, where a native service interfaces with hardware sensors and exposes data to higher-level software.
- The Flutter UI demonstrates the kind of touchscreen interface layer used to display diagnostic results in a clinical setting.
- The Docker/Torizon containerisation model is identical to how they manage software on their embedded Linux platform, enabling OTA updates and service isolation.

Handover to Antigravity — Suggested Next Steps

The following extensions would bring this prototype significantly closer to a production-grade IVD instrument and represent a natural continuation of the work:

Near-term (1–2 weeks)

- Replace simulated sensor data with a real I²C or GPIO temperature sensor (e.g. DS18B20) on a Raspberry Pi running Torizon OS.
- Flash Torizon to a Raspberry Pi and deploy both Docker containers using docker-compose, validating the full embedded Linux deployment path.
- Add MQTT as an alternative to REST for real-time push-based communication between backend and Flutter, which is more appropriate for continuous sensor streams.

Medium-term (2–4 weeks)

- Replace the synthetic cell image generator with a real USB or CSI camera feed, processing actual microscope or test chart images through the OpenCV pipeline.
- Implement a PostgreSQL or SQLite container for persistent data logging of temperature readings and cell analysis results.
- Tune the OpenCV SimpleBlobDetector parameters against real microfluidic chip imagery to improve classification accuracy.
- Add user authentication and role-based access control to the Flutter interface.

Longer-term (1–3 months)

- Integrate IEC 62304-compliant software lifecycle documentation and a risk management framework (ISO 14971) around the codebase.
- Implement OTA container update workflows using Torizon Platform, enabling remote deployment and rollback.
- Explore deep learning-based cell classification (e.g. a TensorFlow Lite model running on-device) to replace the brightness-threshold heuristic.
- Extend the Flutter UI with PDF report generation for diagnostic results, audit logging, and export functionality.

Closing Note

This project was built in a single session as a learning and demonstration exercise. Every component — the C++ services, the OpenCV pipeline, the Docker containers, and the Flutter application — is fully functional and runnable locally with no additional hardware. It is intended to serve as a concrete foundation that Antigravity can assess, extend, and adapt toward a real embedded medical device deployment.

All source files are included alongside this document.