

LabPulse internship achievements

Tommy Davey

Developing a configurable laboratory monitoring platform

My internship work developed LabPulse from a collection of monitoring scripts into a more maintainable system for laboratory infrastructure. I built on the existing Raspberry Pi, Arduino and Home Assistant prototype, adding a consistent architecture, richer alarms, clearer diagnostics and a repeatable installation workflow. Alongside the software, I designed and prototyped an enclosure and developed the layout for a more integrated main unit.

The result brings sensor readings, service health and operator controls into one generated dashboard. It also gives future students and researchers a practical route to configure, test, maintain and extend the system.

Why the work matters

Experiments depend on services such as cooling water, compressed air and electrical power. LabPulse helps researchers see changing conditions and receive a warning when a confirmed problem needs attention. My work focused on making those readings easier to trust and the system easier to operate over time.

Breadth of the current implementation	What this enables
6 input driver types	Arduino serial, MQTT JSON, GPIO inputs, DHT11, SHT40 and X1200 UPS.
3 main dashboard views	Monitor, System Status and Alarm Setup, with custom setup grouping and tabs.
38 automated test modules	Hardware-free checks across configuration, drivers, alarms, messaging and deployment.

My contribution at a glance

Software: isolated workers, reusable firmware and generated dashboards. **Hardware:** enclosure CAD, a printed prototype and component integration. **Reliability:** richer fault reporting, power recovery and repeatable testing. **Handover:** installation commands, backup and restore, release automation and documentation.

Development record: July to September 2026. Comparison snapshot: 17 September 2026. Driver and test-module counts describe the current repository; they are not counts of connected devices or successful live tests.

Building on the original system

The original project already delivered useful monitoring. Arduino hubs supplied compressed-air pressure and cooling-water temperature and flow; the Pi also monitored room conditions and UPS power. Readings appeared in Home Assistant and threshold or power events could trigger SMS alerts. [1]

The legacy directory contains several generations. Its later scripts already added central YAML configuration, stable USB paths, shared SMS code, recovery messages and basic heartbeats. The Future_work folder also proposed a generic driver architecture. My work developed these foundations into the current implementation. [2, 3]

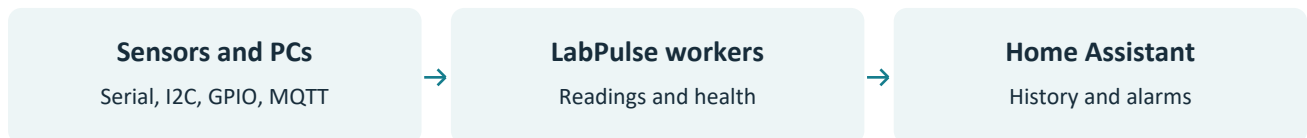
Area	Legacy capability	Current implementation
Deployment	Individual Python scripts, virtual environments and systemd services; setup scripts.	A packaged command-line tool and generated Docker Compose deployment with isolated workers.
Configuration	Fixed service sections in YAML plus a separate thresholds file.	Validated services, measurement defaults and optional measurement files generate runtime and dashboard configuration.
Dashboard	MQTT sensor discovery and some threshold controls.	Generated Monitor, System Status and Alarm Setup views, history graphs, custom tabs and setup grouping.
Alarm decisions	Script-specific threshold checks; later publishers count consecutive readings.	Low, high or range alarms, time-based confirmation, recovery delay and deadband to reduce repeated alerts.
Fault diagnosis	Reconnect loops and periodic Online heartbeats.	Separate service availability, partial faults and missing readings; recovery needs appropriate fresh evidence.
Notifications	SMS and recovery messages, with sender threads in the monitoring scripts.	One SMS worker, duplicate suppression, test recipients, mutes and subscription controls.
Extension and upkeep	Device-specific parsers, sketches and a proposed master hub.	Reusable drivers and firmware, network inputs, diagnostics, simulation, backup and restore, and release workflows.

The main improvement is the consistency of the whole system: adding or changing a measurement can update its deployment, dashboard and alarm configuration through the same validated model. [4, 5]

What I implemented

A consistent route from hardware to the dashboard

I separated hardware acquisition, dashboard generation, alarm decisions and SMS delivery. Docker containers give each enabled service its own runtime, while a shared runner handles connection, retries, reading freshness and cleanup. Drivers follow the same connect, read and close contract and declare the hardware access they need. [4]



MQTT carries readings and health between workers and Home Assistant.

Home Assistant requests SMS alerts and sends manual commands to output workers.

Broader hardware and instrument integration

I implemented support for standard serial readings, direct temperature and humidity sensors, X1200 battery and mains monitoring, named network measurements and generic digital inputs. I rewrote the Arduino firmware around reusable pressure, flow, thermistor and environmental-sensor components, with three hub examples and one consistent serial format. Missing channels can be reported without discarding the remaining readings. [4, 6]

For Triton fridges, I added Windows publishers that decode binary logfiles and send named measurements over MQTT. The production publisher includes reconnection, logging and independent heartbeats, allowing script health to be tracked separately from the age of the latest reading. The network path supports authenticated, encrypted connections. [7]

Alarms that explain what needs attention

I developed configurable threshold modes, observation windows, recovery timing and deadband. The dashboard distinguishes an out-of-range value from missing data, a partial sensor fault and a whole service going offline. Service-level reporting avoids a separate missing-data alert for every channel when an entire hub disappears. [5]

I added calculated measurements, logical setup grouping, graphs and bulk timing controls. Notification controls include global and setup mutes, test-recipient routing and inbound subscription commands. A central SMS worker serializes modem access and suppresses repeated requests. [5, 8]

Controlled outputs with an explicit scope

I added manual GPIO switches with safe-state handling on startup, shutdown and connection loss, plus an optional maximum active time. GPIO readback checks the Pi output state; it does not prove that attached equipment moved. These controls remain separate from alarm-driven automation. [9]

Designing the physical system

My contribution also covered the practical packaging of LabPulse. I worked on how the Pi, backup power, modem, sensor connections and local interface could fit together as a serviceable laboratory unit. This involved CAD layout, mounting choices, cable access and testing a printed enclosure. [14]

From CAD to a physical prototype

I developed an enclosure around the Pi stack and an adjacent housing for the Gravity IO expansion board, including the opening for its ribbon cable. I considered board clearance, port access, fixing positions and how the parts would be oriented for printing.

I had the parts printed and checked their fit. The locating lip was snug enough to hold the case together, while a thin snap clip broke and sections of the lip needed more strength. That gave me concrete feedback for the next design iteration: preserve the useful mating clearance while improving the strength of the retaining features.

Developing an integrated wall unit

I then developed the design direction for a shallow, wall-mounted enclosure with a 7-inch touchscreen. The layout brought together the Pi 5 and X1200 UPS, active cooler, cellular modem, Gravity board and powered USB hub. I explored component placement, consistent screw and standoff fixings, and retaining bars for the display.

Design decision	Practical reason
Removable front assembly and passive wall plate	Keep the electronics together so short internal USB cables do not tie opposite halves of the case together.
Top-facing USB connections	Make the removable Arduino leads accessible while keeping permanent modem wiring inside.
Gravity board on a GPIO ribbon	Place sensor connections where they can be reached without making the Pi stack taller.

Hardware integration and engineering judgement

I investigated the mismatch between the purchased Gravity board and its adapter cable, and worked through the connector arrangement for the room sensor. The enclosure work required balancing physical fit, cable bends, cooling, accessibility and straightforward assembly, rather than treating the electronics as separate boards.

The printed Pi-stack enclosure is a prototype result. The later touchscreen enclosure is documented design work, with final assembly and fit checks still to be recorded. The older STLs in the repository are separate inherited assets. [14]

Making the work maintainable

Installation and everyday operation

I created the labpulse command interface for setup, startup, status, logs, configuration and diagnostics. The doctor command checks common installation and runtime problems and suggests corrective actions. Guarded configuration regenerates the derived files, and backup and restore preserve configuration, Home Assistant state, retained MQTT data and SMS subscription state. [10]

Testing and release engineering

I added full fake-hardware operation so the configured dashboards, sensor workers and output switches can be exercised without physical devices; SMS is logged rather than sent. The repository contains 38 automated test modules and continuous-integration workflows for Python 3.11 and 3.12. Release automation checks installable packages and containers, aligns versions with Git tags and defines publication to PyPI and the container registry. [11]

Recorded installation evidence

The project roadmap records acceptance on 27 July 2026 for the Raspberry Pi 5 reference installation at revision dc6c29f. It reports two weeks of continuous operation, unplug and reconnect tests, injected sensor faults, container and Pi restarts, real UPS outages, abrupt power removal, and SMS delivery and recovery without loss of user-owned state. [12]

That work also isolated recurring USB disconnects to a faulty external hub and identified a DHT11 that could remain unresponsive after a brief power interruption. Separating those hardware defects from software behaviour is part of the engineering result. The July acceptance record applies to that revision; later SHT40, Triton, GPIO and dashboard work needs its own installation checks. [12]

Documentation and continuity

I expanded the documentation into first steps, installation, configuration, operation, troubleshooting, hardware, firmware, development and maintenance guides. I also added examples, release guidance and a screenshot checklist, and recorded the main-unit parts and enclosure design history so the next maintainer can continue the work. [6, 13, 14]

Skills demonstrated

Python architecture; embedded C++ firmware; Linux and Docker; MQTT integration; Home Assistant interfaces; CAD and enclosure design; 3D-print prototyping; hardware integration; fault diagnosis; automated testing; release engineering; and technical communication.

LabPulse remains a monitoring and notification aid. Its engineering scope does not include a safety interlock or guaranteed SMS delivery.

Evidence guide [1] legacy README, setup guide and flowchart. [2] legacy/pi_scripts and its shared library. [3] legacy/Future_work. [4] src/labpulse/common, hardware and deployment. [5] homeassistant package and User Guide. [6] firmware and hardware guides. [7] Triton publishers and guide. [8] sms package. [9] output package. [10] control.py, doctor.py and backup.py. [11] testing and .github/workflows. [12] ROADMAP.md, Stage 1. [13] docs and screenshot.md. [14] MAIN_UNIT.md and the July to August enclosure design discussions. Full references accompany the editable report.