AnomalyDetection

Stephen Hailes

October 2025

ENGF0001 Anomaly Detection in MQTT Streams

Why anomaly detection?

Modern control and monitoring systems must remain safe and efficient even when sensors or actuators start to fail. In chemical and biological processes, such as the ENGF0001 bioreactor, faults (in either sensors or actuators) can cause the temperature, pH or stirring rate to drift away from desired values. Control loops will react to this. However, if the information on which they are basing their decisions is faulty (say the temperature reading is lower than reality), they may cause a significant overreaction (they will heat the liquid until the temperature sensor reading matches the setpoint - but the actual temperature will be higher).

Traditional alarms are fixed-threshold and may miss subtle or evolving faults. **Anomaly detection** aims to recognise when system behaviour no longer matches the “normal” baseline, using only the measured signals. In this exercise you will implement and test such a detector using live process data from a simulator that reproduces the physical dynamics, sensors, actuators, and common failure modes of the real system.

This is a new exercise for us and hopefully it will go smoothly. Any problems, please post on Moodle.

Exercise overview

Rather than ask you to do anomaly detection on data from your own bioreactor, we are instead providing streams of simulated data for clarity and consistency, and to allow you the time to work on this aspect of the problem.

About the simulator The ENGF0001 simulator models the behaviour of a small laboratory bioreactor using a set of coupled differential equations that represent heat transfer, fluid mixing, and acid/base dosing dynamics. The coupling means that changes in one parameter (e.g. stirring speed) can affect others (e.g. measured temperature). Sensor readings (temperature, pH, and rotational speed) are generated from the underlying physical state, with appropriate ADC quantisation and noise. Actuators (heater, stirrer motor, acid and base pumps) are driven by a set of digital control signals.

Each control loop is governed by an internal **PID controller**, tuned using an automatic relay-based method to produce stable and realistic closed-loop behaviour. This means that when faults occur, the control system reacts dynamically — for example, increasing heater duty when a temperature sensor reads too low — allowing your detector to experience the same indirect and delayed effects that would appear in a real bioreactor.

The simulator therefore provides a realistic testbed for anomaly detection, combining process physics, sensor and actuator dynamics, and fault injection under realistic control conditions.

Data streams The ENGF0001 simulator publishes summary telemetry over MQTT in real time. Each message contains average temperature, pH, stirring speed (RPM), actuator duty cycles, and, when enabled, the list of currently active faults. You will subscribe to these data streams, train a detector on fault-free data, and evaluate its ability to identify faults under different conditions. Your detector may run either:

- on an **ESP32** device connected directly to the broker, or
- in **Python** on your laptop, subscribing to the same MQTT topics.

During the final session you will demonstrate your detector working live.

Broker and access

Broker: `engf0001.cs.ucl.ac.uk` **Port:** `1883` (unencrypted)
Access: The broker is reachable only from the UCL network. But if you are working off-campus, and writing code on your laptop, you should connect first using the UCL VPN[1].

MQTT topics in use

Each topic corresponds to a different simulator configuration. Subscribe to topics of the form:

```
bioreactor_sim/<DATA_STREAM>/telemetry/summary
```

for once-per-second summary messages in JSON format, where `<DATA_STREAM>` is:

`<DATA_STREAM>`	**Fault labels that may appear**	**Notes**
`nofaults`	*none* (no faults active)	Clean baseline data for training or calibration.
`single_fault`	`therm_voltage_bias`	Single-fault case: temperature sensor bias. Fault appears and clears on a defined schedule.
`three_faults`	`therm_voltage_bias`, `ph_offset_bias`, `heater_power_loss`	Multiple concurrent or alternating faults; use to test generality and robustness.
`variable_setpoints`	`therm_voltage_bias`, `ph_offset_bias`, `heater_power_loss`	Same faults as previously, and temperature, pH and RPM setpoints vary dynamically on a 24 hour schedule to test setpoint-robust detectors.

Operational notes All simulated faults are generated as independent stochastic sequences of failures. Each fault type has a mean time between failures (MTBF) of 300 s and a mean time to repair (MTTR) of 60 s. There is no minimum period for which a fault may be present and there may be multiple faults present at some points. As a result, faults appear and clear at random intervals, so you should expect to run your detector continuously for several minutes before faults begin to occur. The simulator streams are intended to run continuously (24/7) to support asynchronous development and testing. In the `variable_setpoints` stream, any changes to temperature, pH, or RPM setpoints are deterministic rather than random: they are scheduled to occur on the hour. This allows detectors to be tested for robustness against legitimate process transients as well as faults.

Fault label glossary

`therm_voltage_bias` Adds a voltage bias to the thermistor ADC input, creating a fixed temperature measurement error.

[1] https://www.ucl.ac.uk/isd/services/get-connected/ucl-virtual-private-network-vpn

`ph_offset_bias` Adds an offset to the pH electrode signal, producing a steady pH error.

`heater_power_loss` Scales down heater effectiveness, reducing the available heating power. Note that this also affects temperature.

Your task

1. Subscribe to one or more topics using an MQTT client or library (e.g. `paho-mqtt` in Python, or the `PicoMQTT` library for the ESP32).
2. Train your detector on the `nofaults` stream to establish a baseline of normal behaviour.
3. Test it on the faulted streams (`single_fault` and `three_faults`) and, optionally, on `variable_setpoints`.
4. Decide when to raise an anomaly flag based on deviations from baseline. You are obviously not allowed to use the fault labels in deciding what is anomalous - but you can use them for comparison with your decision.
5. Keep score - your detector should run on every published sample and should show the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Include a summary of this scoring in your final report.
6. Optionally, you may wish to try to identify the names of the faults as well as the simple presence or absence of a fault.
7. Be prepared to demonstrate your detector live during the final session. For this, we can opt to send faulty streams of data *without fault labels*, which is what one would expect in reality: it is the task of your detector to correctly analyse the data.

Expected outcomes

By completing this exercise you will:

- Understand how real-world process data can be monitored for faults using data-driven methods.
- Gain hands-on experience with MQTT messaging and real-time analysis.
- Implement and tune a simple but effective anomaly detector.
- Demonstrate a complete detection pipeline working on live bioreactor telemetry.