PROJECT ABSTRACT

IoT 3D Printer Farm Monitor with Job Queue Dashboard

A farm-wide monitoring system for 3D printers: ESP32 nodes with ACS712 current sensing read each printer's power signature to classify it as printing, idle or in error, an optical filament-runout switch raises instant alerts, and a central MQTT dashboard renders a live farm grid, a shared job queue, an alert feed and job history — with no printer firmware changes required.

Project type: Hardware & software (hybrid)

Availability: Built to order

Suitable branches: IoT & Embedded

Technologies: ESP32, ACS712 current sensor, ESP32-CAM, filament runout switch, MQTT, Mosquitto, HTML/CSS/JS dashboard, Arduino IDE

Price: Request a quotation

Prepared by Projectech · projectech.in — for academic project reference.

1. Abstract

College makerspaces and small print farms run several 3D printers at once, yet nobody can tell at a glance which machine is free, which one failed overnight, or which is about to run out of filament. This project gives the farm a nervous system: each printer gets an ESP32 monitor node that reads the printer's DC supply current through an ACS712 Hall-effect sensor, watches a filament-runout switch, and snaps progress photos with an ESP32-CAM. Nodes publish over MQTT to a central dashboard showing a live farm grid (printing / idle / error), a shared job queue, a filament alert feed and job history. Because status is inferred from real power draw, no printer firmware changes are needed, so any 12/24 V DC printer can join the farm.

2. Problem Statement

Print farms lose hours to invisible failures: a print detaches at 2 AM and the printer idles hot for six hours, filament runs out mid-job, and queued jobs wait while a free machine sits unused. Walking the room to check screens does not scale beyond two or three machines. Commercial fleet tools assume cloud connectivity and subscription pricing; college labs need a local, student-buildable alternative that works on stock printers.

This project provides per-printer power-signature monitoring with runout alerts and a central queue dashboard, built entirely from off-the-shelf embedded modules.

3. Objectives

- Detect each printer's state (printing, idle, error) from its DC supply current signature without modifying printer firmware.
- Raise filament-runout alerts within approximately 2 seconds of depletion (design target).
- Publish per-node telemetry over MQTT to a local broker for a single central dashboard.
- Provide a live farm grid, a shared job queue with priority, an alert feed and job history.
- Capture periodic and event-triggered camera snapshots for visual verification.
- Keep all figures honest: design targets and datasheet values only, with buyer-run calibration procedures documented.

4. System Architecture

The system is organized in three layers. The **node layer** consists of one monitor node per printer: an ESP32, an ACS712 5 A current sensor wired in series with the printer's 12/24 V DC supply line (mains is never touched), an optical filament-runout switch, and an ESP32-CAM for snapshots. The **messaging layer** is a local Mosquitto MQTT broker; each node publishes JSON telemetry every 5 seconds to *farm/<printer>/telemetry* and instant alerts to *farm/<printer>/alert*, with retained status messages so the dashboard is truthful on reload. The **application layer** is a single-file web dashboard subscribing over MQTT WebSockets, rendering the farm grid, job queue, alert feed and history in any browser on the local network.

5. Technologies Used

- ESP32 dev board
- ACS712 5 A current sensor
- ESP32-CAM snapshot module
- Optical filament runout switch
- MQTT · Mosquitto broker
- HTML/CSS/JS dashboard · MQTT WebSockets
- Arduino IDE · C/C++ firmware

- Wi-Fi networking

6. Working Principle

- The ACS712 sensor is wired in series with the printer's 12/24 V DC supply line (low-voltage side only); the node runs on USB 5 V.
- At installation, firmware captures each printer's idle standby baseline and adapts the idle/printing/error current thresholds to it.
- The ESP32 samples current at approximately 1 kHz, computes RMS over a 1 s window, and a state machine classifies the printer as idle, printing or error.
- Filament-switch changes trigger an immediate MQTT alert message (within approx. 2 s, design target).
- Every 5 s the node publishes a JSON packet (current, state, filament, uptime) to farm/<printer>/telemetry.
- An ESP32-CAM grabs a still snapshot every 10 min and on every alert — stills only, no video stream, keeping farm bandwidth low.
- The dashboard subscribes over MQTT WebSockets and renders the farm grid, queue and alerts live.
- Operators queue jobs with estimated durations; progress bars advance from elapsed time, and jobs complete or flag failed when the node reports a state change.

7. Key Features

- Current-signature state detection: idle / printing / error classification from real power draw, no firmware changes.
- Filament-runout alerts within approximately 2 seconds (design target) of depletion.
- Live farm grid dashboard: per-printer status, live current, filament state, elapsed time, progress bar and latest snapshot.
- Shared job queue with priority flag and pending / running / completed / failed tracking plus job history.
- Progress camera snapshots every 10 minutes and on every alert (design target).
- MQTT telemetry backbone: JSON telemetry every 5 s on per-printer topics via a local Mosquitto broker.
- Timestamped alert feed: runout, error-state and node-offline events in one incident log.

8. Applications

- College makerspaces and fab labs running 3–10 printers.
- Small commercial print farms needing local fleet visibility.
- Project labs where unattended overnight prints are common.

- Demonstrations of embedded sensing, state machines and MQTT in coursework.

9. Prerequisites

- Basic knowledge of embedded systems and IoT concepts.
- Understanding of microcontroller interfacing and low-voltage DC wiring safety.
- Basic programming knowledge (C/C++ for firmware, HTML/CSS/JS for the dashboard).
- Familiarity with MQTT publish/subscribe concepts.
- Stable Wi-Fi connectivity at the farm location.

10. Limitations

- Status is inferred from current draw, not printer firmware — a print paused from the screen reads as idle, and progress is an elapsed-time estimate, not true layer tracking.
- The ±1.5% accuracy figure is the ACS712 datasheet typical, not a measured calibration result; the report documents the buyer-run calibration procedure instead.
- Monitoring is snapshot-based (stills every 10 min plus on alerts); there is no live video stream, by design, to keep farm bandwidth low.
- The dashboard is local-network only in this build; remote access needs a VPN or reverse proxy (future scope).
- Mains voltage is never touched — the sensor sits on the low-voltage DC side only, so mains-side failures are undetectable.
- Stable Wi-Fi is required; nodes buffer only the last state locally and do not log long offline histories.

11. Future Scope

- Remote dashboard access via VPN or secure reverse proxy.
- Layer-accurate progress by parsing G-code or reading printer serial output.
- Per-job energy accounting from integrated current readings.
- Automatic job dispatch rules (e.g. fastest idle printer first).
- SMS/push notification integration for critical alerts.

12. Conclusion

The IoT 3D Printer Farm Monitor with Job Queue Dashboard gives a small printer fleet a nervous system built from honest embedded engineering: current-signature state detection, instant runout alerts, MQTT telemetry and a live operational dashboard — without touching any printer's firmware. It stays truthful about its limits: state is inferred from power draw, progress is estimated from elapsed time, and monitoring is snapshot-based by design. For a final-year project it combines current sensing, state-machine firmware, networking and a real dashboard around a genuine makerspace problem.