Smart Garbage IoT for Sustainable Cities using Blynk

ESP32-WROVER Prototype with Fill Monitoring, Fire Detection, Tilt-Over Detection, and Local WebUI/REST

Oğuzhan Alasulu & Serkan Kıyaklı
Abdullah Gül University (AGU), Kayseri, Türkiye
COMP413 – IoT Course Project (Group 8)
https://github.com/alasulu/Comp413_g8_SmartGarbageSystem

January 2026

ABSTRACT

This report presents *Smart Garbage IoT*, a practical IoT prototype for smart waste-bin monitoring in sustainable-city scenarios. The device runs on an ESP32-WROVER, samples low-cost sensors on-device, and transforms noisy raw signals into operationally actionable indicators using a reliability-aware decision pipeline (majority sampling, stability timers, and event latching). The system integrates: (i) a top-mounted KY-032 used as a threshold-style fill/service-needed indicator, (ii) a KY-026 flame sensor where the digital D0 channel is treated as the alarm source and drives local buzzer/traffic-LED behavior, (iii) a KY-020 tilt sensor with stable and latched states for vandalism/impact scenarios, and (iv) DHT11 temperature/humidity telemetry for environmental context. Status is exposed locally through a WebUI and REST endpoints and remotely through a Blynk cloud dashboard using virtual pins for telemetry and limited configuration. The final prototype demonstrates end-to-end observability (local commissioning + remote monitoring) and a clean separation between safety-critical alerts (fire) and maintenance signals (fill/tilt), aligned with realistic deployment expectations for public infrastructure.

Key words: IoT; smart waste management; sustainable cities; ESP32; Blynk; WebUI; REST; fire detection; fill level; tilt detection

1 INTRODUCTION

Municipal solid-waste operations are tightly connected to public health, urban cleanliness, and operational cost. Conventional collection is often schedule-driven, which may cause unnecessary routes (bins that are not yet full) and delayed service (bins already full). In parallel, public bins can experience safety hazards such as localized fires and physical disruption due to wind, accidental impacts, or vandalism. These realities motivate a monitoring approach that is not only sensor-driven, but also *action-driven*: the system should answer operational questions such as "Should this bin be serviced soon?", "Is there a safety hazard requiring immediate attention?", and "Is the bin likely to be unavailable due to tilt-over?"

The goal of *Smart Garbage IoT* is to provide three decision-level indicators with clear operational meaning:

- **Service-needed (Fill):** a top-mounted KY-032 is placed near the bin top to act as a threshold indicator for near-full conditions, mapping directly to a maintenance decision.
- **Safety hazard (Fire):** the KY-026 digital output (D0) triggers a local alarm (buzzer + traffic LEDs) and is reported through both monitoring interfaces.
- **Integrity/Availability (Tilt):** the KY-020 tilt signal is debounced and latched to make physical disturbances visible and traceable.

A complementary project objective is **observability**: technicians should be able to validate installations locally via WebUI/REST, while operators can monitor bins remotely via Blynk.

[H]

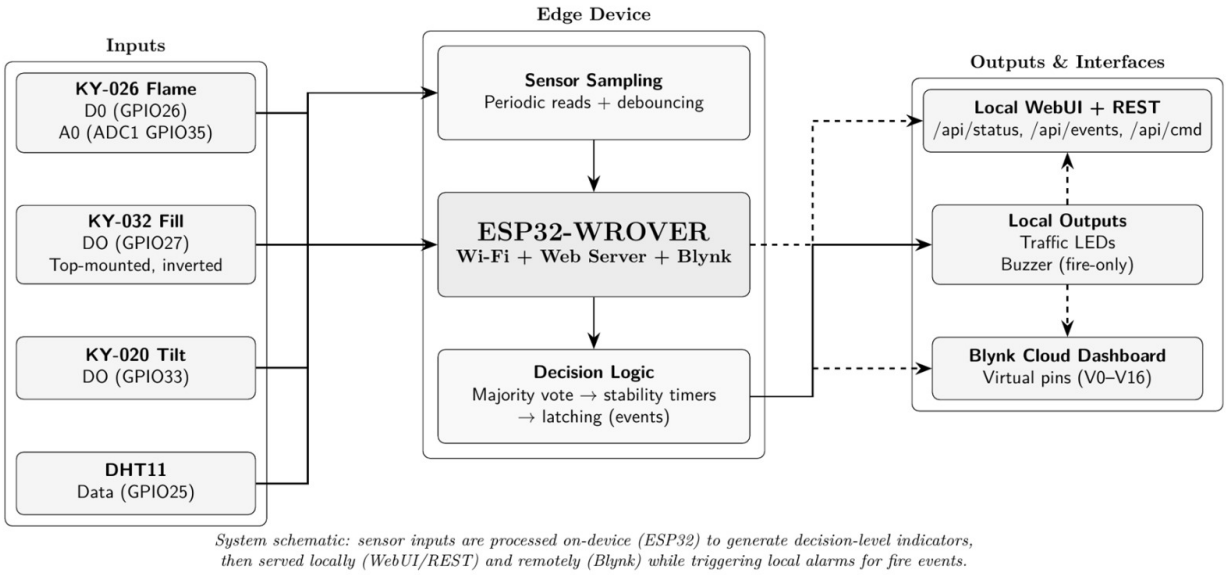


Figure 1. System architecture (`assets/architecture.jpg`). Inputs (KY-026, KY-032, KY-020, DHT11) are sampled by ESP32-WROVER. Firmware applies filtering and decision logic, drives local alarms for fire events, and publishes telemetry via Local WebUI/REST and Blynk cloud.

2 SYSTEM MODEL AND ARCHITECTURE

2.1 End-to-end Data Flow

The firmware implements a closed-loop pipeline: sensor inputs are sampled periodically; raw reads are filtered and converted into stable states; state changes are recorded as events; and the resulting telemetry is published through two interface layers. Figure 1 summarizes this architecture and highlights the design separation between local safety actuation (fire-only) and monitoring/maintenance indicators (fill/tilt).

2.2 From Noisy Inputs to Clean Decisions

Low-cost digital modules are practical for prototyping but can toggle due to reflectivity, mechanical vibration, and placement variability. For a city-style dashboard, raw toggling is not usable: operators require stable decisions and event traces. Therefore, the firmware uses a three-stage strategy:

(i) **Majority sampling:** multiple fast reads reduce impulsive flips.
(ii) **Stability timers:** a candidate state must persist before being accepted.
(iii) **Latching + event log:** transitions are recorded as discrete events suitable for alert rules and debugging.

This logic is visible in the WebUI (raw vs processed fields) and in the event list, which is also accessible via `/api/events`.

3 HARDWARE DESIGN

3.1 Component Selection and Placement

The ESP32-WROVER is used as the main controller due to its WiFi capability, adequate GPIO, and practical memory headroom for running an HTTP server and cloud telemetry concurrently. The sensors were selected for accessibility and clear binary behavior:

- **KY-032 (Fill):** installed near the top of the bin, effectively transforming “object close to top” into “service-needed”. This placement is mechanically simple and aligns with the operational objective of avoiding overflow.
- **KY-026 (Fire):** the digital D0 channel is treated as the alarm decision input, allowing fast and deterministic fire detection.

[H]

Table 1. ESP32 pin mapping used in the repository implementation.

Function	Module	GPIO
Flame D0	KY-026	26
Flame A0 (ADC1)	KY-026	35
Fill DO	KY-032	27
Tilt DO	KY-020	33
DHT11 signal	DHT11/KY-015	25
Buzzer	Alarm	18
Red LED	Alarm	19
Yellow LED	Alarm	21
Green LED	Alarm	22

Analog A0 is still captured for visibility but does not drive the alarm in the final firmware to avoid false triggers caused by saturation or ambient effects.

- **KY-020 (Tilt):** provides a discrete signal for tilt-over conditions; the firmware adds stability and latching to turn a momentary vibration into a meaningful event when sustained.
- **DHT11 (KY-015):** contributes environmental context (temperature/humidity), useful for interpreting sensor behavior during demonstrations and future field testing.

3.2 Pin Mapping

Table 1 lists the GPIO mapping used in the final implementation. The mapping is chosen to keep sensor inputs on stable digital pins and to reserve a suitable output set for the buzzer and the three traffic LEDs (red/yellow/green), which provide immediate local status feedback.

4 SOFTWARE DESIGN

4.1 Firmware Structure

The firmware is implemented as a single ESP32 application that continuously samples sensors and updates a shared state structure used by both interface layers. The execution model is intentionally simple: a periodic loop performs sensor acquisition and decision updates; the HTTP server responds to WebUI/REST requests; and Blynk telemetry is pushed at a controlled rate. This approach keeps the firmware maintainable and suitable for incremental extension (e.g., additional sensors, data logging, or OTA updates).

4.2 Reliability Logic: Majority Sampling, Timers, and Latching

Listing 1 illustrates the core pattern used for digital sensors: multiple reads are aggregated into a majority decision, a timer ensures stability, and then a latched state is updated and logged. The practical outcome is that dashboards and logs reflect sustained conditions rather than micro-jitter.

Listing 1: Simplified excerpt illustrating majority + stability + latch logic used for digital sensors.

```
bool majorityReadDigital(uint8_t pin, bool activeLow, uint8_t samples, uint8_t require) {
  uint8_t c = 0;
  for (uint8_t i = 0; i < samples; i++) {
    bool v = digitalRead(pin);
    bool active = activeLow ? !v : v;
    if (active) c++;
    delayMicroseconds(200);
  }
  return (c >= require);
}

// Pseudocode structure:
// - compute majority state
```

```
// - enforce stability window
// - update latched state and event log on transitions
```

4.3 Bounded WiFi Connection and WebUI Memory Strategy

Two implementation choices significantly improve practical behavior during demonstrations and real deployments. First, WiFi connection is bounded by a timeout so the device does not stall indefinitely in weak coverage; instead, it can continue running local functions and expose commissioning information when connectivity becomes available. Second, the WebUI assets are stored in flash (PROGMEM) to reduce RAM pressure while running WiFi, HTTP server, and telemetry concurrently. These choices directly support reliability and reduce "mysterious" failures during long runtimes.

4.4 Local WebUI and REST Endpoints

The local WebUI is designed for field visibility: it exposes RSSI and uptime, separates raw from processed decision values, and provides an event list that explains why a state changed. In addition, the REST API enables programmatic monitoring and debugging. The key endpoints are:

- `GET /api/status` for a JSON snapshot (fire/fill/tilt/DHT/buzzer/RSSI/uptime),
- `GET /api/events` for the ring-buffer event history,
- `POST /api/cmd` for limited commands (mute/silence and selected configuration).

Listing 2 shows a representative excerpt for generating the `/api/status` JSON response. The intent is to keep the status schema explicit and readable, which simplifies integration into external dashboards or scripts.

Listing 2: Excerpt-style example: building `/api/status` JSON response.

```
void apiStatus() {
  String j = "{";
  j += "\"fireDetected\":" + String(state.fire ? "true":"false") + ",";
  j += "\"fillPct\":" + String(state.fillPct) + ",";
  j += "\"tiltLatched\":" + String(state.tiltLatched ? "true":"false") + ",";
  j += "\"tempC\":" + String(state.tempC, 1) + ",";
  j += "\"humPct\":" + String(state.humPct, 1) + ",";
  j += "\"rssiDbm\":" + String(WiFi.RSSI()) + ",";
  j += "\"uptimeMs\":" + String(millis());
  j += "}";
  server.send(200, "application/json", j);
}
```

4.5 Blynk Telemetry and Controls

Blynk acts as the remote monitoring layer: it provides an operator-friendly dashboard and a lightweight channel for selected controls. Telemetry pins reflect the decision-level indicators (fire/fill/tilt) and environmental data, while control pins are limited to safe runtime actions such as muting/silencing the buzzer and configuring polarity for installation variability. Table 2 summarizes the virtual-pin mapping used by the final project.

5 RESULTS AND DISCUSSION

This section presents interface evidence (WebUI and Blynk), summarizes observed values from the demonstrations, and interprets the results from a deployment perspective.

[H]

Table 2. Blynk virtual pins used in the final implementation.

VPin	Name	Type	Meaning
V0	FireDetected	Telemetry	1 if fire detected via KY-026 D0 (alarm source)
V1	FlameStrengthFilt	Telemetry	Filtered flame strength (visibility only)
V2	FlameA0raw	Telemetry	Raw ADC value from KY-026 A0 (visibility only)
V3	BinFillPct	Telemetry	Fill percentage inferred from KY-032 logic
V4	TempC	Telemetry	DHT11 temperature ()
V5	HumPct	Telemetry	DHT11 humidity (%)
V6	TiltDetected	Telemetry	1 if tilt is latched
V10	FlameThreshold	Ctrl/Tel	Display threshold (A0 detection disabled in final firmware)
V11	MuteBuzzer	Control	Mute fire alarm buzzer
V12	Silence10Min	Control	Timed silence trigger
V13	KY032_ActiveLow	Control	KY-032 polarity configuration
V14	FlameD0_ActiveLow	Telemetry	Display-only (fixed polarity)
V15	KY026_ActiveLow	Telemetry	Display-only (fixed polarity)
V16	KY020_ActiveLow	Control	KY-020 polarity configuration

[H]

Table 3. Observed values from Blynk screenshots (Default / Event / Filled).

VPin	Name (Widget)	Meaning	Default	Event	Filled
V3	Filling Perc. (Gauge)	Bin fill percentage inferred from KY-032 logic	0	0	100
V4	Temperature of Ambiance	DHT11 temperature ()	21.6	21.6	19.9
V5	Humidity of Ambiance	DHT11 humidity (%)	38.7	24.6	10.0
V11	Mute (Switch)	Fire alarm mute control	Unmuted	Unmuted	Unmuted
V10	Flame Detection Thr. (Slider)	Display threshold (A0 detection disabled)	900	900	900
V1	FlameStrengthFilt	Filtered flame strength (visibility only)	4094	3097	2070
V6	TiltDetected (Indicator)	Tilt event state (latched)	0	1	0
V2	FlameA0raw	Raw ADC value from KY-026 A0 (visibility only)	0	1053	2874

5.1 Local WebUI Evidence Across System States

Figure 2 shows the local WebUI in four representative conditions. The nominal view confirms healthy operation (WiFi RSSI visible, fire/tilt inactive, fill at 0%). The fire view demonstrates that the alarm decision is driven by the KY-026 digital channel and that local actuation (buzzer + traffic LEDs) is tied to this safety event. The filled view demonstrates that the fill indicator is presented as an operational percentage (0/100 in this prototype) derived from the top-mounted threshold logic. Finally, the tilted view illustrates the benefit of stability and latching: a physical disturbance becomes a clear, persistent maintenance signal rather than a momentary glitch.

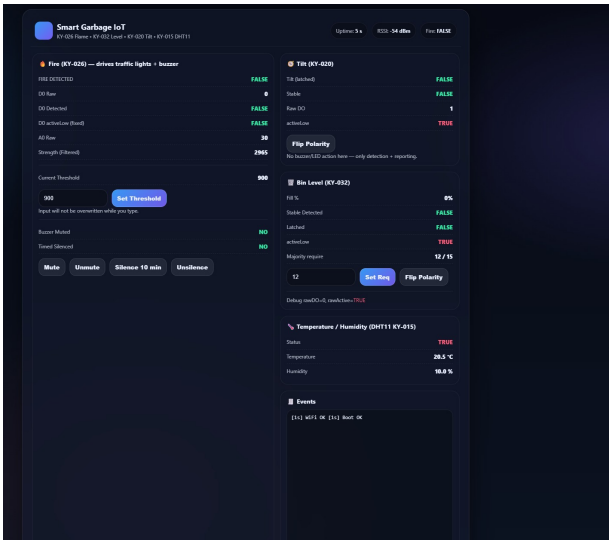
5.2 Remote Monitoring Evidence via Blynk

Figure 3 shows the Blynk dashboard in nominal, event, and filled scenarios. The dashboard aggregates the system into a compact operational view: fill percentage, temperature/humidity, tilt indicator, and fire-related telemetry. Importantly, the Blynk layer complements the WebUI rather than replacing it: WebUI is ideal for onsite validation (including RSSI and event list), while Blynk supports remote supervision and basic runtime control (mute/silence and polarity configuration).

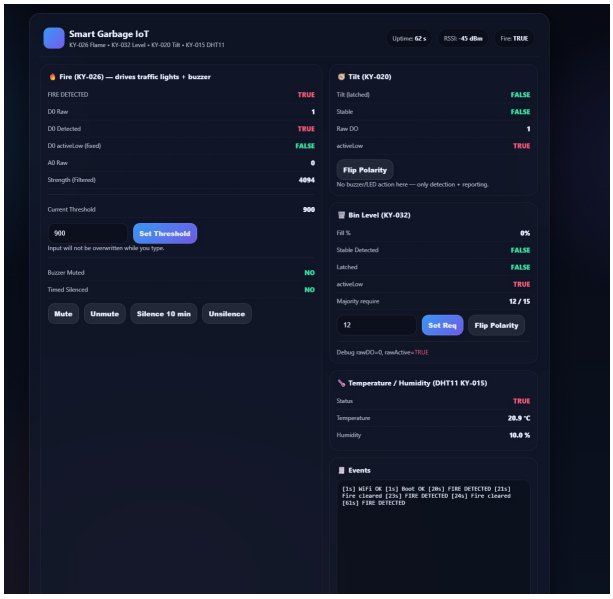
5.3 Observed Dashboard Values and Consistency Checks

Table 3 summarizes values visible in the captured Blynk screenshots. The table serves as a consistency check: the filled scenario correctly reports 100% fill while returning to 0% in other scenarios; the tilt scenario correctly toggles only the tilt indicator; and environmental values remain plausible and change across captures. Fire-related analog values are shown as visibility-only telemetry, while the alarm decision is tied to the digital D0 channel.

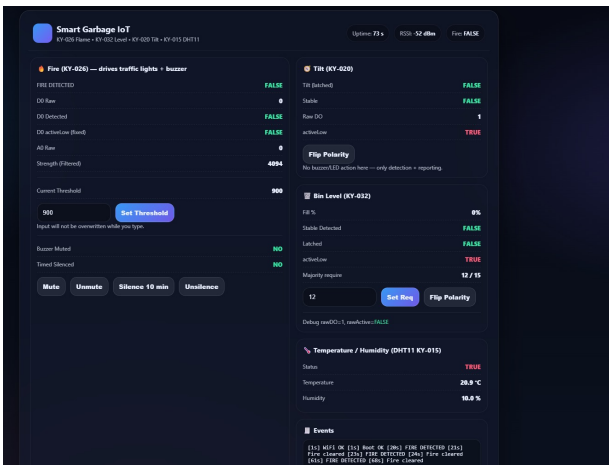
[H]



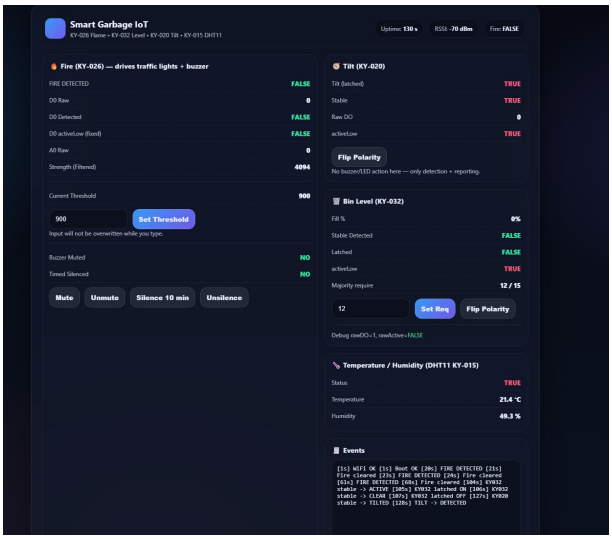
(a) Default (nominal) WebUI view.



(b) Fire detected (alarm state shown in WebUI).



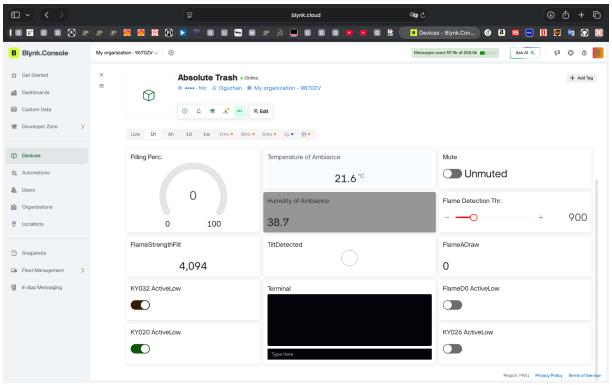
(c) Filled state (service-needed indicator at 100%).



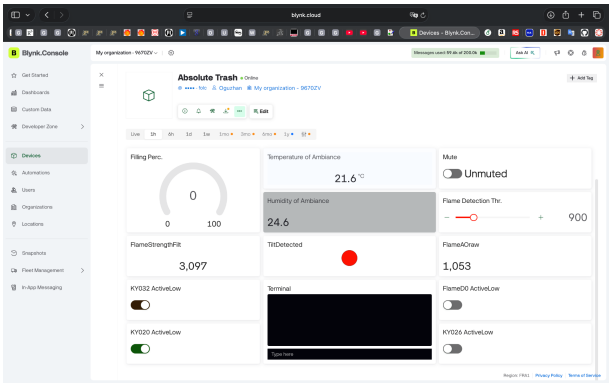
(d) Tilt-over detected (stable + latched).

Figure 2. Local WebUI snapshots (`assets/webui_*.png`). The interface exposes both raw readings and decision-level states, enabling com-

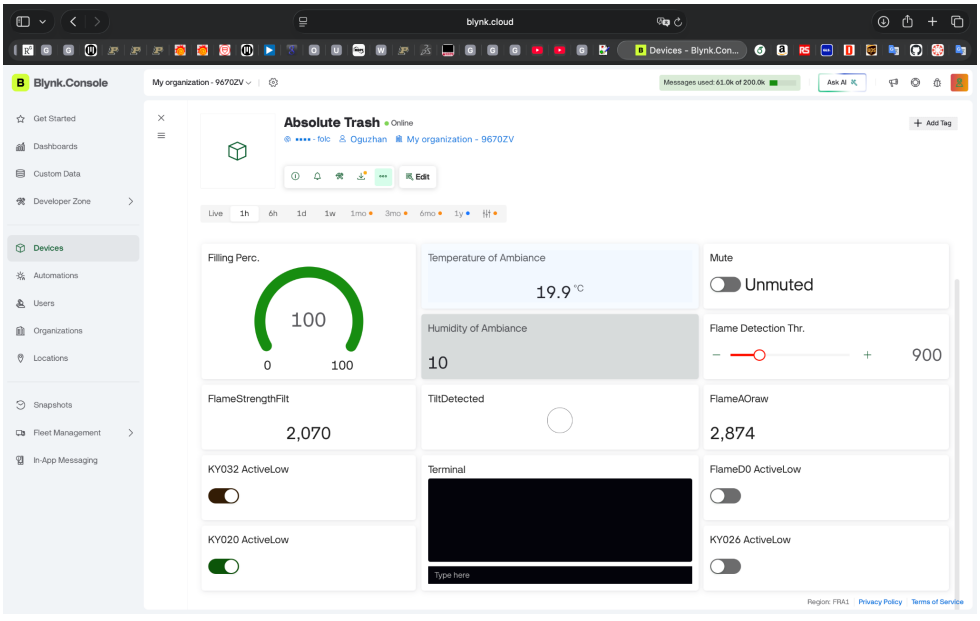
[H]



(a) Nominal dashboard view.



(b) Event view (tilt example).



(c) Filled view (fill gauge at 100%).

Figure 3. Blynk dashboard snapshots (`assets/blynk_*.png`). Telemetry reflects decision-level states and supports remote monitoring and limited configuration.

[H]

Table 4. Snapshot metrics populated from captured WebUI values (directly observable evidence).

Metric	Min	Avg	Max	Evidence source (WebUI captures)
WiFi RSSI (dBm)	-70	–	-45	Top bar values across captures: -54 (default), -45 (fire), -51 (filled), -70 (tilted).
Uptime (s)	5	–	130	Top bar values across captures: 5 (default), 62 (fire), 104 (filled), 130 (tilted).
Bin fill (%)	0	–	100	Bin Level card: 0% in default/fire/tilt, 100% in filled.
Temperature ()	20.5	–	21.4	DHT11 card values across captures.
Humidity (%)	10.0	–	49.3	DHT11 card values across captures.
Tilt status	FALSE	–	TRUE	Tilt card shows TRUE in tilted capture, otherwise FALSE.
Fire status	FALSE	–	TRUE	Fire card shows TRUE in fire capture, otherwise FALSE.

5.4 Snapshot Metrics from WebUI Captures

Table 4 lists quantitative values directly visible on the WebUI (RSSI, uptime, fill, temperature, humidity, and boolean event states). These values are not intended as long-run statistics; instead, they demonstrate that the device reports coherent telemetry and that key status fields are observable during staged scenarios. In a deployment setting, these same fields support practical workflows: RSSI helps diagnose connectivity quality, uptime helps confirm stability, and the event booleans provide immediate understanding of why alerts are active.

5.5 Reliability and Deployment Perspective

The central result of the project is not merely that sensors can be read, but that their signals can be turned into **stable operational indicators** with clear meaning. Majority sampling and stability timers reduce noise sensitivity, while latching makes event transitions explicit and traceable. This is essential for real deployments: fleet dashboards and alerting policies cannot depend on raw edge jitter. The architecture also demonstrates a practical observability stack: WebUI supports onsite commissioning and debugging (including event history), and Blynk supports remote monitoring and quick controls (mute/silence and polarity).

6 REAL-LIFE DEMONSTRATION

The physical prototype and its staged scenarios are documented both through real-life images and a short demonstration video:

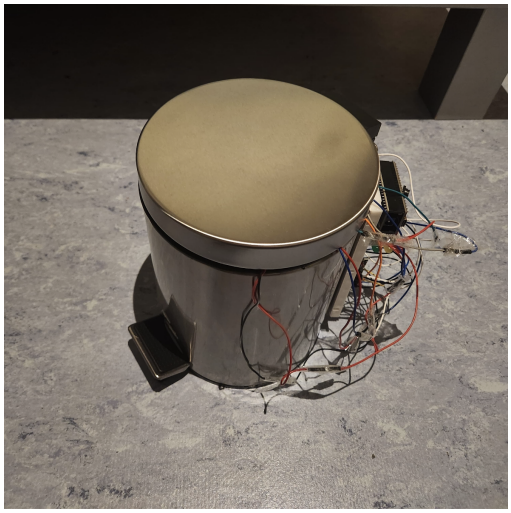
https://www.youtube.com/watch?v=eFk5osx5mvY

The demonstration validates the end-to-end workflow: device boot, WiFi connectivity, local WebUI visibility, and remote Blynk updates under representative conditions. The staged tests include a fill/service-needed scenario aligned with top-mounted sensing, a tilt-over scenario aligned with physical disturbance, and a fire scenario where local alarm behavior is intentionally tied to the fire decision input.

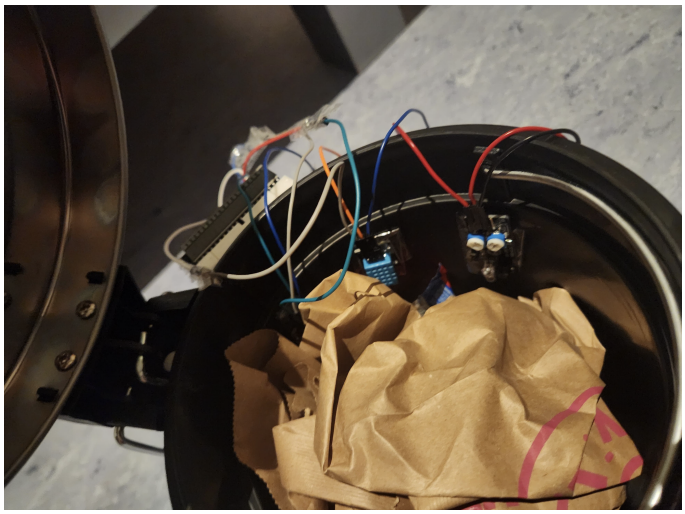
7 POSTER

Figure 5 shows the final course poster. The poster is structured as a single narrative: it motivates the sustainable-city problem, summarizes objectives (fill, fire, tilt), visualizes the architecture, and presents implementation evidence through the physical prototype image and dashboard screenshots. The inclusion of a QR code and repository reference supports reproducibility and allows viewers to directly access the project artifacts (firmware, WebUI, and dashboard mapping).

[H]



(a) Assembled prototype hardware (ESP32, sensors, LEDs, buzzer).



(b) Inside-bin placement illustrating top-mounted fill sensing near the lid area.

(c) Demonstration scenario with active local outputs during an event condition.

Figure 4. Real-life prototype evidence (`assets/real_life_*.jpg`). The physical build and sensor placement correspond directly to the decision-level states shown in WebUI and Blynk, supporting a coherent end-to-end validation of sensing, actuation (fire), and reporting.

[H]

Figure 5. Course poster summarizing Smart Garbage IoT (`assets/poster.png`). The layout connects motivation, architecture, hardware configuration, monitoring interfaces, and results into a single compact deliverable for presentation.

8 CONCLUSION AND FUTURE WORK

Smart Garbage IoT demonstrates that a low-cost sensor set can become operationally meaningful when paired with reliability-aware decision logic and a two-layer observability strategy. The ESP32-WROVER platform successfully integrates fill/service-needed monitoring, safety-driven fire detection with local alarms, and tilt-over detection for integrity/availability. The WebUI provides commissioning-grade visibility through raw/processed fields and an event trace, while Blynk enables remote monitoring and limited safe controls suitable for multi-bin scenarios.

Future work toward city-scale readiness includes long-run field logging to quantify false alarms across seasons, power optimization for battery/solar operation, OTA update support for maintainability, and aggregated analytics for route optimization. In addition, deployment-focused enclosure design and sensor calibration procedures would improve mechanical robustness and consistency across installations.

ACKNOWLEDGEMENTS

We thank the COMP413 teaching team for guidance and the rubric-driven feedback that helped refine the final repository implementation, monitoring interfaces, and demonstration deliverables.

REFERENCES

Blynk Inc. *Blynk Documentation*. Available at: https://docs.blynk.io/. Accessed: 2026-01.

Espressif Systems. *ESP32 Technical Documentation*. Available at: https://www.espressif.com/en/support/documents/technical-documents. Accessed: 2026-01.

Espressif Systems. *Arduino core for the ESP32*. GitHub repository: https://github.com/espressif/arduino-esp32. Accessed: 2026-01.

Aosong (ASAIR). *DHT11 Temperature and Humidity Sensor Datasheet*. Available at: https://components101.com/sites/default/files/component_datasheet/DHT11%20Datasheet.pdf. Accessed: 2026-01.