FASS SmartPark-IoT Design (AC Power)

Single-camera, per-slot occupancy sensing node with event-driven telemetry, reliability, and ops monitoring.

Prepared for CS 48007 / CS 58007 - Internet of Things Sensing Systems (Sabanci University)

Target site: Tuzla Campus - FASS Parking Lot

Date: 2026-01-03

Document intent: Provide a technically detailed, deployment-ready system design for a single-lot, per-slot parking occupancy sensing node. The system prioritizes IoT architecture (sensing, calibration, telemetry, reliability, ops) while keeping ML as a supporting component.

1. Executive summary

This document specifies an IoT-first system architecture for real-time parking slot occupancy monitoring at the FASS parking lot. A fixed overhead camera acts as the primary sensor. A Raspberry Pi 4 performs on-edge processing to convert raw video into calibrated, per-slot occupancy events. The node publishes only low-bandwidth telemetry (state changes + periodic summaries + health metrics) to a campus broker/server, enabling a live dashboard and historical analytics. The design emphasizes deployability, reliability under network outages, remote operations, and privacy-by-design. ML is optional and used only when it materially improves sensing quality.

• Coverage: all visible slots from a single mounted camera viewpoint.

• Outputs: per-slot state events (occupied/free/unknown), lot summary, node health telemetry.

• IoT guarantees: store-and-forward buffering, heartbeat watchdog, remote configuration, versioned sensing artifacts.

• Power mode: AC-powered edge node and camera with safety-first electrical practices.

2. Site assumptions and requirements

2.1 Target site

FASS parking lot (Sabanci University Tuzla Campus). System scope is limited to the parking lot footprint visible in the mounted camera view.

2.2 Functional requirements

• Detect per-slot occupancy in real time and publish events only on state changes.

• Provide periodic lot-level summaries (free/occupied/unknown counts).

• Expose operational telemetry for maintainability (fps, temperature, connectivity, buffer depth).

• Support remote configuration updates without physical access (debounce, publish rates, ROI map version).

• Operate continuously during short network outages and recover without losing events.

2.3 Non-functional requirements

• Low bandwidth: do not stream video to the server by default; publish structured events.

• Privacy: process video on-device; store only aggregated/events; optionally keep a short local debug ring buffer.

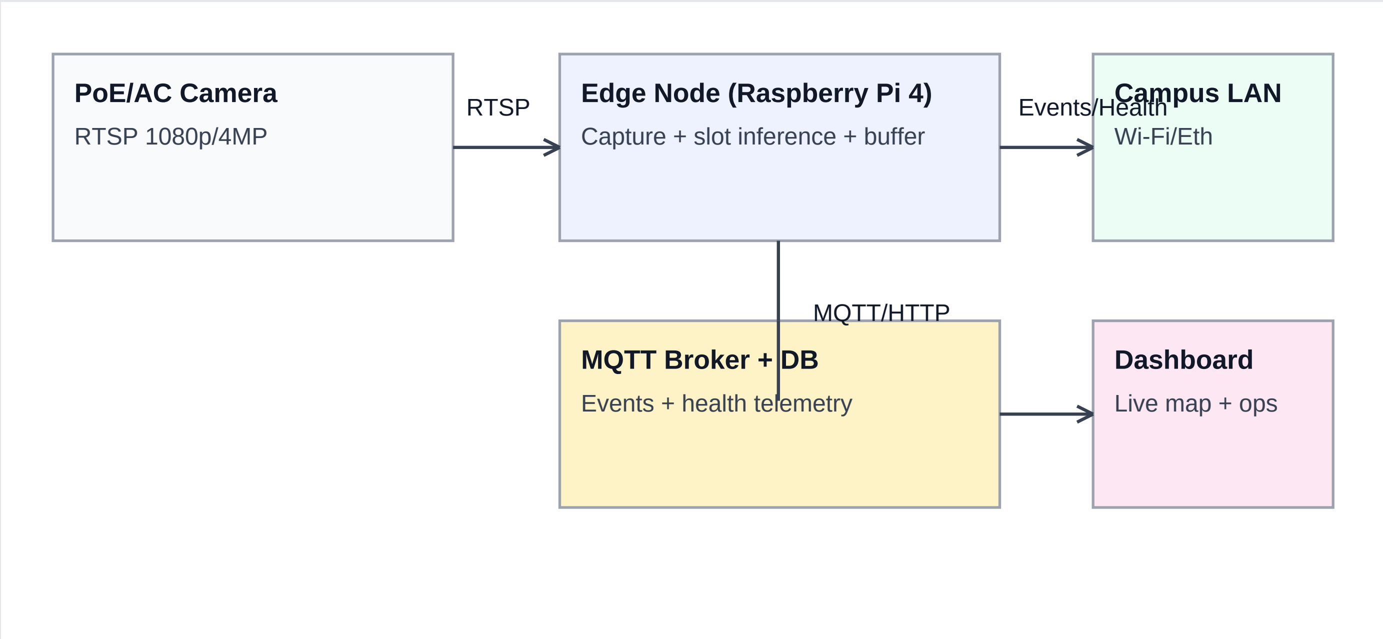
• Robustness: tolerate illumination changes (shadows, glare) and camera exposure drift; minimize flicker.

• Maintainability: systemd-managed services, crash recovery, clear logs and metrics.

• Security: authenticated broker access; minimize exposed services on the edge node.

3. System architecture

High-level dataflow and separation of concerns (sensor, edge inference, telemetry, storage, visualization).



3.1 Node roles

• Camera (sensor): provides a stable RTSP stream. Fixed pose; no PTZ; exposure locked if possible.

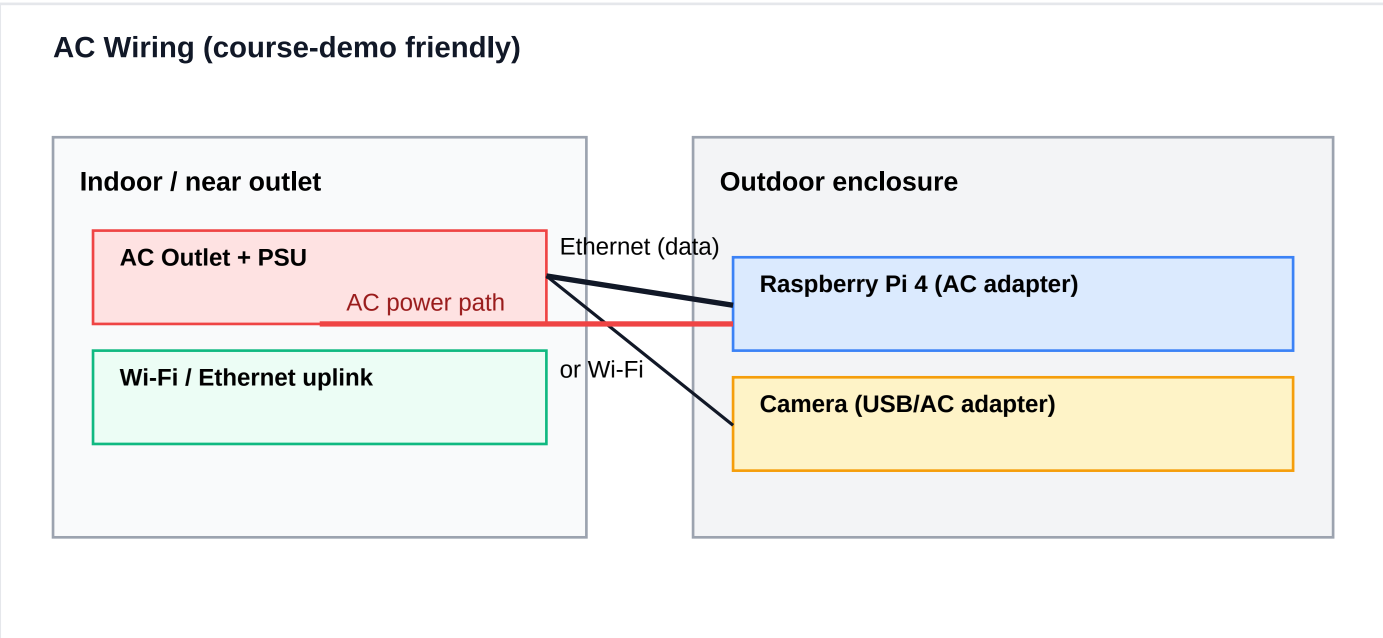
• Edge node (Pi 4): performs capture, geometric normalization, per-slot occupancy inference, debouncing, buffering, and telemetry publishing.

• Broker/DB: receives MQTT messages; stores events and health telemetry; supports retention policies and queries.

• Dashboard: shows live lot state, historical occupancy, and node health; can push config updates.

4. Power and wiring

AC mode is suitable for course demos when PoE is unavailable. It requires stricter safety practices (RCD/GFCI) and weatherproof power routing. Use separate, stable power supplies for the Pi and camera to avoid brownouts and noise coupling.



• All AC wiring must comply with campus safety rules; use outdoor-rated cabling, RCD/GFCI, and proper enclosures.

• Avoid long USB power runs; convert AC to DC near the enclosure.

• Use EMI filtering or ferrites if the camera introduces noise into the Pi supply.

4.1 Bill of materials

Item	Recommended spec	Notes
Edge compute	Raspberry Pi 4 (4GB+), microSD 64GB+ (A2)	Run capture/inference, buffering, MQTT client
Camera	Fixed PoE IP camera w/ RTSP (1080p+), wide angle 2.8-3.6mm	If AC mode: USB camera or IP camera with separa
Enclosure	IP65 weatherproof box + cable glands + mounting brackets	Consider heat: heatsinks, vents, optional 5V fan
Networking	Cat6 outdoor-rated + strain relief	Wi-Fi fallback supported but not primary
Mount	Wall/pole mount, vibration-resistant	Prefer non-vibrating surface, avoid glare direction
AC power	Outdoor-rated AC line + RCD/GFCI protection	Safety-first; comply with campus rules
PSU (Pi)	5V 3A official supply or DC buck converter	Avoid undervoltage brownouts
PSU (camera)	USB 5V or camera-specific adapter	Separate power isolation reduces interference

5. Sensor calibration and slot mapping

In this project, calibration is treated as a first-class sensing activity. The parking lot is modeled as a calibrated sensor array, where each slot is a fixed channel with a stable polygon ROI. Calibration artifacts are versioned and traceable in telemetry.

5.1 Calibration artifacts (versioned)

• roi_mask.png: binary mask limiting processing to the parking area.

• fass_slots_vX.json: all visible slots as polygons in image or rectified coordinates.

• homography_vX.json: optional planar transform for rectification; improves robustness to perspective and small camera shifts.

• camera_config.json: stream URL, resolution, frame rate cap, exposure/white-balance notes.

5.2 Slot mapping workflow

• Capture 2-3 reference frames at different times (midday + shadowy).

• Use a labeling tool to draw polygons for every visible slot; export to JSON.

• Validate polygons by overlaying them live and visually inspecting coverage and alignment.

• Freeze ROI version (v1) for baseline experiments; update to v2 only when justified and tracked.

5.3 Rectification (recommended)

• Select 4+ stable ground control points (lot corners / painted marks) visible in the frame.

• Compute homography to a canonical plane; run inference in rectified space.

• Monitor drift: if camera shifts, detect misalignment via reference markers and trigger maintenance.

6. Edge software stack (IoT-first)

6.1 Processes and services

• capture_service: pulls frames from RTSP, timestamps, and feeds the inference pipeline.

• inference_service: per-slot occupancy scoring, debouncing, event generation.

• telemetry_service: MQTT publish, buffering, and replay logic.

• health_service: periodically reports node metrics and camera stream status.

• config_service: subscribes to config topic and applies safe, validated updates.

6.2 Data handling and privacy

• Default mode: do not upload/store raw frames off-device.

• Optional local ring buffer: keep last 2-5 minutes of low-fps frames for debugging; auto-delete and never publish unless explicitly requested.

• Region masking: exclude pedestrian walkways and building entrances from processing.

• Persist only events and aggregate statistics on the server.

6.3 Occupancy inference (supporting ML)

Occupancy can be implemented as (A) a non-ML baseline (background difference + thresholds) and/or (B) a lightweight classifier per ROI. The system design is agnostic: inference produces a probability/confidence and an 'unknown' state when uncertainty is high.

• Debounce/hysteresis: require state stability for K seconds before publishing a change; separate thresholds for enter vs exit to reduce flicker.

• Temporal smoothing: exponential smoothing or simple HMM on slot state to avoid rapid toggles under shadows.

• Confidence gating: label as UNKNOWN when confidence is below a threshold; publish unknown_count in summary.

7. Telemetry protocol and schemas

Telemetry is event-driven. Slot state is published only on transitions. Periodic summaries and node health metrics are published on a schedule. All messages include timestamps and version fields to ensure reproducibility.

Topic	QoS	Direction	Payload (key fields)
su/parking/fass/slot/<slot_id>/state	1	Edge -> Broker	occupied, confidence, ts_utc, dwell_s, roi_version, model_ver
su/parking/fass/summary	0	Edge -> Broker	free_count, occupied_count, unknown_count, ts_utc
su/parking/fass/node_health	0	Edge -> Broker	uptime_s, fps, cpu_temp_c, mem_mb, net_rssi_dbm, dropped
su/parking/fass/config	1	Broker -> Edge	publish_period_s, debounce_s, hysteresis, roi_version, loggin

7.1 Timestamps and time sync

• Use NTP on the edge node. Emit ISO-8601 UTC timestamps (ts_utc) in all payloads.

• Include local monotonic time internally for debounce logic; do not rely solely on wall-clock time.

• Server stores received_at timestamps to measure end-to-end latency.

8. Reliability, buffering, and operations

8.1 Store-and-forward buffering

• On publish failure, append messages to a local queue (SQLite or append-only log).

• Replay queue on reconnect in chronological order; deduplicate by (slot_id, transition_ts).

• Expose buffer_depth in node_health; alert if it grows beyond a threshold.

8.2 Watchdogs and self-healing

• Run services via systemd with Restart=always and appropriate restart backoff.

• Implement an application-level heartbeat (e.g., last_loop_ts) and restart if loop stalls.

• Monitor camera stream: if RTSP breaks, attempt reconnect; after N failures, raise an alert.

8.3 Remote configuration

• Config messages are validated (schema + bounds) before application.

• Changing ROI or homography requires bumping roi_version; node refuses unknown ROI versions unless the file is present locally.

• Keep a last-known-good config snapshot for rollback.

9. Security considerations

• MQTT authentication: per-node username/password or token; restrict topics by ACL.

• TLS where feasible; if not available, use campus-private network segmentation and rotate credentials.

• Disable unnecessary services on the Pi; firewall inbound connections; access via SSH keys only.

• No raw video egress by default; only structured telemetry.

10. Validation and evaluation plan

10.1 IoT-centric metrics

• End-to-end latency: edge event ts_utc -> dashboard render time.

• Bandwidth: bytes/sec and events/sec; compare event-driven vs periodic publishing.

• Uptime: percentage of time node is producing telemetry; downtime causes and recovery time.

• Outage test: disconnect network for 2 minutes; verify buffer replay with no lost transitions.

10.2 Sensing metrics (supporting)

• Detection delay for arrivals/departures (seconds).

• Flicker rate: false toggles per slot per hour.

• Per-slot F1 score for occupied class on a labeled sample.

• Robustness: measure performance at midday vs long shadows vs evening.

11. Deployment checklist (field-ready)

• Mount stability checked; camera field-of-view covers all visible slots; focus locked.

• Cable strain relief, drip loops, and enclosure sealing verified.

• Thermal test: run for 1 hour; confirm CPU temperature remains within safe limits.

• NTP synchronized; MQTT credentials provisioned; topics verified.

• Baseline calibration v1 saved and backed up; ROI overlay verified live.

• Outage test performed; buffer replay validated; dashboard shows continuous timeline.

Appendix A. Recommended repository layout

Keep the project reproducible by structuring code and artifacts clearly. Version calibration artifacts and configs alongside code.

```
repo/
  edge/
    services/ (capture, inference, telemetry, health, config)
    configs/ (camera_config.json, mqtt.json, thresholds.json)
    calibration/ (fass_slots_v1.json, homography_v1.json, roi_mask.png)
    buffer/ (sqlite queue, logs)
  server/
    mqtt/ (broker config, ACLs)
    db/ (schema, retention policies)
    dashboard/ (map overlay, charts, ops panel)
  docs/ (this document, figures, evaluation reports)
```