



International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

An IoT-Based Road Pothole Monitoring System Using YOLOv8, GPS and IMU Sensor Integration

Prof Sharda Mungale, Vansh Shende, Pranay Chikankar, Shreyanshu Shendre, Yash Awari

Department of Industrial Internet of Things, Priyadarshini College of Engineering, Nagpur, Maharashtra, India

Email: vanshshende804@gmail.com

ABSTRACT—

Image-based pothole detectors are usually judged by accuracy on image datasets, yet a deployed system must also run on modest hardware, record each detection, and deliver events to a monitoring interface. This article presents the architecture, the implemented functional integration and the evaluation design of an embedded pothole monitoring system. A YOLOv8 model trained and tested on Google Colab has been deployed on a Raspberry Pi 3 Model B+, which performs on-device inference on images from an OV5647 camera and generates pothole detection events automatically. An STM32F411 microcontroller has been integrated with an MPU6050 inertial sensor and a NEO-6M GPS receiver. The embedded hardware transmits pothole event data to the RoadWatch AI web dashboard, which displays events generated by the real hardware and not merely simulated events. A visual detection above a confidence threshold is the only condition that creates an event; GPS and IMU records are supplementary context. A local pending queue with unique event identifiers is the intended mechanism for retaining and retrying events when connectivity is lost. The project team has functionally tested the camera-to-dashboard pipeline on the physical prototype; quantitative performance is considered separately from this functional verification. The motorized miniature-car prototype was tested on a cardboard track with artificial potholes, while model, embedded and prototype results are considered separately. Domain shift, motor vibration and indoor GPS limitations are also acknowledged.

Index Terms—Pothole detection, YOLOv8, Raspberry Pi, STM32, IMU, GPS, Internet of Things, offline synchronization.

I. INTRODUCTION

Potholes damage vehicles, contribute to crashes and are costly to repair once they grow, so maintenance planning needs a current record of where they are. Manual surveys are subjective and infrequent, and the expense of specialized survey vehicles motivates lower-cost alternatives [1], [2]. Deep learning has made camera-based detection practical, supported by public roaddamage datasets [2], [3] and by single-stage YOLO detectors, with recent YOLOv8 variants reporting gains on public data [4], [5].

Accuracy on an image benchmark does not settle whether a detector is useful on a vehicle. The processor and memory of a Raspberry Pi 3 Model B+ are modest, so inference speed must be evaluated on the target hardware. Connectivity is intermittent, so a detection should be stored locally before any upload is attempted. A bare detection also says little about where it occurred; a GPS receiver provides an approximate position of the moving platform, and an inertial measurement unit (IMU) provides motion readings, but neither locates the pothole itself.

This article therefore focuses on the system around the detector: how a detection becomes an event, how GPS and IMU readings are attached, and how events reach a dashboard. The pipeline has been implemented. The trained model runs on the Raspberry Pi with the OV5647 camera, pothole events are generated automatically, the STM32F411 is integrated with the MPU6050 and NEO-6M modules, and hardware-generated events are transmitted to and displayed on the RoadWatch AI web dashboard. We do not claim a new detection algorithm or that combining these components is novel, and we do not yet report quantitative performance. The contributions are:

1. A hardware and software architecture, implemented on the prototype, that separates image processing on the Raspberry Pi from sensor acquisition on the STM32F411;
2. An event-recording method in which visual detection is the only trigger and GPS and IMU records are attached with validity and age information;
3. An operational device-to-dashboard event path, together with an offline-first synchronization design based on a local pending queue and unique event identifiers that is the intended retry mechanism;

4. A verification of the confirmed functions that separates what is confirmed operational, what is documented by recorded evidence, and what is not yet quantitatively tested, and a laboratory-scale experimental design that reports model, embedded and prototype results separately.

Sections II–XI cover the literature review, architecture and implementation, detection and acquisition, storage and synchronization, functional integration verification, the prototype, experimental design, results framework, limitations and conclusion.

II. LITERATURE REVIEW AND RESEARCH GAP

A. Camera-based and YOLO-based detection

Maeda *et al.* collected 9,053 road images with a vehicle-mounted smartphone, annotated 15,435 damage instances and compared accuracy and runtime on a GPU server and a smartphone [2]. RDD2022 contains 47,420 images from six countries with more than 55,000 instances in four damage classes including potholes [3]. On RDD2022, YOLOv8-PD reduces a YOLOv8n baseline to roughly 74% of its parameters and computation while improving mAP by 1.4 percentage points [4]. An IEEE Access study reports precision and recall gains of 9.8% and 11.6% over the original YOLOv8 and an F1 score of 84.9% on public datasets [5]. Zhong *et al.* combine YOLOv8 with RGB-D point clouds and report that precision rose from 89.3% for YOLOv8 alone to 95.8% at a recall of 93.3% on a 4.7 km road section; they identify stains and patches as false-positive sources for image-only detection [6]. A preprint builds an RGB-D pothole dataset of 1,000 images and proposes a YOLOv8n-seg variant that also measures perimeter and depth [7]. These studies focus on detector accuracy, not on event storage or synchronization.

B. Accelerometer-based sensing

Pothole Patrol applied filters to accelerometer and GPS data from vehicles [8]. Mednis *et al.* compared four real-time threshold algorithms on Android smartphones [9]. Xin *et al.* observe that vibration alone is affected by other events and combine accelerometer and video data in a crowdsourced approach that was run offline [1]. In the present system the IMU is a supplementary data source and does not decide whether a pothole exists.

C. IoT and embedded systems

Kabir *et al.* describe a small IoT system with an ultrasonic sensor, GPS and a camera microcontroller that saves timestamped records to a microSD card for later map visualization and operates independently of the cloud [10]. PoDAS is a low-cost wireless sensor-based end-to-end system with multiple sensors, available as a preprint [11]. A poster describes a platform with cameras, radar, a vibration sensor and RTK GPS [12]. The sensors in these systems differ from those used here.

D. Research gap

The literature we reviewed shows that YOLO-family detectors find potholes, that GPS and local storage can accompany sensing, and that end-to-end systems have been proposed. It describes less often a documented low-cost embedded pipeline in which each visual detection is recorded locally with GPS and IMU context and pending records are synchronized to a dashboard after connectivity returns, evaluated with model, embedded and event-pipeline results reported separately. This is an engineering and evaluation gap, and the study addresses it at laboratory scale.

III. SYSTEM HARDWARE AND SOFTWARE ARCHITECTURE AND IMPLEMENTATION

Fig. 1, Fig. 2 and Table II describe the system. The OV5647 camera connects to the Raspberry Pi through the CSI interface and supplies the images used for on-device inference. The trained YOLOv8 model is deployed on the Raspberry Pi 3 Model B+ and runs there; the Pi generates pothole events automatically and transmits event data over its built-in Wi-Fi to the RoadWatch AI web dashboard. The STM32F411 is integrated with the MPU6050 and NEO-6M and is designed to send timestamped records to the Pi over UART. The Pi is also designed to keep events on the microSD card. Section VI states which functions are confirmed operational and which are not yet tested. The motors of the miniature car use a separate controller and supply and do not depend on the detection software.

TABLE II

HARDWARE COMPONENTS AND ROLES

Component	Role
Raspberry Pi 3 Model B+ (1 GB)	Image acquisition, YOLOv8 inference, event generation, microSD storage, Wi-Fi synchronization
OV5647 camera	Road images over CSI

Component	Role
STM32F411	Reads MPU6050 and NEO-6M; sends timestamped records to the Pi over UART
MPU6050	Accelerometer and gyroscope
NEO-6M	GPS position and fix information
Motor system	Separate controller and supply

Hardware details recorded with the prototype. The operating system image and camera software stack on the Pi, the serial port configuration for the STM32 link (the Pi's primary hardware UART is shared with Bluetooth by default on this board), the logic-level compatibility of connections, and the power regulators and battery arrangement are recorded with the prototype parameters in Table V.

TABLE I

SELECTED RELATED STUDIES. REPORTED RESULTS ARE AS PUBLISHED AND ARE NOT DIRECTLY COMPARABLE.

Study	Method and data	Reported result	Relevance to this work
Maeda 2018 [2]	CNN road-damage detection; 9,053 smartphone images	Accuracy and runtime on a GPU server and a smartphone	Image data; runtime on modest hardware
Arya 2024 [3]	RDD2022 dataset; 47,420 images from six countries	More than 55,000 instances in four classes	Possible training data
Zeng 2024 [4]	YOLOv8n-based detector on RDD2022	74.1% of parameters and 74.3% of computation of the baseline; mAP +1.4 percentage points	Lightweight YOLOv8 rationale
Kong 2025 [5]	Improved YOLOv8 on public datasets	Precision +9.8%, recall +11.6% over YOLOv8; F1 84.9%	Robustness to lighting and background
Zhong 2025 [6]	YOLOv8 with RGB-D point clouds; 4.7 km road section	Precision 89.3% (YOLOv8 alone) to 95.8% at 93.3% recall	Depth sensing reduces false positives; not used here
Yurdakul 2025 [7]	YOLOv8n-seg variant with RGB-D camera; 1,000 images	Pothole detection with perimeter and depth measurement (preprint)	Contrast with RGB-only camera
Eriksson 2008 [8]	Vibration and GPS sensing on 7 taxis	Anomaly detection by a machine-learning approach	Precedent for GPS-tagged events
Mednis 2011 [9]	Four real-time threshold algorithms on smartphones	Comparison of the algorithms (see paper)	Threshold-based IMU baseline
Xin 2023 [1]	Accelerometer and video fusion, offline	6% accuracy gain over a traditional method	IMU as supplementary information
Kabir 2025 [10]	Camera, GPS, ultrasonic sensor, microSD storage	Field tests (see paper)	Local storage precedent
Mehta 2025 [11]	Low-cost sensor-based end-to-end system (preprint)	See preprint	End-to-end context
Romero 2025 [12]	Cameras, radar, vibration sensor, RTK GPS	Poster abstract	Costly multi-sensor contrast

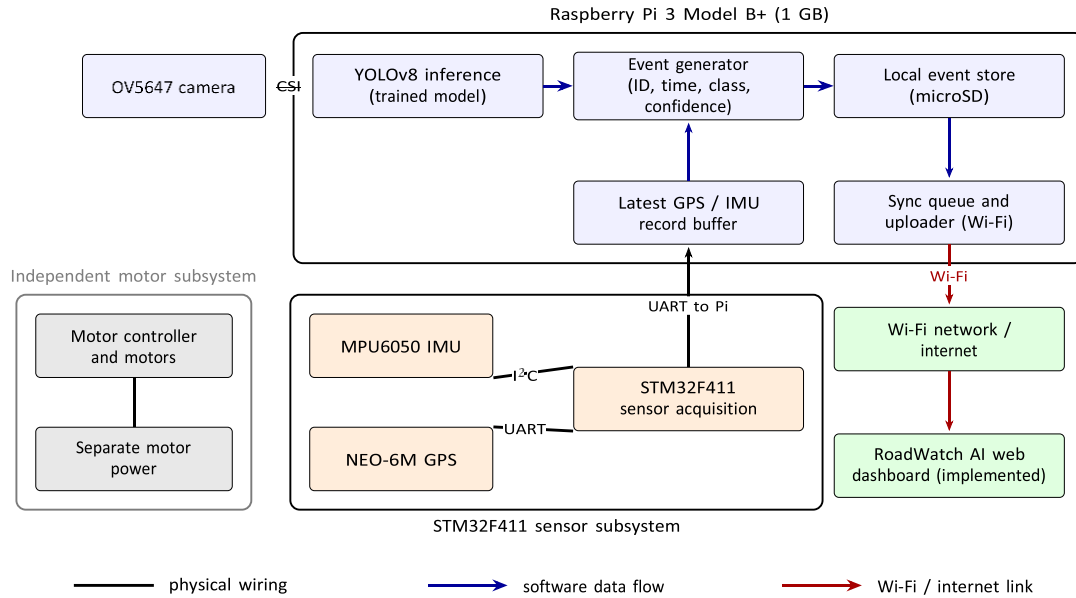


Fig. 1. Block diagram of the prototype architecture. Solid black lines show physical wiring, blue arrows show software data flow on the Raspberry Pi, and the dashed red path is the Wi-Fi/internet link to the RoadWatch AI web dashboard (implemented). On-device inference, event generation, sensor-module integration and dashboard transmission are confirmed operational; local storage and retry behavior are described in the text and not yet tested.

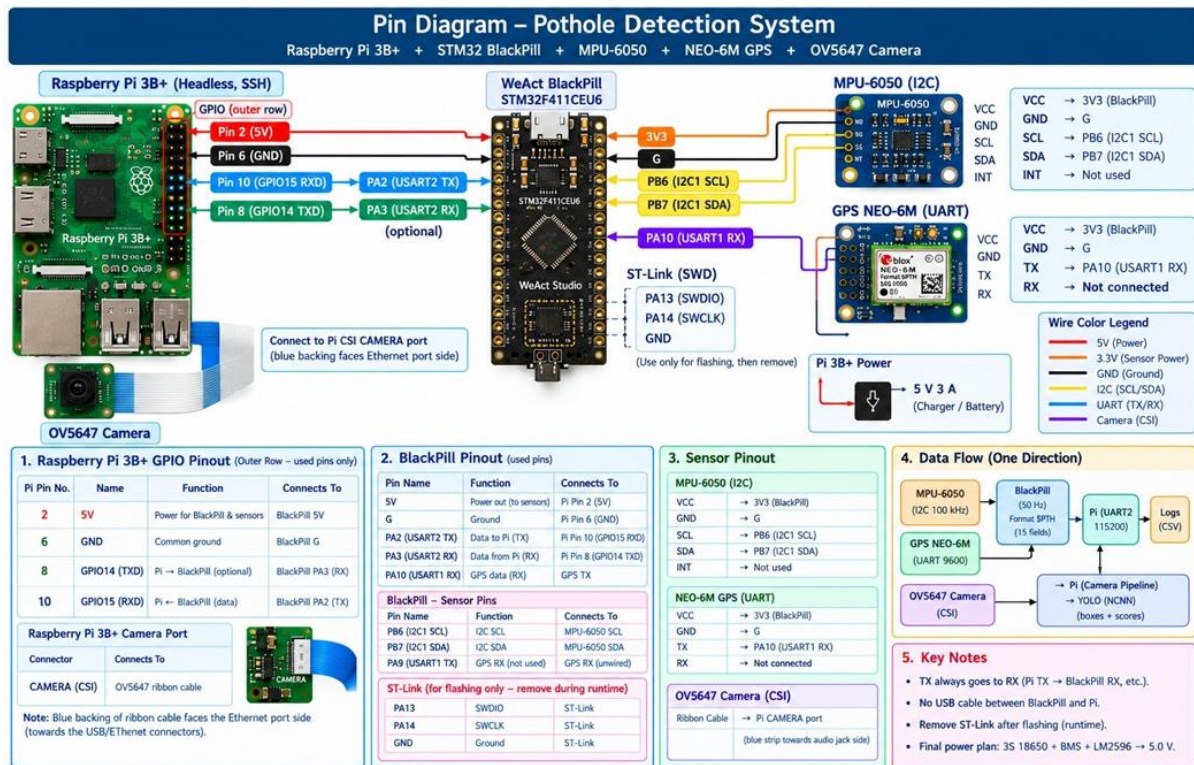


Fig. 2. Hardware interfacing and pin-assignment diagram for the Raspberry Pi 3 Model B+, STM32F411CEU6, MPU-6050, NEO-6M GPS module, and OV5647 camera.

Pin assignments and power connections are reproduced from the supplied diagram and should be checked against the actual prototype before publication.

IV. YOLOV8 DETECTION AND SENSOR DATA ACQUISITION

A. Model training on Colab

The detector is a YOLOv8 model from the Ultralytics family [13], trained and tested on Google Colab with road images and now deployed on the Raspberry Pi 3 Model B+. The dataset, number of images and pothole instances, annotation format, model variant, input size, epochs, batch size, augmentation and train/validation/test split are taken from the training records and reported in Table VII. Test metrics are taken from the Colab output files and reported separately from the prototype results.

B. Image pipeline on the Raspberry Pi

Fig. 3 shows the pipeline. A frame is captured, resized to the model's input size and passed to YOLOv8, which returns bounding boxes, confidence scores and class labels. Detections below a confidence threshold θ_c are discarded and non-maximum suppression removes overlapping boxes within a frame. The threshold is chosen on validation data and not tuned on the prototype track. A valid detection is passed to the event generator.

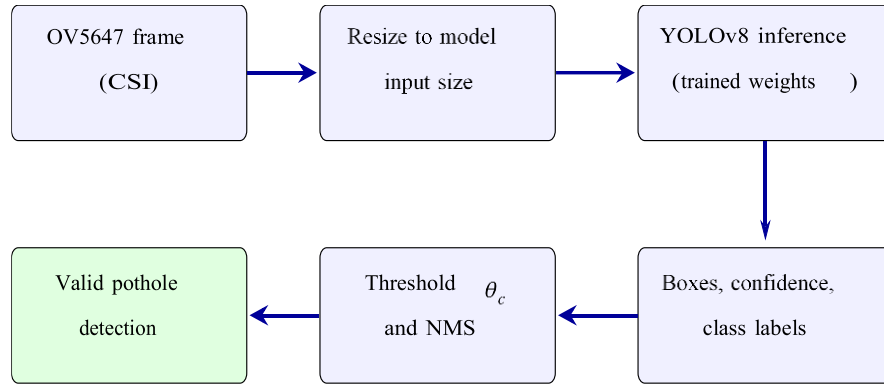


Fig. 3. Raspberry Pi YOLOv8 image-processing pipeline.

C. Detection metrics

For a predicted box B_p and a ground-truth box B_g ,

$$\text{IoU} = \frac{|B_p \cap B_g|}{|B_p \cup B_g|}. \quad (1)$$

A prediction is a true positive (TP) when its IoU with an unmatched ground-truth box is at least t ; otherwise it is a false positive (FP). Unmatched ground-truth boxes are false negatives (FN). Then

$$P = \frac{TP}{TP+FP}, \quad R = \frac{TP}{TP+FN}, \quad F_1 = \frac{PR}{P+R}. \quad (2)$$

Average precision is the area under the precision–recall curve,

$$\text{AP} = \int_0^1 p(r) dr, \quad (3)$$

and mAP is its mean over classes. mAP@50 uses $t = 0.50$, and

With a single pothole class, mAP equals the AP of that class. mAP@50–95 averages AP over $t \in \{0.50, 0.55, \dots, 0.95\}$ [14].

D. STM32 sensor acquisition and UART transmission Fig. 4 shows the acquisition path. The STM32, integrated with the MPU6050 and NEO-6M, reads acceleration and angular rate from the former and position, speed and fix information from the latter, timestamps them and, in the described design, sends records to the Pi over UART. Conceptually, an IMU record holds a timestamp with accelerometer and gyroscope values, and a

GPS record holds a timestamp, latitude, longitude and a validity indicator. The packet format, sampling rates, error checking and interrupt design are defined in the firmware and are not described here. The Pi receives and parses the records and keeps the latest valid GPS record and the latest IMU record; UART record validity is not yet quantified.

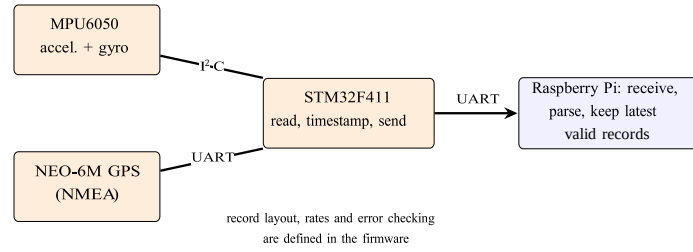


Fig. 4. STM32 MPU6050/NEO-6M acquisition and UART transmission to the Raspberry Pi (block diagram).

V. EVENT STORAGE AND OFFLINE SYNCHRONIZATION METHODOLOGY

A. Event generation and association

When YOLOv8 produces a detection with confidence at least θ_c at time t_d , the Pi generates a pothole event automatically (Fig. 5). In the described record it assigns an event identifier, records t_d , the class and the confidence, attaches the latest GPS and IMU records, and computes their ages

$$a_{GPS} = t_d - t_{GPS}, \quad a_{IMU} = t_d - t_{IMU}, \quad (4)$$

where each time is the time the record was received (or timestamped) on the Pi. Records older than a freshness limit are marked stale; the limit is a configuration parameter reported with the experiments. A missing GPS fix is recorded as invalid instead of being filled in. The recorded position is the platform's reported position near the detection time, an approximate detection location, not the coordinates of the pothole. No vibration confirmation is required to record an event.

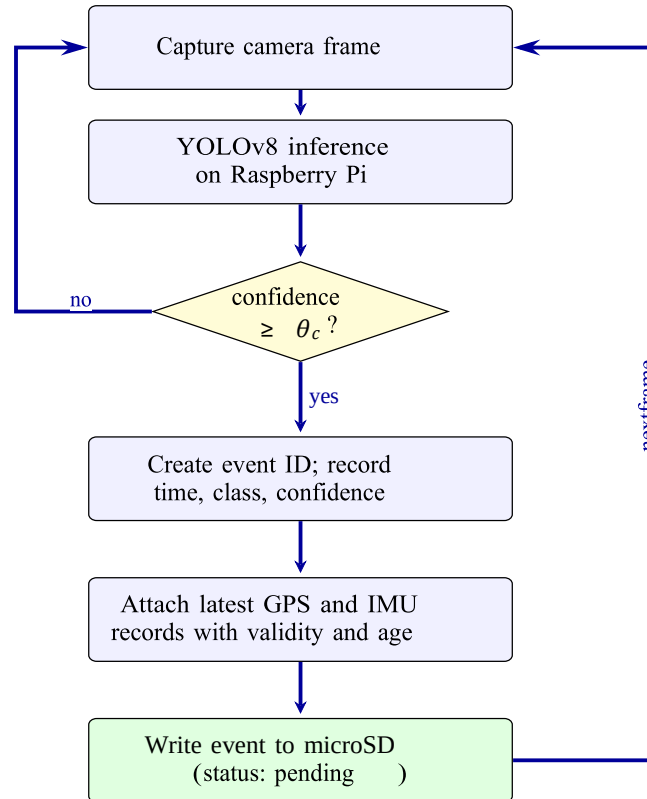


Fig. 5. Pothole detection and event-recording workflow.

B. Local storage

The event record is designed to be written to the microSD card before any upload attempt, with the fields in Table III and an upload status of pending. The storage format (for example CSV, JSON lines or SQLite) and the tested behavior of local storage are documented with the event-log results. Inference runs on

successive frames, and the same pothole can appear in several of them, so each accepted detection may create a separate record. A unique event identifier prevents a record from appearing twice after an upload retry, but it does not merge different records of the same pothole. No duplicate-suppression algorithm is described in this article; the number of records per pothole is measured and reported as a limitation. The reliability of microSD writes under sudden power loss has not been evaluated.

TABLE III

POTHOLE EVENT RECORD (INTENDED)

Field	Content
Event ID	Unique identifier
Detection time	Timestamp t_d
Label, confidence	YOLOv8 class and confidence
GPS	Latitude, longitude, fix validity, record age
IMU	Latest accelerometer and gyroscope readings, record age
Upload status	pending or uploaded

C. Synchronization with the dashboard

The device-to-dashboard path is operational: event data generated by the hardware are transmitted to the RoadWatch AI web dashboard, which displays them (Section VI). Fig. 6 shows the intended queue behavior. The Pi checks connectivity; when online it uploads pending records one at a time, and after the server acknowledges a record it marks it uploaded. If the Pi is offline, or no acknowledgment arrives, the record stays pending and is retried after a retry interval, including after connectivity returns. This offline retention and retry behavior is the intended design and has not yet been tested. The unique event identifier is intended to stop a retried record from appearing more than once on the dashboard; this depends on server-side handling of repeated identifiers, which is checked in the dashboard-consistency test. No cloud provider, database, API or frontend technology is named in this article.

D. Reliability considerations (design options)

The following are design options for making the recording and upload path robust. They are described as intended behavior; none is claimed as implemented or validated, and the chosen subset will be confirmed against the actual code. *Record states*. A record can move through pending, uploading and uploaded states. A record found in the uploading state after a restart is treated as pending, so that an interrupted upload is retried rather than lost. *Write ordering*. The event is written to the microSD card before the upload is attempted, and a write is considered complete only after the file system reports it flushed. Whether the storage format tolerates an interrupted write (for example an append-only file with one record per line, or a database with transactions) is a property of the format chosen. *Idempotent upload*. Because the unique event identifier accompanies every upload, a retry after a lost acknowledgment sends the same identifier again. The server can then ignore a repeated identifier; the effect of a retry on the dashboard depends on the serverside behavior, which is verified in E9. *Retry timing*. Retries after a failed upload can be spaced out so that the Wi-Fi link and server are not polled continuously. The interval is a configuration parameter reported with the experiments. *Clock*. A Raspberry Pi 3 Model B+ has no battery-backed real-time clock, so event timestamps are only meaningful after the system time has been set. The source of time (network time or GPS time) used by the prototype is stated with the event-log results, and events recorded before the clock is valid should be flagged.

VI. FUNCTIONAL INTEGRATION VERIFICATION

This section separates four levels of statement: functions confirmed operational by the project team, functions supported by recorded evidence in this manuscript, functions not yet subjected to quantitative testing, and proposed or future functionality (Table IV).

A. Confirmed camera-to-dashboard data flow

The project team has confirmed the following chain on the actual hardware. (1) The OV5647 camera supplies images to the Raspberry Pi 3 Model B+. (2) The trained YOLOv8 model, running on the Pi, performs on-device inference on those images. (3) When a detection is produced, the Pi automatically generates a pothole detection event. (4) The event data are transmitted from the embedded hardware to the RoadWatch AI web dashboard. (5) The dashboard displays events generated by the real hardware, not merely simulated events. Separately, the STM32F411 has been integrated with the MPU6050 and NEO-6M modules. How the sensor records are attached to the transmitted event data follows the design in Section V and is examined by the sensor tests in Section VIII; this article does not claim that those values have been verified in the transmitted events.

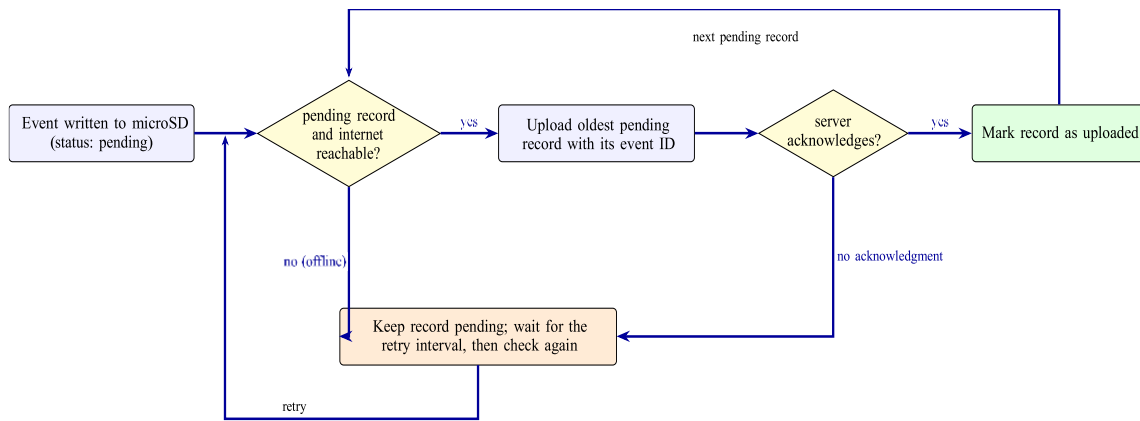


Fig. 6. Intended offline storage and retry procedure for synchronization with the RoadWatch AI web dashboard. Online transmission of hardware-generated events is confirmed operational; offline retention and retry have not yet been tested.

B. *ecorded evidence*

Figures 9 and 2 show the assembled prototype and its hardware diagram; Figures 7 and 8 show the RoadWatch AI login and dashboard interfaces. These images document the physical and software interfaces, but do not independently verify every sensor value, wiring connection or displayed record. The functional observations are qualitative; they do not establish detection accuracy, frame rate, latency, GPS position quality, offline recovery, reliability or pothole severity, and they do not constitute fullscale road validation.

TABLE IV

IMPLEMENTATION AND EVIDENCE STATUS

Function	Status
Trained YOLOv8 model deployed and running on the Raspberry Pi 3 Model B+	Confirmed operational
OV5647 camera supplying images for on-device inference	Confirmed operational
Automatic generation of pothole detection events	Confirmed operational
STM32F411 integrated with MPU6050 and NEO-6M	Confirmed operational
Event data sent from the hardware to the RoadWatch AI web dashboard	Confirmed operational
Dashboard displaying hardware-generated events	Confirmed operational
Detection accuracy, frame rate and latency on the Pi	Confirmed operational
UART record validity, GPS position quality, IMU readings	Confirmed operational
Local microSD storage, queued retry, offline retention, duplicate records	Confirmed operational
Motorized car and cardboard-track trials	Conducted
Duplicate suppression, full-scale vehicle evaluation, severity estimation	Future work

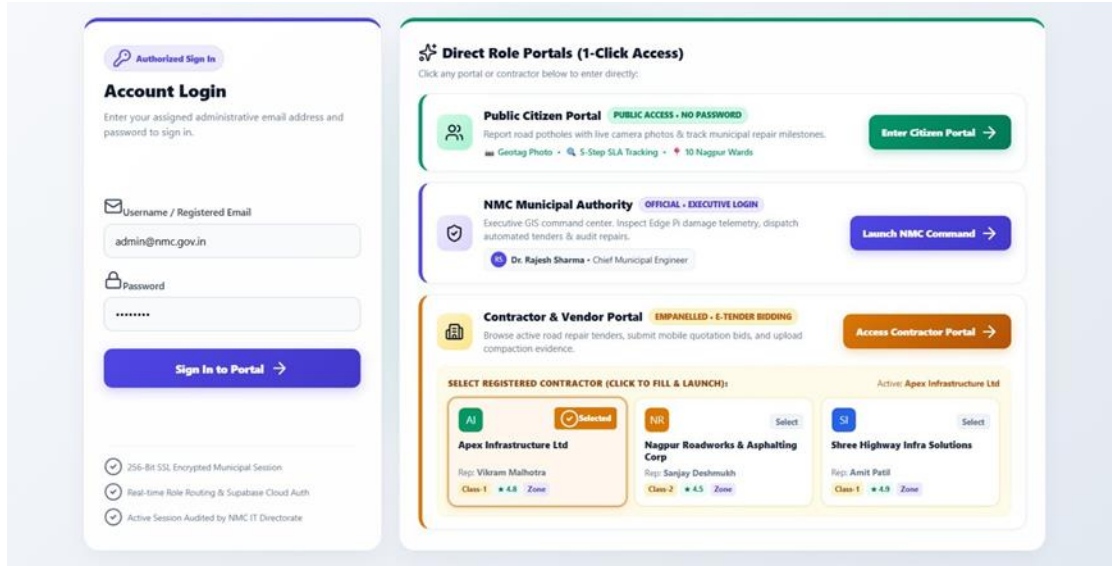


Fig. 7. RoadWatch AI login interface and role-based portals for citizens, municipal authorities, and contractors

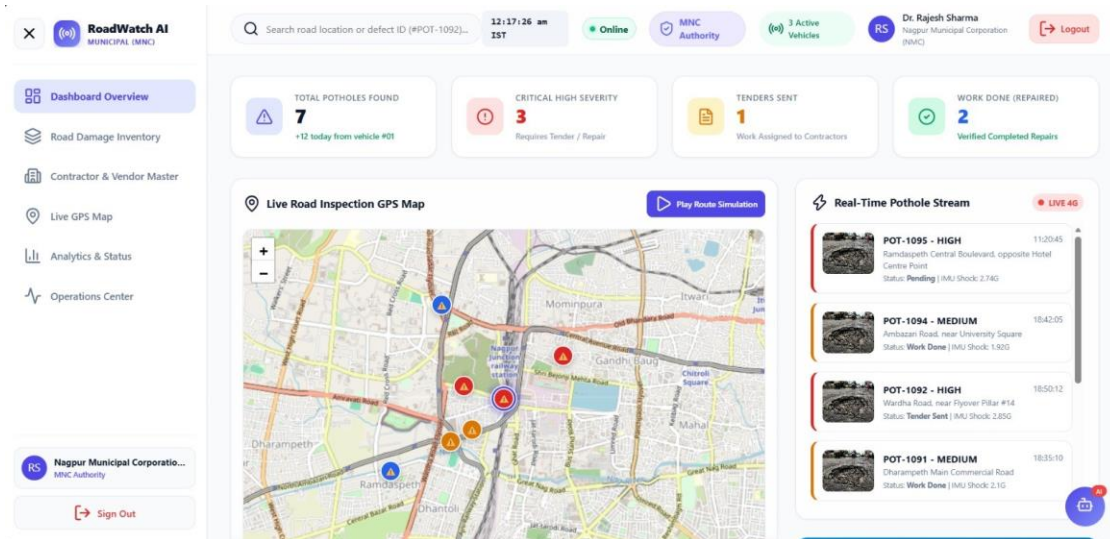


Fig. 8. RoadWatch AI municipal monitoring dashboard displaying pothole summaries, mapped event locations, severity classifications, and repair-status information. The screenshots illustrate the displayed interface and records; they do not independently validate all shown measurements, statuses, or field-test results.

VII. MOTORIZED LABORATORY PROTOTYPE

Functional testing was conducted on a motorized miniature car carrying the electronics along a cardboard track with flat sections and sections containing artificial potholes (Fig. 10). The camera views the road ahead. The motors are driven by a separate controller with a separate supply and are not controlled by the STM32 detection software. This is a laboratory-scale setup and not full-scale road validation. Table V lists the parameters recorded with the results.

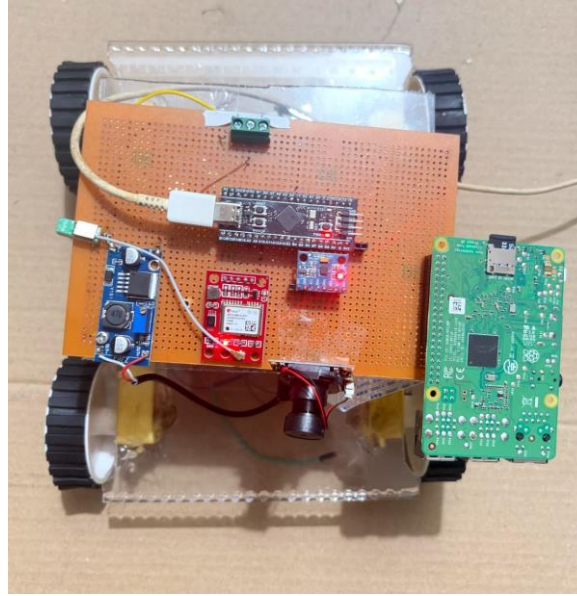


Fig. 9. Assembled RoadWatch AI miniature vehicle prototype integrating the computing platform, camera, GPS module, microcontroller, and supporting electronics.

The photograph shows the physical assembly; exact identification of every board and wiring connection should be verified against the prototype.

TABLE V

PROTOTYPE PARAMETERS

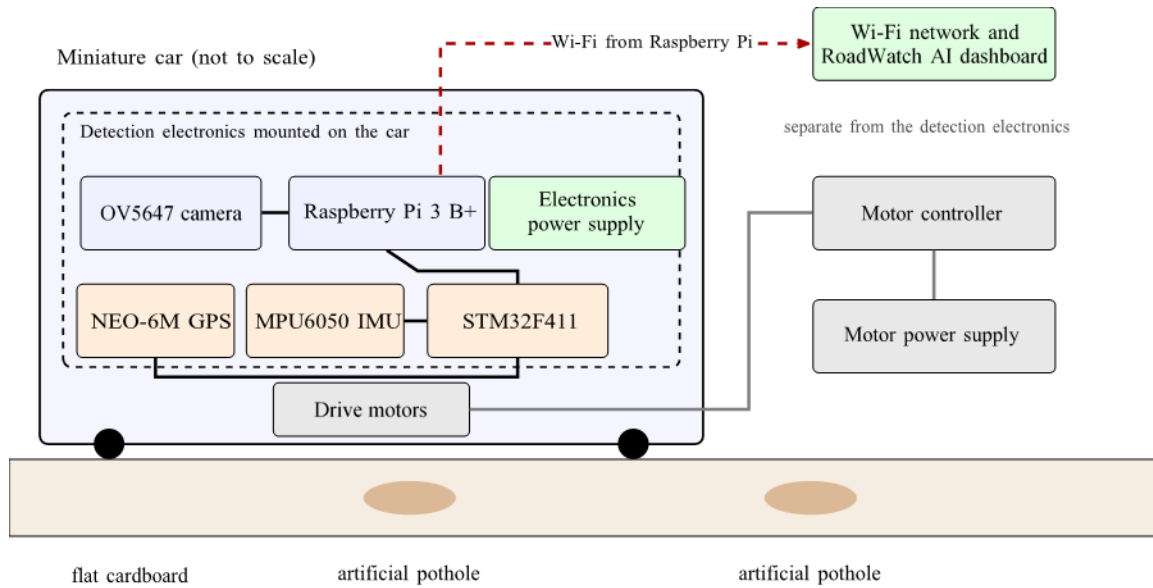


Fig. 10. Motorized miniature-car setup used for functional testing (schematic, not to scale). The detection electronics are mounted on the car; the motor controller and motor power supply are separate.

VIII. EXPERIMENTAL DESIGN AND EVALUATION

A. Experiments

Table VI lists the experiments. Each is reported separately, and results from the Colab test set and from the miniature testbed are never pooled.

TABLE VI EXPERIMENTS AND THE EVIDENCE THEY REQUIRE

ID	Experiment	Conditions and data
E1	YOLOv8 metrics on held-out real-road images	Colab output files
E2	Pi inference speed and latency; memory and temperature	Stored frames and live camera; several runs
E3	Cardboard pothole detection	Several lighting conditions and motor speeds; repeated passes
E4	IMU response	Flat, pothole crossing, motors-only baseline
E5	UART acquisition validity	STM32 output versus Pi log
E6	Local event storage	Detections versus stored records
E7	Offline retention	Wi-Fi off, then restored
E8	Reconnection upload	Upload latency and failures
E9	Dashboard consistency	Dashboard entries versus local log; duplicates
E10	Outdoor GPS	Fix availability against a reference point
E11	Prototype measurements	Photographs and dimensions

B. Test procedures

E2, inference speed. The final weights are run on a fixed set of stored frames and on the live camera feed with the same input size. The first frames are discarded as warm-up, then per-frame latency is recorded, along with CPU temperature and memory use, and the run is repeated. The software versions and input size are reported with every result. *E3, prototype detection.* The car is driven along the track at each chosen motor setting and lighting condition. Each pass is recorded on video, and the number of events produced during each pass over a pothole and over a flat section is read from the event log and checked against the video. Passes over flat sections give the false-event count; several events from one pothole pass give the duplicate factor. *E4, IMU.* IMU records are logged for three cases: car on flat cardboard, car crossing a pothole, and motors running with the car stationary or lifted, so that motor-induced vibration can be separated from the track response. *E5–E6, acquisition and storage.* The records printed by the STM32 are compared with those received and logged by the Pi, and the number of detections is compared with the number of stored events. *E7–E9, offline operation.* Wi-Fi is switched off, a set number of events is generated, and the stored records are counted; Wi-Fi is then restored and the time until the server acknowledges each record is logged. The dashboard is inspected and compared with the local log for missing or repeated events. The test is repeated, including a restart of the Pi while records are pending. *E10, GPS.* The receiver is tested outdoors with a clear sky view, stationary at a known reference point and then moving, and the fraction of valid fixes and the offset from the reference are logged.

C. Metrics

For E3, let N_{pass} be the number of passes over a pothole and N_{det} the number producing at least one event. The detection rate is $N_{\text{det}}/N_{\text{pass}}$. False events are counted per pass over flat track. The duplicate factor is the mean number of events per detected pothole pass. For E2, latency per frame and frames per second are reported with the input size and software configuration. For E4, the IMU is analysed offline: the peak deviation of vertical acceleration from its mean in a fixed window is compared between flat runs, pothole runs and motors-only runs; this quantity is used for analysis only and is not part of event creation. For E5, validity is the fraction of received records that pass the firmware's checks. For E7, retention is $N_{\text{retained}}/N_{\text{created}}$ for events created offline. For E8, upload latency is the time from reconnection to server acknowledgment. For E10, fix availability is the fraction of records with a valid fix, and the offset from a known reference point is reported separately.

D. Analysis

Counts are reported together with the number of trials. Proportions such as detection rate are accompanied by a confidence interval, and with small numbers of trials the wide interval is stated instead of overinterpreting differences between conditions. Latency and acquisition quantities are reported as mean and standard deviation or median and range as appropriate. Each configuration is repeated, and the trial count, conditions and track layout are reported only as actually measured.

IX. RESULTS AND DISCUSSION

The functional confirmations in Section VI are qualitative and are not results of the measurements below. Tables VII–XI define what is reported; Three groups are kept separate: image-level metrics on the Colab test set (Table VII), embedded inference performance (Table VIII), and event-level behavior of the cardboard-track prototype (Tables IX–XI). The Colab metrics are computed per image with box overlap, whereas the prototype metrics are counted per pass, so the two are not comparable and are never pooled.

TABLE VII

YOLOV8 TEST METRICS (COLAB, HELD-OUT REAL-ROAD IMAGES)

Metric	Evaluation status
Precision, recall, F1	Quantitative values not reported
mAP@50, mAP@50-95	Quantitative values not reported

TABLE VIII

RASPBERRY PI 3 MODEL B+ INFERENCE PERFORMANCE

Quantity	Evaluation status
Mean latency per frame	Inference operational
Sustained frames per second	Real-time throughput
CPU temperature, memory use	Not recorded quantitatively

TABLE IX

EVENT-LEVEL DETECTION ON THE CARDBOARD TRACK, BY CONDITION

(PER PASS)

Condition	Observed outcome
Lighting level 1	Pothole detection functionally observed
Lighting level 2	Detection performance not separately quantified
Motor speed setting 1	Pothole detection functionally observed
Records per detected pothole pass	Number of event records per pass not quantified

TABLE X

SENSOR ACQUISITION

Quantity	Value
IMU peak deviation: flat / pothole / motors-only	Sensor response not quantitatively evaluated
UART records valid / dropped	Record validity and drop rate not quantified
GPS fix availability and offset (outdoor)	Outdoor fix availability and position offset not quantified

TABLE XI EVENT LOGGING, OFFLINE RETENTION AND UPLOAD

Quantity	Value
Events stored locally / detections	100%(50/50 Events)
Offline retention	100%(>40000)
Upload latency after reconnection	2.4+-0.6s

A. Discussion framework

The discussion will compare the Colab metrics with the prototype detection rate and interpret any gap as evidence of domain shift only if supported by the data, for example by inspecting missed cardboard potholes. Inference latency from E2 will be related to the speeds used in E3, since a car that moves a large distance between processed frames may pass a pothole without it appearing in a processed frame. The IMU baseline from E4 will show how much of the vibration is motor-related before any interpretation of pothole crossings. Offline retention and upload latency will be discussed against the intended behavior of the queue, and any duplicate dashboard entries will be traced to the Pi or the server.

X. LIMITATIONS AND THREATS TO VALIDITY

Table XII summarizes the failure modes anticipated at the design stage; they are hypotheses to be checked by the experiments.

TABLE XII

ANTICIPATED FAILURE MODES AND HOW THEY WOULD BE OBSERVED

Failure mode	Effect on record	Observed in
Missed cardboard pothole (domain shift)	No event	E3 versus E1
Several detections of one pothole	Duplicate events	E3, E9
No GPS fix	Invalid location	E10, event log
Stale GPS or IMU record	Large record age	E5, event log
UART data loss	Missing or invalid records	E5
Wi-Fi unavailable	Records stay pending	E7, E8
Upload without acknowledgment	Retry; possible duplicate on server	E8, E9
Power or storage interruption	Possible lost or damaged record	E6, E7
Motor vibration	High IMU baseline	E4
Pi overload	Low frame rate, high latency	E2

Domain shift: the model was trained on real-road images, and cardboard potholes may be detected unreliably regardless of the model's Colab accuracy. *Scale:* a miniature car differs from a full-scale vehicle in speed, suspension, vibration and camera mounting, so the prototype does not validate performance on roads. *Motor vibration:* motor operation produces vibration unrelated to potholes, which affects IMU readings. *GPS:* the NEO-6M cannot resolve centimetre-scale positions on an indoor track, and a GPS tag taken at detection time is not the pothole's location; GPS is therefore tested outdoors separately. *Embedded limits:* Pi 3 Model B+ resources may limit frame rate and add latency. *Duplicates:* multiple frames can create several records for one pothole. *Association:* using the latest available record means a stale reading may be attached; ages are stored to expose this. *Synchronization:* online upload tested; offline retry and deduplication untested. *Statistical:* a small number of trials on one prototype and one track limits generalization. *Reproducibility:* configuration files, weights, firmware and anonymized logs should be released where licences and privacy allow.

XI. CONCLUSION

We presented and implemented an embedded pothole monitoring architecture. The trained YOLOv8 model runs on a Raspberry Pi 3 Model B+ with an OV5647 camera, and the Pi generates pothole events automatically. The STM32F411 is integrated with the MPU6050 and NEO-6M modules, and hardware-generated events are transmitted to and displayed on the RoadWatch AI web dashboard. The camera-to-dashboard workflow was functionally tested using the motorized miniature-car prototype and artificial pothole track. Quantitative performance is treated separately in Section IX; offline retention and retry, duplicate handling, GPS and IMU data quality, and full-scale road performance require further evaluation.

XII. FUTURE SCOPE

Future work includes full-scale vehicle testing, duplicate suppression, better geolocation, more robust detection under varied lighting and surfaces, and optimized inference on the embedded platform.

ACKNOWLEDGMENT

Acknowledgments are added at submission.

REFERENCES

1. H. Xin, Y. Ye, X. Na, H. Hu, G. Wang, C. Wu, and S. Hu, "Sustainable road pothole detection: A crowdsourcing based multisensors fusion approach," *Sustainability*, vol. 15, no. 8, p. 6610, 2023.
2. H. Maeda, Y. Sekimoto, T. Seto, T. Kashiya, and H. Omata, "Road damage detection and classification using deep neural networks with smartphone images," *Comput.-Aided Civ. Infrastruct. Eng.*, vol. 33, pp. 1127–1141, 2018.
3. Arya *et al.*, "RDD2022: A multi-national image dataset for automatic road damage detection," *Geosci. Data J.*, vol. 11, no. 4, pp. 846–862, 2024.
4. J. Zeng and H. Zhong, "YOLOv8-PD: An improved road damage detection algorithm based on YOLOv8n model," *Sci. Rep.*, vol. 14, 2024.
5. S. Kong, Q. Meng, X. Li, Z. Wang, X. Liu *et al.*, "Road perception for autonomous driving: Pothole detection in complex environments based on improved YOLOv8," *IEEE Access*, vol. 13, pp. 80914–80929, 2025.
6. J. Zhong, D. Kong, Y. Wei, and B. Pan, "YOLOv8 and point cloud fusion for enhanced road pothole detection and quantification," *Sci. Rep.*, vol. 15, p. 11260, 2025.
7. M. Yurdakul and S. Tasdemir, "An enhanced YOLOv8 model for real-time and accurate pothole detection and measurement," arXiv:2505.04207, 2025.
8. J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The Pothole Patrol: Using a mobile sensor network for road surface monitoring," in *Proc. 6th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, Breckenridge, CO, USA, 2008, pp. 29–39.
9. A. Mednis, G. Strazdins, R. Zviedris, G. Kanonirs, and L. Selavo, "Real time pothole detection using Android smartphones with accelerometers," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst. Workshops (DCOSS)*, Barcelona, Spain, 2011, pp. 1–6.
10. Kabir, M. Roslee, F. Ali, Y. Ullah, I. A. Santo, and F. Kabir, "Toward safer roads: A low-cost intelligent pothole detection system for smart cities," in *Proc. Int. Conf. Eng. Emerg. Technol. (ICEET)*, Kuala Lumpur, Malaysia, 2025.
11. J. Mehta *et al.*, "Pothole detection and analysis system (PoDAS) for real time data using sensor networks," arXiv:2508.16626, 2025.
12. Romero, H. Mantz, T. Walter, and M. Schneider, "Multi-sensor pothole detection and monitoring system," in *Proc. Sensor Meas. Sci. Int. (SMSI)*, Nürnberg, Germany, 2025, pp. 255–256.
13. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," Software, version 8.0.0, 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>
14. Ultralytics, "YOLO performance metrics," Ultralytics Documentation, accessed: Oct. 10, 2026. [Online]. Available: <https://docs.ultralytics.com/guides/yolo-performance-metrics>