Yolo-Based Pothole Detection System for Road Safety and Maintenance

Aditya Wasnik[1], Archit Kanfade[2], Aniket Telang[3], Manswi Chekbandalwar[4], Dr. N. S. Ambatkar[5]
[1,2,3,4]*Department of Electronics and Telecommunication Engineering, Priyadarshini College of Engineering (An Autonomous Institute Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University), Nagpur, Maharashtra, India*
[5]*Professor, Department of Electronics and Telecommunication Engineering, Priyadarshini College of Engineering (An Autonomous Institute Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University), Nagpur, Maharashtra, India*

Abstract*—Potholes remain among the most hazardous manifestations of flexible-pavement distress, yet municipal surveys in many Indian cities still proceed on foot or by drive-by observation, and the resulting complaint records carry no depth, no area and no repair estimate. This paper presents a rover-mounted detection chain that couples edge inference with a serverless cloud backend and converts each sighting into a costed, route-tagged maintenance record. A Raspberry Pi 5 runs a fine-tuned YOLO detector over frames from a USB camera, while a downward-facing HC-SR04 sonar, read on GPIO 5 and GPIO 6 with three-sample median filtering and a bench-calibrated 40 mm mounting offset, supplies the depth term that imagery alone cannot provide. Detections clearing a 0.60 confidence gate are fused into a JSON payload and posted over HTTPS to an API Gateway endpoint in ap-south-1, where Lambda functions verify the survey state, compute volume, material and cost, and commit the record to DynamoDB; images reside in a private S3 bucket and reach the browser only as one-hour presigned URLs. Chassis motion is deliberately kept off the vision path: an ESP8266 in soft-AP mode serves a five-button control page and drives an L298N bridge, so a fault in the inference loop can never leave the rover motoring unattended. Route identifiers are minted by the backend state machine rather than by any satellite receiver, which removes GNSS hardware from the bill of materials without sacrificing route-wise accountability. Bench and field trials confirm the detection-to-dashboard pipeline end to end. The work demonstrates that maintenance-grade road records can be produced from commodity hardware and pay-as-you-go cloud services.

***Index Terms*—Edge inference, pothole detection, YOLO, serverless backend, ultrasonic ranging, route-wise prioritization.**

I. INTRODUCTION

Flexible pavements fail progressively. Water ingress into hairline cracks, raveling of the binder and repeated axle loading together open depressions that widen faster than any periodic inspection cycle can catch them [1], [2]. In municipal practice the depression is usually reported by a commuter long after it has become a hazard, and the report arrives without depth, without area and without any indication of the material quantity a patch would consume. The maintenance engineer is left to triage on anecdote.

Vision-based automation attacks the first half of this problem. A camera on a moving platform, paired with a trained detector, can flag distress continuously [1], [3], and the YOLO family makes such detection feasible on embedded hardware because localization and classification collapse into a single forward pass of one network [4]–[6]. What the detector cannot do is measure. A bounding box around a crater says nothing about whether the crater is two centimeters deep or twelve, and depth is precisely the quantity that separates a cosmetic skim from a structural repair. A second gap is organizational: most prototypes terminate at a screen on the rover or at a cloud bucket of images, leaving the route-level bookkeeping — which street, how many defects, how severe, how much cold-mix — to a human with a spreadsheet. A third gap is positional. Localization is routinely assumed to require a GNSS receiver, adding cost, continuous power draw and urban-

canyon uncertainty to a system whose genuine need is route-wise accountability rather than metre-level coordinates.

The detection chain presented here closes all three gaps with commodity parts. A Raspberry Pi 5 carries the inference load
[12]; a single downward-looking HC-SR04 supplies the depth term through time-of-flight ranging; and an AWS serverless backend turns every accepted detection into a structured, priced record bound to a cloud-minted route identifier. Motion is delegated to a separate ESP8266 that serves its own control page in soft-AP mode [13], so the control plane and the vision plane fail independently of one another. Neither a GNSS receiver nor a cellular modem appears anywhere in the bill of materials.

The remainder of the paper is organized as follows. Section 2 reviews related work and isolates the gaps addressed here. Section 3 develops the system: the edge pipeline, the sonar channel, the severity arithmetic, the backend state machine and the isolation of the control plane. Section 4 reports bench and field observations and discusses them against the literature. Section 5 concludes.

II. RELATED WORK

Maeda et al. showed that smartphone imagery collected during ordinary driving suffices to train damage detectors, establishing the low-cost data-acquisition paradigm on which most subsequent work rests [1]. The RDD2020 and RDD2022 releases generalized this across countries, pavement types and illumination regimes, and remain the standard benchmarks for supervised road-damage models [2], [3]. On the architecture side, single-stage detectors descended from YOLO trade a modest accuracy penalty for a large speed gain, which is why they dominate embedded deployments [4]–[7]. Zeng and Zhong trimmed YOLOv8n specifically for pavement distress and reported that channel reweighting recovers most of the accuracy lost to pruning
[8]; Chen et al. strengthened feature fusion for coarse, low-contrast surfaces [9]. IoT-oriented prototypes such as DeepBus [10]
and a smartphone-based real-time detector [11] prove that detections can leave the device, but both treat the cloud as a sink rather than as a bookkeeping layer, and neither attaches a physical measurement to the sighting.

Table 1 summarizes these threads. Read together, they leave three openings. Depth is either absent or obtained from range sensors whose cost exceeds the rest of the node; the cloud tier, where present, stores rather than computes; and localization is conflated with GNSS even when the downstream consumer only needs to know which surveyed stretch a defect belongs to. The present work answers with a sonar depth channel on the device, a computing backend that prices every record it accepts, and route identifiers minted by the survey state machine instead of by satellites.

Table 1: Summary of representative road-damage detection studies

Ref.	Platform / approach	Major contribution	Limitation noted
[1]	Smartphone imagery + deep CNN	Practical vehicle-mounted damage monitoring	Quality bound to acquisition conditions
[2]	RDD2020 dataset	26,336 images, four damage classes	Narrower geographic spread than later sets
[3]	RDD2022 dataset	Six-country multi-class benchmark	Domain variation across environments
[8]	YOLOv8-PD on embedded target	Lightweight pavement-distress detection	Validation on limited road types
[9]	Enhanced YOLOv8 fusion	Improved coarse-surface detection	Higher architectural complexity
[10]	DeepBus, ML + IoT sensing	Real-time remote reporting	No depth, cost or prioritization output
[11]	Smartphone real-time detector	On-device detection with GNSS tagging	GNSS-dependent, no maintenance record

III. SYSTEM DESIGN

3.1 Architecture at a glance

The system decomposes into three planes (Fig. 1). The sensing plane comprises the USB camera and the

HC-SR04, both terminated on the Raspberry Pi 5, which performs acquisition, inference, depth fusion, severity assignment and the HTTPS uplink. The control plane comprises the operator's phone, the ESP8266 soft-AP control page and the L298N bridge driving four DC gear motors; it carries no sensor and no cloud credential. The presentation plane is a static single-page application hosted on S3 that polls the API and renders records, images and route rankings. Keeping the planes disjoint is a deliberate reliability decision, discussed in Section 3.6.

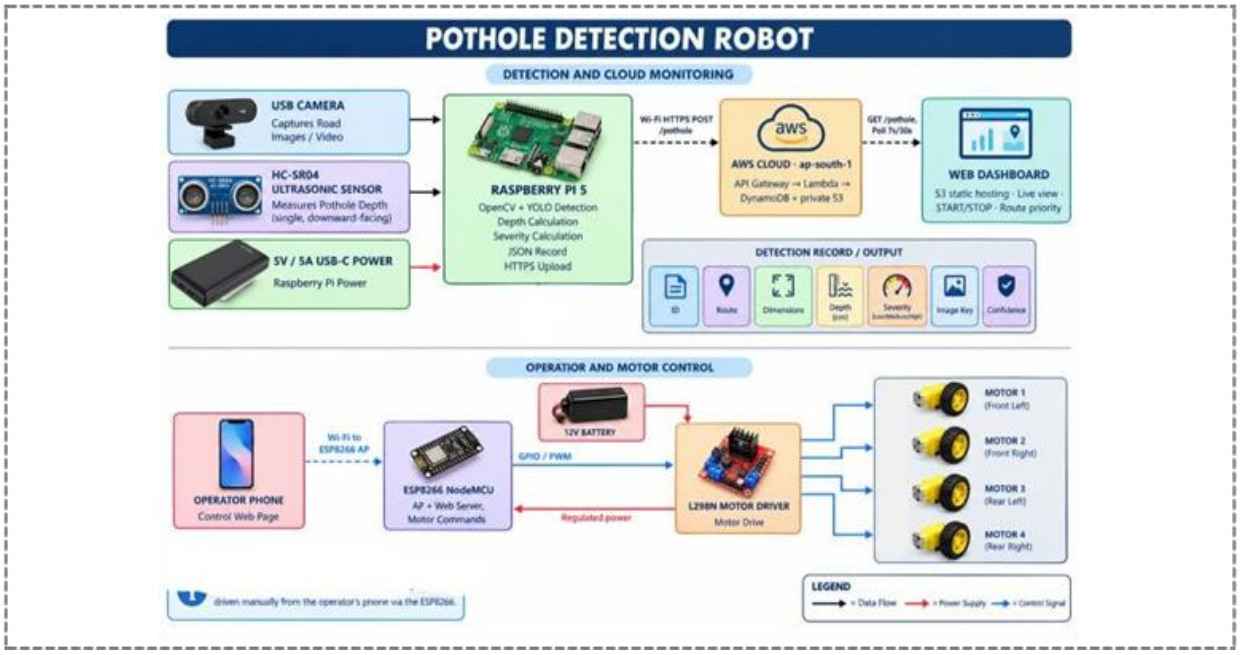


Figure 1: Block diagram of the proposed system

3.2 Edge vision pipeline

Frames are captured through the OpenCV video-capture interface, resized and normalized to the tensor geometry expected by the fine-tuned weights, and passed through a single YOLO forward pass that returns candidate boxes with class confidences. A detection is admitted only when

$$C \geq C_{th}, \text{ with } C_{th} = 0.60 \qquad (1)$$

The gate value was selected on validation splits during development; epoch counts, input resolution and the augmentation schedule are omitted here for brevity and are reported in the companion thesis. Admitting on confidence alone would reproduce the weakness of purely visual systems, so the admitted box immediately triggers the sonar channel.

3.3 Sonar depth channel

The HC-SR04 is mounted facing the pavement beneath the chassis. Each trigger pulse yields a round-trip time from which the standoff distance follows as

$$d = v \cdot t / 2 \qquad (2)$$

where v is the speed of sound in air and t the measured round-trip propagation time. With d_r the standoff over intact pavement and d_p the standoff over a depression, the estimated depth is

$$D_p = | d_p - d_r | \qquad (3)$$

Three pings are taken per reading and the median retained; returns beyond 4 m are discarded as spurious. A fixed 40 mm offset compensates the mounting geometry. This offset is a bench-calibrated constant of our fixture, not a datasheet value, and it must be re-derived if the sensor is re-seated. Because the HC-SR04 echo line swings to 5 V, a resistive divider level-shift it before the Pi's 3.3 V-tolerant GPIO.

3.4 Severity taxonomy and maintenance arithmetic

Severity is assigned on the device from the fused measurement, using the depth-and-dimension rules of Table 2. Confidence plays no part in this step; it governs admission only.

Table 2: Severity taxonomy implemented on the device

Class	Condition
Critical	depth ≥ 10 cm or longest side ≥ 50 cm
High	depth ≥ 5 cm or longest side ≥ 35 cm
Medium	depth ≥ 2 cm or longest side ≥ 20 cm
Low	otherwise

The backend then prices the defect. Volume follows the prismatic approximation

$$V = L \cdot W \cdot D \quad (4)$$

converted to cubic metres by

$$V_{m^3} = V_{cm^3} / 1{,}000{,}000 \quad (5)$$

Base material mass, wastage and final requirement collapse to

$$M = V_{m^3} \cdot \rho \cdot (1 + w/100) \quad (6)$$

and the repair estimate to

$$Cost = M \cdot p \quad (7)$$

where ρ is the configured material density, w the wastage percentage and p the unit price per kilogram. These three constants are project configuration, not universal pavement constants, and the dashboard exposes them as such.

3.5 Serverless backend and survey state machine

All traffic enters through a single HTTP API in ap-south-1 (Table 3). Pothole Calculator first reads the Survey State table; if the survey is not ACTIVE, or no route identifier exists, the payload is rejected with a conflict response and nothing is written. When the survey is active, the function takes the official route identifier from Survey State — the rover never invents one — evaluates Eqs. 4–7, performs a head-object check against the private image bucket, preserves the device-supplied severity and commits the record to Pothole Data. Pothole Reader scans Pothole Data and, for records carrying an image key, returns a preassigned GET URL valid for one hour; large Base64 blobs are deliberately avoided because they strain Lambda response limits, and the private object is never exposed to the browser. Survey Control implements the lifecycle: START mints the next sequential Route-N and latches ACTIVE, STOP closes the route and returns IDLE, STATUS reports both.

Table 3: API routes and their Lambda handlers

Method / route	Handler	Responsibility
POST /pothole	Pothole Calculator	Gate on survey state, compute Eqs. 4–7, commit record
GET /pothole	Pothole Reader	Return records with preassigned image URLs
POST /survey/start	Survey Control	Mint Route-N, latch ACTIVE
POST /survey/stop	Survey Control	Close route, return IDLE
GET /survey/status	Survey Control	Report state and active route

The dashboard polls on a timer rather than a push channel: seven seconds while a survey is active, thirty seconds while idle. Its START and STOP buttons mutate survey state only; they neither power nor steer the rover, a separation the interface states explicitly. Route prioritization aggregates the stored records through

$$S = 10 \cdot Critical + 7 \cdot High + 4 \cdot Medium + 2 \cdot Low + 1.5 \cdot D_{max} \quad (8)$$

with High assigned at $S \geq 30$, Medium for $18 \leq S < 30$ and Low below 18. Completed-route summaries cached in browser local Storage are user-interface history only; every authoritative figure is re-derived from AWS records on each poll.

3.6 Control-plane isolation

The ESP8266 broadcasts its own network and serves a minimal five-button page (Table 4); each button issues an HTTP GET that maps directly onto L298N input states, giving a differential-drive chassis with left, right, spin and halt primitives. Motor lines occupy D1, D2, D5 and D6, leaving the boot-strap pins D3, D4 and D8 untouched, and the ENA/ENB jumpers remain fitted so that speed trimming is unnecessary. The rationale for this segregation is fault containment: a default in the vision process, or a lost uplink, must not translate into an unattended four-kilogram chassis rolling toward traffic. An operator thumb on a phone button is the watchdog.

Table 4: Manual control commands

Request	Chassis response
/?cmd=F	Both groups forward
/?cmd=B	Both groups reverse
/?cmd=L	Left group braked, right forward

/?cmd=R	Right group braked, left forward
/?cmd=S	All lines low

3.7 Hardware summary

Table 5 lists the modules and the role each plays. Power is split: a 5 V/5 A USB-C supply feeds the Pi, a regulated 5 V rail feeds the ESP8266 and the sonar VCC, and a 12 V pack feeds the bridge, with a single common ground tying the rails together.

Table 5: Hardware modules and roles

Module	Specification	Role
Raspberry Pi 5	Quad-core Cortex-A76, USB, GPIO, Wi-Fi	Acquisition, inference, fusion, uplink
USB camera	1080p class, OpenCV capture	Road-surface frames
HC-SR04	40 kHz sonar, level-shifted echo	Downward depth channel
ESP8266 NodeMCU	Soft-AP + web server, 3.3 V logic	Manual control plane
L298N	Dual full-bridge driver	Motor interface
DC gear motors	×4, differential pairs	Traversal

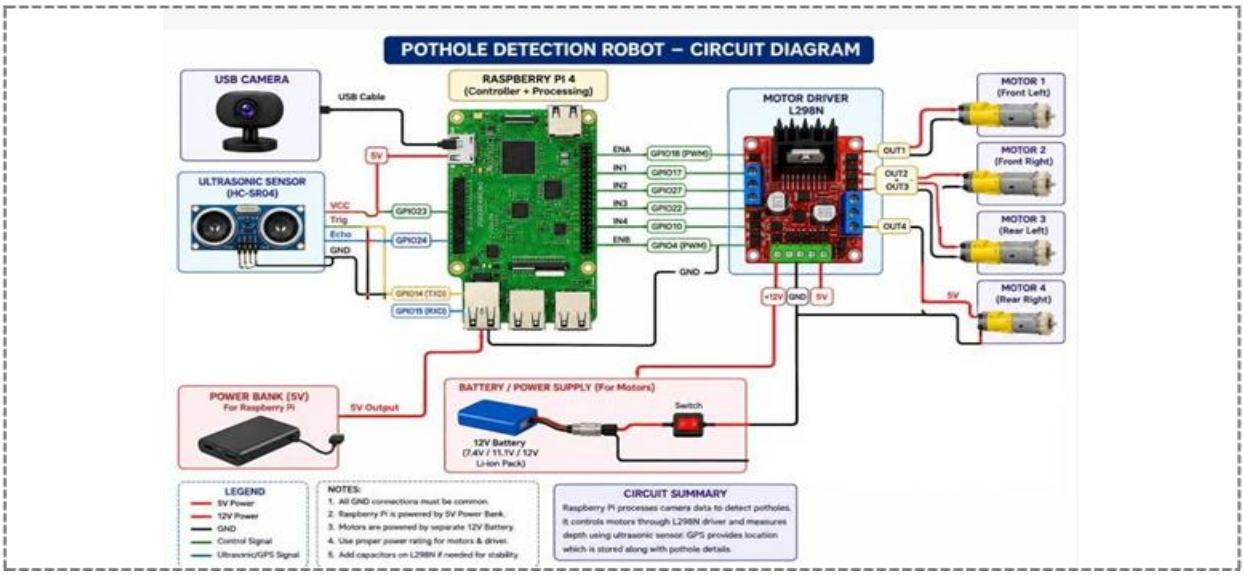


Figure 2: Circuit diagram of the rover

IV. RESULTS AND DISCUSSION

4.1 Trial procedure

The prototype was exercised module by module and then as an integrated chain: sonar against known step heights, detector against held-out road frames, backend against scripted payloads, and the dashboard against live surveys. Where three views of the same record exist — Pi console, API response, dashboard cell — they were compared to confirm that no tier silently re-interpreted the data.

4.2 Detection behavior

Under clear daylight on dry pavement the detector held well-defined craters comfortably above the 0.60 gate, and the bounding boxes tracked the visible rim closely enough for the dimension estimate to remain stable across successive frames.

Partial shadow degraded margin confidence before it degraded margin position: the same crater that scored strongly at noon dipped toward the gate in late afternoon, and occasionally fell beneath it. Water-filled depressions produced the only outright misses observed, the specular surface defeating the texture cues the network relies upon. These behaviors mirror the condition-sensitivity reported across the literature [2], [3], [8], and they argue for training sets that deliberately over-sample wet and shadowed pavement.

4.3 Depth channel behaviors

Presented with machined step heights, the sonar

tracked the reference within a few millimeters once the 40 mm fixture offset was applied; before calibration it read a consistent positive bias, which is exactly the systematic error the offset exists to remove. Over real potholes the median-of-three filter proved its worth: single pings occasionally returned wild echoes from sloped sidewalls, and the median rejected them without smoothing away genuine depth changes. Depth values should nonetheless be read as supporting estimates. Irregular geometry, chassis pitch and sensor tilt all perturb the standoff, and the channel is not a substitute for profilometry.

4.4 Backend, dashboard and arithmetic

Scripted payloads posted while a survey was ACTIVE were accepted, tagged with the minted route identifier, priced and committed; identical payloads posted while IDLE were refused and left no trace in Pothole Data, confirming that the state gate behaves as a gate rather than as a warning. Preassigned URLs rendered images in the browser with the private bucket untouched, and the poll cadence settled at seven seconds during active surveys and thirty seconds at idle, as designed. Table 6 reproduce es the supplied backend calculation test, which exercises Eqs. 4–7 end to end; it is a pipeline verification, not a road measurement.

Table 6: Supplied backend calculation test

Quantity	Value
Length / width / depth	12 cm / 8 cm / 2 cm
Volume	192 cm^3 (0.000192 m^3)
Base material	≈ 0.46 kg
Wastage	≈ 0.05 kg
Total material	≈ 0.51 kg
Estimated cost	≈ ₹7.60

Route ranking behaved monotonically: injecting a single Critical record into an otherwise Low route moved it above the High boundary, exactly as Eq. 8 prescribes. Demo mode, used for classroom demonstration, never leaked a simulated record into an AWS route summary.



Figure 3: Real-time detection output with bounding box and confidence

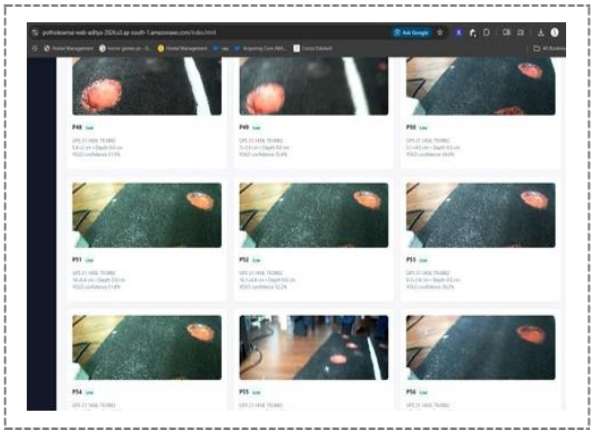


Figure 4: Dashboard live-detection view during an active survey



Figure 5: Prototype rover during field testing

4.5 Functional verification

Table 7: Functional verification of the assembled chain

Function	What was checked	Outcome
Sonar interface	Trigger/echo on GPIO 5/6, median filter, offset	Steps tracked after calibration
Confidence gate	Boxes below 0.60 discarded	No sub-gate record reached backend
Survey gating	POST /pothole while IDLE	Rejected, nothing committed

Route minting	POST /survey/start	Sequential Route-N returned
Image path	GET /pothole with image key	Presigned URL rendered, bucket private
Poll cadence	Timer inspection	7 s active / 30 s idle
Control plane	Phone on soft-AP, five buttons	Chassis responded, vision unaffected
START/STOP semantics	Button press during run	Survey state changed; rover power untouched

4.6 Discussion

The value of the chain lies in its concatenation rather than in any single stage. Confidence alone yields a box; depth alone yields a hole without context; a route tag alone yields a street without evidence. Fused on the device and priced in the cloud, they produce a record a maintenance engineer can act upon without re-measuring anything. The survey gate doubles as a data-quality interlock: stray detections outside an intentional run cannot pollute a route summary, which matters when route scores drive crew allocation through Eq. 8.

Against the literature, the comparison is deliberately asymmetric. Zeng and Zhong [8] and Chen et al. [9] report stronger detection metrics but stop at the bounding box; DeepBus [10] and the smartphone detector [11] report remotely but bind location to GNSS and carry no depth term. Our figures are therefore not line-by-line comparable, and we claim no accuracy crown. What this work adds is the missing tail: measurement, pricing and route-wise organization on hardware cheaper than the range sensors other groups reach for.

Three strengths stand out. Depth and cost are produced on commodity parts without GNSS or cellular dependency; the control plane is fault-isolated from the vision plane; and the private-bucket-plus-preassigned-URL design keeps evidence confidential by default. The limitations are equally plain. The sonar channel is geometry-sensitive and yields supporting estimates, not survey-grade depths. Traversal is manual, so coverage inherits the operator's path and patience. Browser-to-API communication depends on correct CORS and stage configuration — a transient fetch failure on the status route was observed during development and must be re-verified on the deployed stage before any demonstration. And all trials were conducted on a single unit.

Practically, the chain suits campus roads, municipal preliminary surveys and classroom demonstration of edge-to-cloud pipelines. For certified pavement assessment it should support, not supplant, professional survey equipment.

V. CONCLUSION

Manual surveys produce photographs; maintenance teams need quantities. To bridge that gap we built a rover around a Raspberry Pi 5, one ultrasonic sensor and an ESP8266, in which an on-device YOLO model detects potholes, a fused depth channel rates their severity, and a serverless AWS backend converts each accepted detection into volume, material, wastage, cost and a route-tagged record served through preassigned URLs to a static dashboard. Survey gating ensures that only intentional inspection runs enter the record, and route scoring ranks streets for repair without any satellite receiver in the bill of materials. Bench and field trials confirmed the detection-to-dashboard chain end to end, with the sonar tracking calibrated steps and the state machine refusing out-of-survey payloads. The principal contribution is therefore a complete offline-to-cloud pipeline that senses, classifies, prices and prioritizes on inexpensive hardware. Future work will add stereo or LiDAR depth for volumetric fidelity, GNSS-tagged map views for cross-system integration, autonomous traversal to relax the operator dependency, and multi-rover aggregation against the same backend.

REFERENCES

[1] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama and H. Omata, "Road damage detection and classification using deep neural networks with smartphone images," Comput.-Aided Civ. Infrastruct. Eng., vol. 33, no. 12, pp. 1127–1141, 2018, doi: 10.1111/mice.12387.

[2] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal and Y. Sekimoto, "RDD2020: An annotated image dataset for automatic road damage detection using deep learning," Data Brief, vol. 36, art. 107133, 2021, doi: 10.1016/j.dib.2021.107133.

[3] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, Y. Sekimoto, H. Omata et al., "RDD2022: A multi-national image dataset for automatic road damage detection," Geosci. Data J., vol. 11, no. 4, pp. 846–862, 2024, doi: 10.1002/gdj3.260.

[4] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc.IEEE CVPR, 2016, pp. 779–788.

[5] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in Proc. IEEE CVPR, 2017, pp. 7263–7271.

[6] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv:1804.02767, 2018.

[7] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," arXiv:2004.10934, 2020.

[8] J. Zeng and H. Zhong, "YOLOv8-PD: An improved road damage detection algorithm based on YOLOv8n model," Sci. Rep., vol. 14, art. 12052, 2024, doi: 10.1038/s41598-024-62933-z.

[9] W. Chen, J. S. Yang, C. Xia, Y. Li and X. Xiao, "Road surface damage detection based on enhanced YOLOv8," Comput. Ind., vol. 173, art. 104363, 2025, doi: 10.1016/j.compind.2025.104363.

[10] K. Bansal and S. S. Gill, "DeepBus: Machine learning based real time pothole detection system for smart transportation using IoT," Internet Technol. Lett., 2020, doi: 10.1002/itl2.156.

[11] "Deep learning approach to detect potholes in real-time using smartphone," in Proc. IEEE PuneCon, 2019, doi: 10.1109/PuneCon46936.2019.9105737.

[12] Raspberry Pi Ltd., Raspberry Pi 5 product brief, Raspberry Pi Ltd., 2023.

[13] Espressif Systems, ESP8266EX datasheet, ver. 7.1, Espressif Systems, 2025.

[14] STMicroelectronics, L298 dual full-bridge driver, datasheet.