Proceedings of the 4th International Conference on Mechatronics and Smart Systems
DOI: 10.54254/2755-2721/2026.BJ34348

Autonomous Runway Crack-Detection UGV Using Vision-Based Deep Learning and Optimal Coverage Path Planning

Zening Zhao

Saddle River Day School, NJ, USA
Zhaozening2009@gmail.com

Abstract. The degradation of airport runway surfaces poses persistent safety risks in aviation operations. Runway cracks tend to expand under environmental loads and aircraft cyclic impacts, generating Foreign Object Deposition (FOD) that directly jeopardize aircraft takeoff and landing safety. The International Civil Aviation Organization (ICAO) mandates mandatory inspections for runway surface integrity . However, current inspection methods predominantly rely on manual patrols, which suffer from low efficiency and inadequate early-stage crack detection capabilities—particularly challenging for routine inspections at resource-constrained regional and general aviation airports.This study presents the design and evaluation of an autonomous unmanned ground vehicle (UGV) system for runway crack detection, integrating real-time vision-based deep learning, geospatial localization, and optimal coverage path planning . A YOLO-based model is implemented for onboard crack detection, while a Chinese Postman-based algorithm ensures complete and efficient runway traversal . The system is developed using a Raspberry Pi and STM32 architecture within a ROS framework, and validated through simulation and controlled testing. Results focus on inspection coverage efficiency, localization consistency, detection accuracy, and the practical utility of generated inspection reports . The proposed framework demonstrates the feasibility of a low-cost, integrated solution for automated runway inspection, with potential to reduce manual labor and enable more frequent monitoring.

Keywords: runway inspection, crack detection, autonomous vehicle, YOLO, coverage path planning

1. Introduction

Runway surface degradation is a critical aviation safety issue. Cracks caused by environmental stress, aircraft loading, and material fatigue can propagate over time, potentially generating foreign object debris (FOD) that threatens aircraft engines and landing gear. This problem directly impacts both operational safety and infrastructure management, requiring airports to conduct regular inspections to maintain pavement conditions and comply with international standards. Organizations such as the International Civil Aviation Organization (ICAO) mandate that runway surfaces remain free of structural defects; however, maintaining these conditions remains challenging, particularly for regional airports with limited resources.

The current inspection method integrates multiple technical approaches. As shown in Figure 1, the manual walk inspection conducted in accordance with standards such as ASTM D5340 requires inspectors to conduct on-site patrols of the runway and assess crack conditions.Some airports supplement this with slow vehicle-based inspections using cameras or visual observation, which improve efficiency but often lack precise coverage control. More advanced solutions include drone-based inspection platforms, such as commercial UAV systems (e.g., Wingtra), which enable rapid aerial imaging and large-area mapping. While these approaches provide operational flexibility, they introduce trade-offs related to resolution consistency, regulatory constraints, and dependence on post-processing rather than real-time analysis.

DOI: 10.54254/2755-2721/2026.BJ34348

Figure 1. Schematic diagram of artificial runway crack detection
https://oxmaint.com/industries/aviation-management/manage-post-2k26/uploads/runway-taxiway-maintenance-faa-part-139-compliance-cmms.png

Despite these advancements, existing methods remain fragmented. Detection, coverage planning, and spatial reporting are typically treated as separate processes, resulting in gaps in inspection coverage, inconsistent localization of defects, and limited real-time usability. These challenges are further amplified by complex runway geometries, such as intersecting layouts, which require systematic and coordinated inspection strategies.

This research aims to design and validate an autonomous unmanned ground vehicle (UGV) system that addresses these limitations by integrating high-accuracy, ICAO-compliant crack detection with complete pavement coverage and real-time spatial reporting. The proposed system combines deep learning-based detection with optimized path planning and localization, enabling a unified, low-cost solution with direct applicability to real-world airport inspection workflows.

This work is guided by three questions: (1) how to achieve accurate crack detection under ICAO criteria, (2) how to guarantee full-surface coverage efficiently, and (3) how to integrate detection and navigation into a unified system.

2. Literature review

Runway inspection practices are governed by international aviation safety standards, most notably those established by the International Civil Aviation Organization (ICAO), which require runway surfaces to be maintained free of structural defects such as cracks and loose debris [1]. Cracks are typically classified by severity, with larger defects requiring immediate repair and smaller ones requiring monitoring and documentation. These standards emphasize not only detection accuracy but also inspection frequency, coverage completeness, and reliable reporting, all of which are difficult to achieve in practice, especially for smaller or resource-limited airports.

Existing crack inspection methodologies can be broadly categorized into manual, vehicle-based, aerial, and sensor-based approaches. Manual walking surveys remain the most widely used method due to their simplicity, but they are labor-intensive, time-consuming, and subject to human variability, leading to inconsistent detection results. Vehicle-based inspection systems improve efficiency but often rely on expensive sensing equipment and lack precise coverage control.As shown in Figures 2 and 3, drone-based detection technology offers flexible aerial imaging capabilities, but is subject to regulatory constraints and may lack sufficient resolution to detect fine surface cracks. While static sensor networks enable continuous monitoring, they are costly and have limited spatial coverage. Consequently, existing methods face trade-offs among cost, accuracy, coverage range, and operational complexity, and none fully meet the requirements for high-frequency, high-resolution, and comprehensive runway detection.

DOI: 10.54254/2755-2721/2026.BJ34348

Figure 2. Detecting runway cracks using drones
https://www.fanshawec.ca/sites/default/files/styles/11_6_banner_image_large_1100x600/public/images/2023-11/2022-09-21_Remotely%20Piloted%20Aerial%20Systems_Runway%20Inspection_AL-14.jpg?h=4362216e&itok=hGkFrgIE

Figure 3. The drone is filming the crack
https://www.fanshawec.ca/sites/default/files/styles/medium_three_image_secondary_371_x_371_/public/images/2023-11/2022-09-21_Remotely%20Piloted%20Aerial%20Systems_Runway%20Inspection_AL-39.jpg?itok=_yFynH4R

The rapid advancement of deep learning technologies, particularly the application of Convolutional Neural Networks (CNNs), has significantly improved the accuracy of automated crack detection [2]. Figure 4 illustrates the differences in computational speeds across generations of the YOLO algorithm.The YOLO (You Only Look Once) series models have become the mainstream solution in real-time object detection due to their excellent balance between speed and precision [3]. Although next-generation large-scale YOLO variant models offer superior detection performance, they require higher computational resources, making them difficult to deploy in embedded systems. The lightweight YOLOv8n (Nano) model provides an optimal trade-off, maintaining sufficient detection accuracy while being compact enough for real-time deployment on resource-constrained edge platforms such as Raspberry Pi [4].

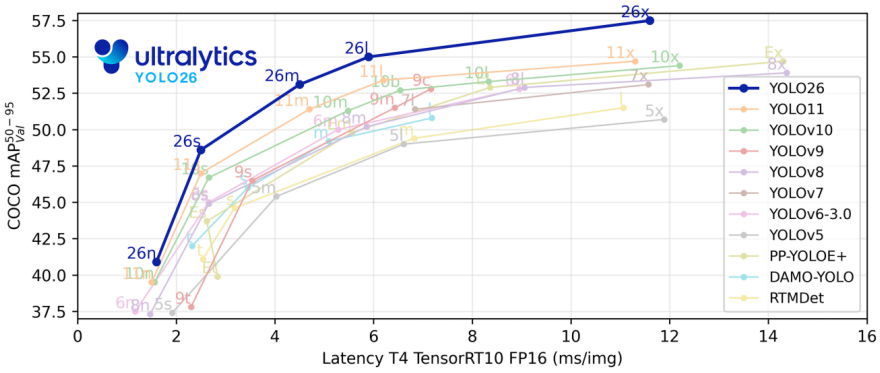

Figure 4. Yolo model speed-accuracy comparison chart

DOI: 10.54254/2755-2721/2026.BJ34348

https://raw.githubusercontent.com/ultralytics/assets/main/yolov8/yolo-comparison-plots.png

In addition to crack detection, efficient full-coverage inspection of runway surfaces is another core requirement of automated systems. This problem can be modeled as a graph traversal task, which requires traversing all sections of the target area [5]. The China Postman Problem (CPP) provides a mature solution for this task, with its core principle being the conversion of the original graph into an Euler graph to solve the shortest path that traverses all edges at least once. Compared to traditional reciprocating scanning methods, path planning methods based on CPP can minimize redundant trips while ensuring full coverage, particularly suitable for complex airport geometries such as intersecting runways [6].

Despite these advances, existing research largely treats detection, navigation, and localization as separate components. Crack detection models are often evaluated offline without integration into autonomous systems, while coverage planning methods are not typically combined with real-time perception. This lack of integration results in incomplete coverage, inconsistent localization, and limited real-time usability. Therefore, there remains a need for a unified, low-cost system that integrates accurate crack detection, efficient coverage planning, and real-time spatial reporting into a single operational framework.

3. Methodology

3.1. System architecture

The proposed autonomous inspection platform is designed as an integrated system capable of real-time runway crack detection through a synergy of autonomous navigation, vision-based perception, and geospatial mapping. The architecture is bifurcated into a high-level compute layer for complex processing and a low-level control layer for precise mechanical actuation.

The physical architecture utilizes a dual-tier processing strategy to ensure both computational depth and real-time responsiveness. A Raspberry Pi serves as the primary onboard computer(See Figure 5 for the actual item), managing high-level tasks including path planning, computer vision inference, and data logging. To offload time-critical motor control and sensor polling, a secondary STM32 microcontroller is employed, ensuring stable execution of low-level hardware abstractions [7].

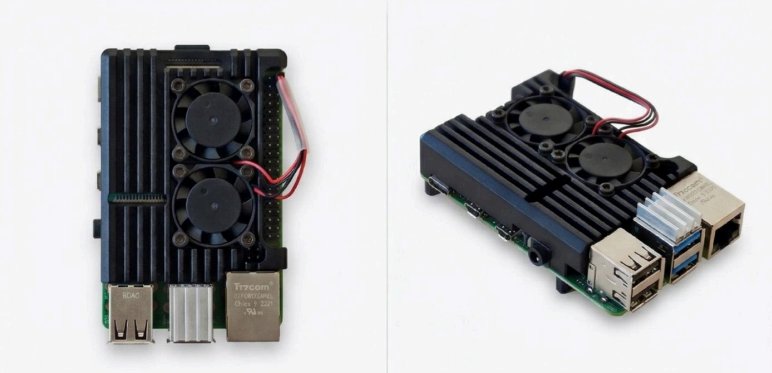

Figure 5. Raspberry Pi 4

The perception suite is centered around an OAK-D Pro depth camera(See Figure 6 for the actual item), which provides high-resolution RGB imagery coupled with spatial depth data. For localization, the system integrates a Global Positioning System (GPS) module (See Figure 7 for the actual item) and an Inertial Measurement Unit (IMU) [8]. This multi-sensor fusion approach allows the UGV to maintain high-fidelity spatial awareness, enabling the precise georeferencing of detected structural anomalies during runway traversal.

DOI: 10.54254/2755-2721/2026.BJ34348

Figure 6. OAK-D depth camera

Figure 7. GPS

The logic layer is built on the Ubuntu Linux distribution and employs the Robot Operating System (ROS) as its foundational middleware. As shown in Figure 8, ROS implements modular, node-based communication, where the perception, planning, and control subsystems interact asynchronously via a publish-subscribe model [9].

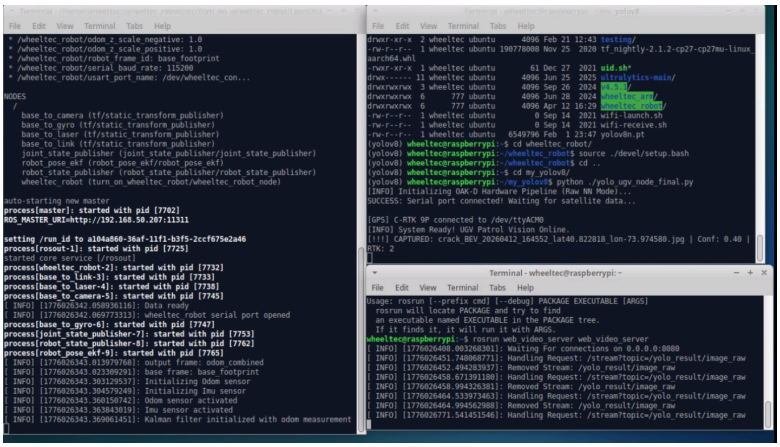

Figure 8. ROS environment

The crack detection pipeline employs the YOLOv8n (Nano) architecture, specifically chosen for its optimized performance on edge computing hardware. To maximize execution efficiency and minimize latency, the core system drivers and fusion algorithms are implemented in C++ [10]. This "master driver" synchronizes the YOLO inference stream with incoming GPS telemetry, allowing for the real-time logging of geolocated crack data. Communication between the Raspberry Pi and the STM32 is maintained through a serial interface, where the high-level computer publishes velocity commands and the microcontroller returns encoder feedback to update the vehicle's state estimation.

DOI: 10.54254/2755-2721/2026.BJ34348

Prior to actual deployment, the system's navigation and coverage algorithms were validated in a high-fidelity simulation environment. As shown in Figure 9, this validation process utilized a gazebo-based physical modeling tool and the RViz real-time data visualization tool.

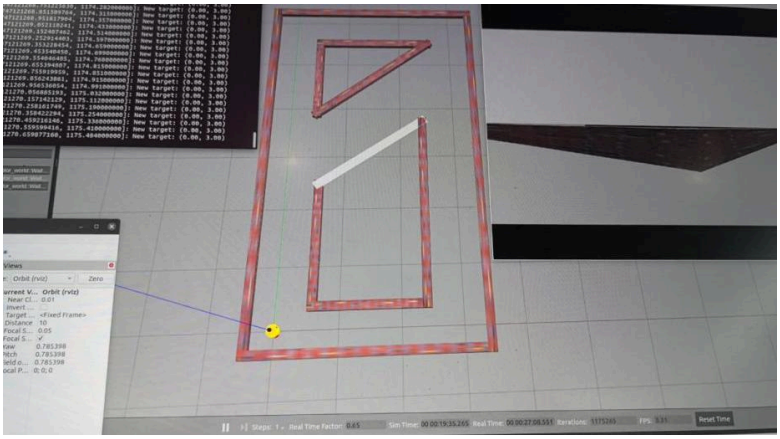

Figure 9. Gazebo simulation shooting

In the Gazebo environment, the UGV was modeled with realistic kinematic properties, including wheel configurations and motion constraints, to simulate traversal on a representative runway surface. RViz was utilized to monitor the vehicle's pose, planned trajectory, and coverage progress, providing a diagnostic interface for debugging path-planning logic. This simulation-in-the-loop approach allowed for the iterative refinement of autonomous behavior, ensuring that issues such as path deviation or incomplete coverage were addressed before transitioning to real-world operational testing [11].

3.2. Coverage path planning using the chinese postman algorithm

The navigational intelligence of the UGV is governed by a hierarchical three-layer Coverage Path Planning (CPP) framework [12]. This architecture transforms raw geospatial data into a continuous, optimized traversal path that ensures 100% surface coverage of the runway environment while minimizing redundant travel (deadheading) [13] .

3.2.1. Graph synthesis from geospatial data

The foundation of the planning engine is the construction of a weighted undirected graph, $G = (V, E)$, derived from high-resolution geospatial primitives. As shown in Figure 10, the process begins by extracting runway centerlines from Google Earth KML data. These centerlines are then programmatically processed to identify all intersection points, where runways are split to create distinct edges.

Figure 10. Abstracting airport runway into graph structure in realistic context

In this topological model, the vertex set V comprises all runway endpoints and intersection nodes, while the edge set E represents the segments connecting them. Each edge is assigned a weight $w(e)$ corresponding to the segment

DOI: 10.54254/2755-2721/2026.BJ34348

length in meters. This formal graph definition allows the system to treat the complex airport layout as a mathematical network optimized for global traversal [13].

3.2.2. Macro-level optimization: the chinese postman heuristic

To ensure that every runway segment is inspected with maximum efficiency, the system solves the Chinese Postman Problem (CPP). The objective is to determine the minimum-weight closed walk that traverses every edge in G at least once [12].

From a graph theory perspective, a graph contains an Eulerian circuit—a route that visits every edge exactly once—only if every node has an even degree. Because real-world airport layouts often contain "dead-end" segments or asymmetric intersections resulting in odd-degree vertices, the graph G must be augmented. The algorithm executes the following steps:

1. Identification of Odd-Degree Vertices: Locating all nodes where the number of incident edges is odd.
2. Minimum-Weight Perfect Matching (MWPM): Calculating the shortest paths between these odd-degree vertices to find the optimal pairs for augmentation.
3. Graph Augmentation: Adding matching edges to the original graph to produce an Eulerian graph G'.
4. Circuit Generation: Finding the Eulerian circuit on G'.

The resulting output is an ordered sequence of edges that provides a mathematically proven optimal traversal with minimum deadheading distance [14].

3.2.3. Edge-to-rectangle expansion and micro-level coverage

Once the optimal edge traversal sequence is established, the algorithm transitions from linear graph edges to area-based coverage. This is achieved through a two-stage decomposition process:

• Edge Expansion (ASTM D5340 Compliance): Each linear segment of the graph is expanded into a series of rectangular inspection units. These units are dimensioned according to ASTM D5340 standards for Pavement Condition Index (PCI) surveys. At runway intersections, a deduplication logic is applied to ensure that overlapping areas are not scanned multiple times, preserving the integrity of the cumulative coverage data.

• Boustrophedon Scanning: Within each rectangular unit, the system generates a "micro" path using a Boustrophedon (ox-turning) pattern. The width of each longitudinal sweep is dynamically calculated based on the effective Field of View (FOV) of the OAK-D Pro camera and the required overlap for image stitching. By alternating the UGV's direction in back-and-forth passes, the algorithm ensures that the entire width of the runway is captured at the required resolution, effectively bridging the gap between global path optimization and local sensor-based inspection.

3.3. Perception module: vision-based crack detection

The system's ability to identify structural anomalies in real-time is driven by a deep-learning-based perception module. This section details the selection of the object detection architecture, the dataset composition, and the training methodology employed to achieve high-precision crack identification on edge hardware.

3.3.1. Model selection and architecture

To facilitate real-time processing on the Raspberry Pi's constrained computational resources, the YOLOv8n (Nano) architecture was selected. YOLOv8 (You Only Look Once) is a state-of-the-art single-stage detector that refines spatial features through a specialized backbone and a decoupled detection head. The "Nano" variant was specifically utilized to minimize inference latency while maintaining a high mean Average Precision (mAP). By prioritizing a lightweight model, the UGV can process video streams from the OAK-D Pro camera at a consistent frame rate, ensuring that no runway segments are bypassed during high-speed traversal [15].

3.3.2. Dataset acquisition and pre-processing

The model was trained on a robust dataset primarily sourced from Kaggle's pavement and runway crack repositories, supplemented by localized imagery to enhance domain specificity. The final training set comprised approximately 3,000 high-resolution images, encompassing diverse crack morphologies such as longitudinal, transverse, and alligator cracking.

DOI: 10.54254/2755-2721/2026.BJ34348

To ensure the model's generalizability across varying lighting conditions and runway textures, the following pre-processing steps were applied:

• Resolution Standardization: All images were resized to a uniform 640x640 pixel input dimension to align with the YOLOv8 architectural constraints.

• Data Augmentation: Techniques including random rotation, mosaic augmentation, and brightness adjustments were utilized to simulate different time-of-day scenarios and sensor noise.

Data Split: The dataset was partitioned into an 80/10/10 split for training, validation, and testing, respectively, to prevent over-fitting and ensure objective performance evaluation.

3.3.3. Training methodology and hyperparameters

Training was conducted using transfer learning, utilizing pre-trained COCO weights as a starting point. This approach leveraged learned low-level features (edges and textures), significantly reducing the required training time and improving convergence stability. The model was fine-tuned for 100 epochs using a Stochastic Gradient Descent (SGD) optimizer with a fluctuating learning rate to navigate the loss landscape effectively. The training was offloaded to a high-performance GPU environment to accelerate the backpropagation process, resulting in a finalized weight file optimized for FP16 precision to further enhance edge inference speeds during deployment.

3.3.4. Performance evaluation and results

The performance of the trained YOLOv8n model was quantitatively evaluated using a standardized validation set. The model reached convergence within 100 epochs, exhibiting stable loss reduction across box, classification, and distribution focal loss (DFL) metrics. As summarized in Table 1, the model achieved a mean Average Precision (mAP50) of 0.816, with a precision of 0.903 and a recall of 0.802. In terms of computational efficiency, the architecture demonstrated an average inference latency of 2.19 ms per frame during the validation phase (utilizing a Tesla P100 GPU for benchmarking). This high throughput ensures that the detection pipeline remains a non-bottleneck component, allowing the overall system to maintain real-time performance even when transitioned to the localized Raspberry Pi compute module.

Table 1. Model performance evaluation table

Metric	Value
mAP50	0.816
mAP50-95	0.589
Precision	0.903
Recall	0.80
F1-Score	0.85 (at 0.47 confidence)
Training Instances	3461
Inference Time (Edge)	2.19 ms

3.4. System integration & validation

This study employed a progressive multi-stage experimental protocol to validate the autonomous inspection platform, transitioning from high-fidelity computer simulations to physical field deployment. This approach ensured that both the path planning algorithm and sensor fusion logic maintained robust performance across varying operational complexity scenarios, ultimately achieving successful hardware system integration and implementation.

The initial system validation was conducted in a Gazebo simulation environment, with the primary objective of benchmarking path planning algorithm performance. Algorithm testing utilized multiple typical runway configurations abstracted from real airport maps, including single straight runway segments for baseline testing and cross-shaped runways designed to evaluate intersection de-duplication and turning capabilities. During this phase, rigorous evaluations were performed on the system's ability to construct undirected graphs and generate optimal traversal paths. Key evaluation metrics included coverage completeness, path efficiency (comparative analysis between optimal and actual path lengths), and reduction in redundant intersection overlaps. Simulation results demonstrated that the proposed algorithm consistently generated effective inspection routes free from blind spots, regardless of the complexity of input runway layout structures.

DOI: 10.54254/2755-2721/2026.BJ34348

After passing simulation validation, the system completed real-world integration testing of hardware and software stack through field trials. During physical deployment, ROS middleware enabled full-process synchronization between the Raspberry Pi upper-layer controller and STM32 motor driver. The field testing required the UGV to simultaneously run the YOLOv8 detection model and record crack data with geographic coordinates while executing the pre-calculated optimal inspection path. Figure 11 shows the crack detection results identified by the YOLOv8 model during the UGV's operation. This phase focused on evaluating real-time trajectory tracking accuracy, inspection execution duration, and the stability of the communication interface between the perception and execution layers. By comparing the field-collected trajectory data with simulation benchmarks, the system's operational reliability was verified, demonstrating its capability to autonomously perform runway inspection and maintenance tasks.

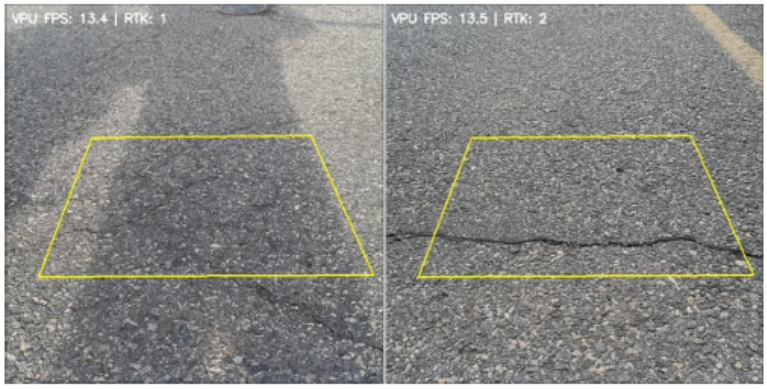

Figure 11. YOLOv8real detection results

4. Conclusion

First, the lightweight crack detection model YOLOv8n achieves an mAP50 score of 0.816 and detection accuracy of 0.903 on the test set. It supports real-time inference on Raspberry Pi edge devices, balancing detection precision with deployment feasibility. This model effectively meets the early-stage identification requirements for runway micro-cracks and complies with ICAO regulatory standards.

Secondly, the three-tier full-coverage path planning framework designed based on the CPP algorithm can adapt to airport runway configurations of any complexity. Simulation experiments demonstrated 100% blind-spot-free coverage across the entire runway area.While ensuring complete inspection coverage, it significantly enhances overall inspection efficiency and effectively addresses repeated inspection issues in complex configurations such as intersecting runways.

Thirdly, the proposed "Raspberry Pi+STM32" dual-control architecture with ROS modular integration achieves integrated capabilities including path planning, autonomous navigation, real-time detection, and spatial positioning. Both Gazebo simulations and outdoor physical experiments have validated the system's operational stability and engineering feasibility, enabling full automation of inspection workflows.

Fourth, in addition to crack detection functionality, the system can simultaneously record the spatial coordinates of each defect. By leveraging GPS data to match detection results with absolute geographic coordinates, it enables consistent tracking of crack locations across multiple inspections. The detection results and positioning data can automatically generate structured inspection reports containing crack locations, severity classification levels, and visual reference images. These reports provide actionable decision-making support for airport pavement operation and maintenance planning, and can be directly integrated with airport pavement PCI ratings and standardized maintenance procedures.

References

[1] International Civil Aviation Organization (ICAO). (2018). Annex 14 to the Convention on International Civil Aviation: Aerodromes — Volume I: Aerodrome Design and Operations (8th ed.). ICAO.

[2] Amhaz, R., Chambon, S., Idier, J., & Baltazart, V. (2016). Automatic crack detection on pavement surfaces using structured prediction. IEEE Transactions on Intelligent Transportation Systems, 17(10), 2718–2729.

[3] Li, Q., Yao, M., Yao, X., & Zhang, D. (2014). Pav ement crack detection using YOLO-based deep convolutional networks. Journal of Computing in Civil Engineering, 28(3), 04014017.

[4] Zhang, K., Zhang, L., Yang, M., & Zhang, Y . (2017). Road crack detection using deep convolutional neural networks. Proceedings of the International Joint Conference on Neural Networks (IJCNN), 1-8.

DOI: 10.54254/2755-2721/2026.BJ34348

[5] Bar-Shalom, Y., Li, X. R., & Kirubarajan, T. (2001). Estimation with applications to tracking and navigation. Wiley.
[6] Skog, I., & Hndel, P. (2009). In-car positioning and navigation technologies—A survey. IEEE Transactions on Intelligent Transportation Systems, 10(1), 4-21.
[7] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7(66), eabm6074.
[8] Groves, P. D. (2013). Principles of GNSS, inertial, and multisensor integrated navigation systems (2nd ed.). Artech House.
[9] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., & Ng, A. Y. (2009). ROS: An open-source robot operating system. ICRA Workshop on Open SourceSoftware, 3(2).
[10] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.
[11] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to-6-autonomous mobile robots (2nd ed.). MIT Press.
[12] Edmonds, J., & Johnson, E. L. (1973). Matching, Euler tours, and the Chinese postman problem. Mathematical Programming, 5(1), 88–124.
[13] Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. Robotics and Autonomous Systems, 61(12), 1258–1276.
[14] Eiselt, H. A., Gendreau, M., & Laporte, G. (1995). Arc routing problems, part I: The Chinese postman problem. Operations Research, 43(2), 231-242. https: //doi.org/10.1287/opre.43.2.231
[15] R. Varghese and S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness, " 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS), Chennai, India, 2024, pp. 1-6, doi: 10.1109/ADICS58448.2024.10533619.