JRAEE

Journal of Robotics, Automation, and Electronics Engineering
Vol. 4, No. 1, 2026, pp. 226-236
e-ISSN 3025-3780
https://journal.uny.ac.id/v3/jraee

A Real-Time Road Damage Monitoring System using YOLOv11 on an Edge Computer

Zaki Nugraha[a,1,*], Aris Nasuha[b,2]

[a,b]Dept. of Electrical and Electronics Engineering, Vocational Faculty, Universitas Negeri Yogyakarta
[1]zakinugraha.2021@student.uny.ac.id; [2]arisnasuha@uny.ac.id
[*]Corresponding Author

ARTICLE INFO

Article History
Received Date Month Year
Revised Date Month Year
Accepted Date Month Year

ABSTRACT

This study developed a real-time road damage detection system on an edge device to address the inefficiency and subjectivity of manual monitoring. Following the Engineering Design Process (EDP), the system uses a YOLOv11 model—trained in PyTorch and converted to ONNX for optimization—to classify four types of road damage on a Raspberry Pi 5, integrated with a GPS module for automatic coordinate recording. Field evaluation showed that after optimization, the system achieved an average processing speed of 3.66 FPS with an inference latency of 277.69 ms, while maintaining efficient resource usage (45.06% RAM, 65.05% CPU load) and an mAP@50 of 0.554. These results demonstrate the feasibility of an autonomous, accurate, real-time road damage detection system on a low-cost edge computing platform, producing geospatial data ready to support operational decision-making in road maintenance.

Keywords
YOLOv11;
GPS;
Computer Vision;
ONNX;
Edge Computing.

ABSTRAK

Penelitian ini mengembangkan sistem deteksi kerusakan jalan secara real-time pada perangkat edge untuk mengatasi inefisiensi dan subjektivitas pemantauan manual. Mengikuti Engineering Design Process (EDP), sistem menggunakan model YOLOv11—dilatih dengan PyTorch dan dikonversi ke ONNX untuk optimasi—guna mengklasifikasikan empat jenis kerusakan jalan pada Raspberry Pi 5, yang diintegrasikan dengan modul GPS untuk pencatatan koordinat otomatis. Evaluasi lapangan menunjukkan bahwa setelah optimasi, sistem mencapai kecepatan pemrosesan rata-rata 3,66 FPS dengan latensi inferensi 277,69 md, sambil mempertahankan penggunaan sumber daya yang efisien (RAM 45,06%, beban CPU 65,05%) dan mAP@50 sebesar 0,554. Hasil ini membuktikan kelayakan sistem deteksi kerusakan jalan yang otonom, akurat, dan real-time pada platform edge computing berbiaya rendah, yang menghasilkan data geospasial siap pakai untuk mendukung pengambilan keputusan operasional dalam pemeliharaan jalan.

1. Introduction

Road infrastructure is a vital asset supporting mobility and the economy, yet its condition is often degraded

by damage such as cracks and potholes, which impact safety and comfort. Manual road condition monitoring has fundamental weaknesses: it is slow, expensive, and highly subjective. Although modern systems like the City Road Management System (CRMS) have been introduced in Indonesia, the core process of identifying specific damage types still heavily relies on manual visual interpretation by operators [1]. This dependence on the "human eye" perpetuates issues of scalability and objectivity, necessitating a faster, automated, and more measurable monitoring solution.

As a solution, survey automation using computer vision and deep learning has grown rapidly. Various studies have explored deep learning architecture for this task. For example, Hsieh et al. (2024) integrated YOLOv7 with IMU data on an NVIDIA Jetson Nano, but this system relies on 5G connectivity and cloud processing [2]. Mutijarsa et al. (2023) designed an IoT-based system, but the damage detection process was performed as post-processing, not real-time in the field [3]. Other studies focus on model comparison, such as Sidqi et al. (2024), who tested various YOLO variants and found YOLOv8s provided the best precision for four damage classes [4], while Wu et al. (2023) designed a lightweight model (YOLO-LWNet) for mobile devices [5]. Furthermore, Zanevych et al. (2024) achieved a high mAP with YOLOv11, but their focus was limited to only detecting potholes [6]. This state-of-the-art review reveals a gap: a lack of fully autonomous, low-cost systems capable of real-time inference directly on an edge device without cloud dependency, while simultaneously integrating GPS data.

To address this gap, this study proposes the design and implementation of an autonomous road damage detection system using the YOLOv11 model [7], optimized for a low-cost edge computing device, the Raspberry Pi 5. The contribution of this research is the development and validation of a platform capable of real-time detection of four types of damage (longitudinal cracks/D00, transverse cracks/D10, alligator cracks/D20, and potholes/D40) while simultaneously and automatically recording geospatial coordinates (GPS) [8], operating fully on the edge device without requiring cloud connectivity.

2. Method

2.1 Research Methodology

This research adopted the Engineering Design Process (EDP) as its core methodology, encompassing a structured cycle of Ask, Research, Imagine, Plan, Create, Test, and Improve. This systematic approach guided the development from the initial requirements definition and literature review to the final validated prototype, allowing for iterative optimization based on empirical test results. The cycle of this process, which guided the project from planning to improvement, is illustrated in Fig. 1.

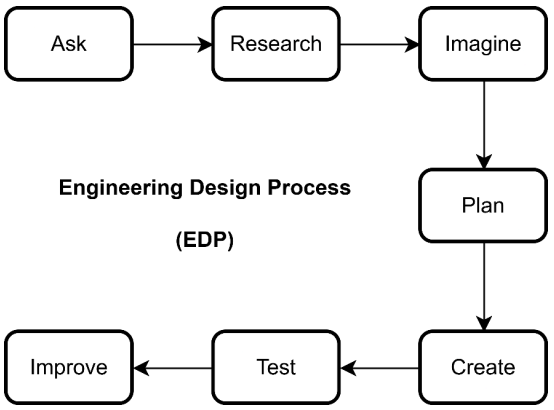


Fig.1 Engineering Design Process (EDP)

2.2 System Architecture and Hardware Design

The system's high-level architecture is designed around three core modules: Input, Processing, and Output, as illustrated in the block diagram in Fig. 2. The input module consists of a GoPro Hero 10 camera for video stream acquisition and a u-blox NEO-6MV2 GPS module for location coordinate acquisition. The Processing module is the system's core, utilizing a Raspberry Pi 5 (8GB) [10] to run the YOLOv11 model, perform object tracking, and synchronize data. The Output module provides real-time feedback to a portable monitor via the GUI and saves all resulting data (video, logs) to a MicroSD card.

ISSN 3025-3780 (online)
ISSN 3025-4590 (printed)

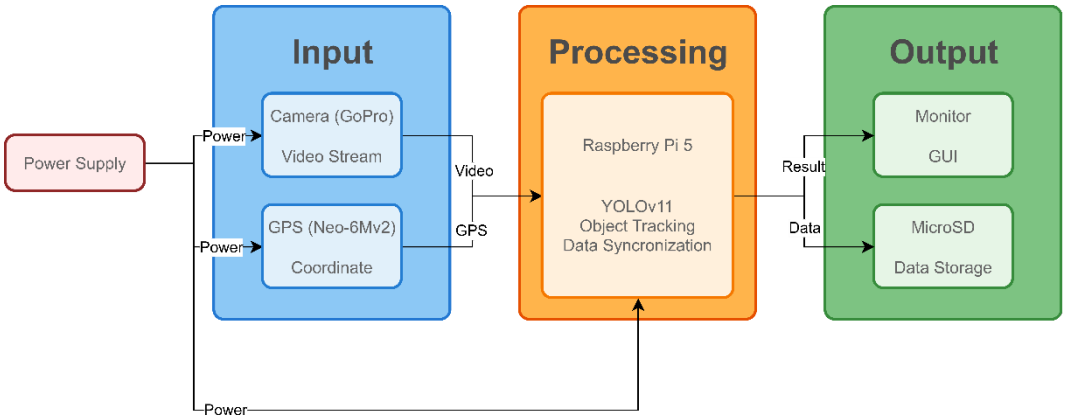


Fig.2 Block diagram system for architecture design

The detailed electronic hardware design, shown in Fig. 3, details the physical interconnection of these components. The entire system is powered from a vehicle's 12V source via a Taffware Car Power Inverter, which provides a 220V AC output. This powers a 5.1V/5A USB-C Power Supply for the Raspberry Pi 5. The Raspberry Pi 5 connects to the GoPro camera (UVC) and the u-blox GPS module via its USB ports for data transfer. Video output for the GUI is sent to the portable monitor via a micro-HDMI-to-mini-HDMI connection, while an active fan connected to the Pi's JST 4-pin header ensures thermal management.

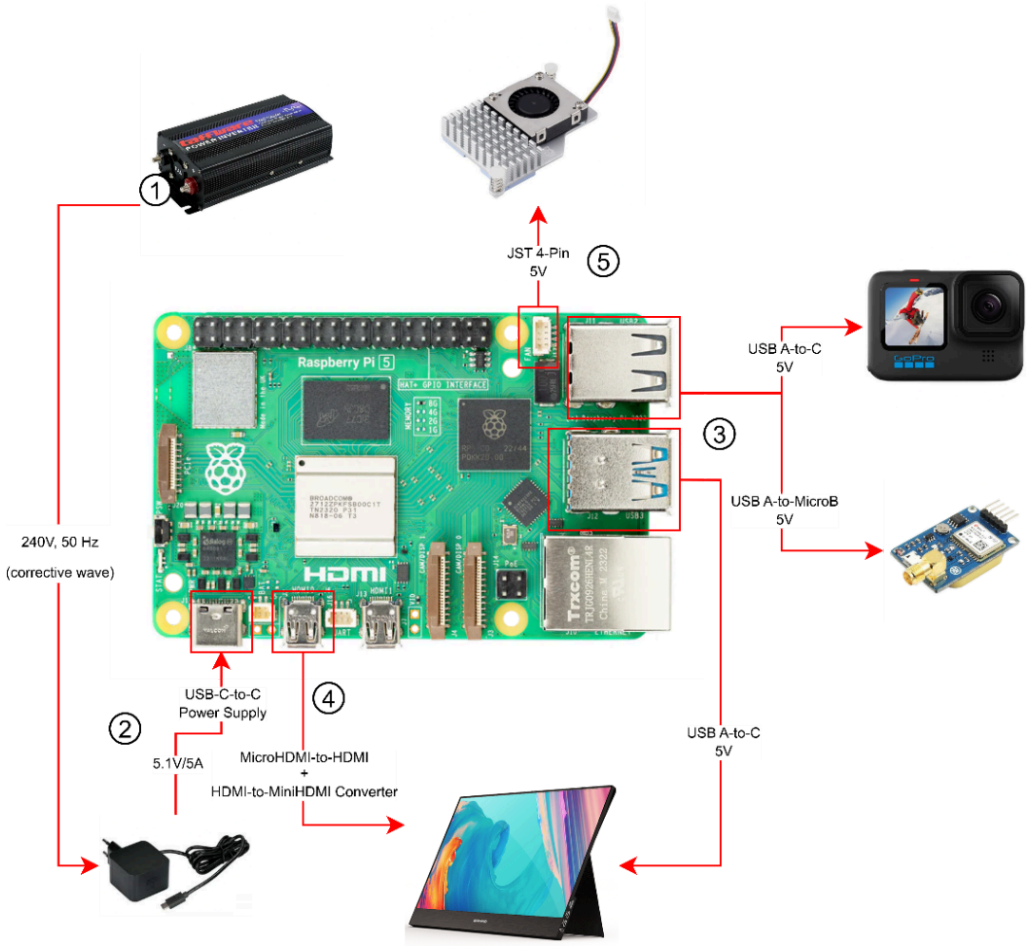


Fig.3 Hardware design

2.3 Software Design and Data Flow

The system's software, developed in Python 3.12, operates on a multi-threaded architecture to ensure real-time data synchronization. The main process initializes a custom-built GUI using `customtkinter`. Upon

starting, two parallel threads are launched:

- Inference Thread: This thread captures frames from the video stream, processes them using the YOLOv11 model (via the Ultralytics library and OpenCV), and annotates the video.
- GPS Thread: This thread continuously reads and parses NMEA sentences from the GPS module using `pynmea2` and `serial` libraries.

The main loop synchronizes the outputs from these threads, annotating video frames with both detection results (bounding boxes, class labels) and the most recent GPS coordinates (latitude, longitude, timestamp). All synchronized data is saved in structured formats (MP4 for video, JavaScript Object Notation (JSON) for detection logs, and Keyhole Markup Language (KML) / GPS Exchange Format (GPX) for geospatial tracks) for post-survey analysis. The complete software flowchart, detailing the multi-threaded logic and data flow, is shown in Fig. 4.

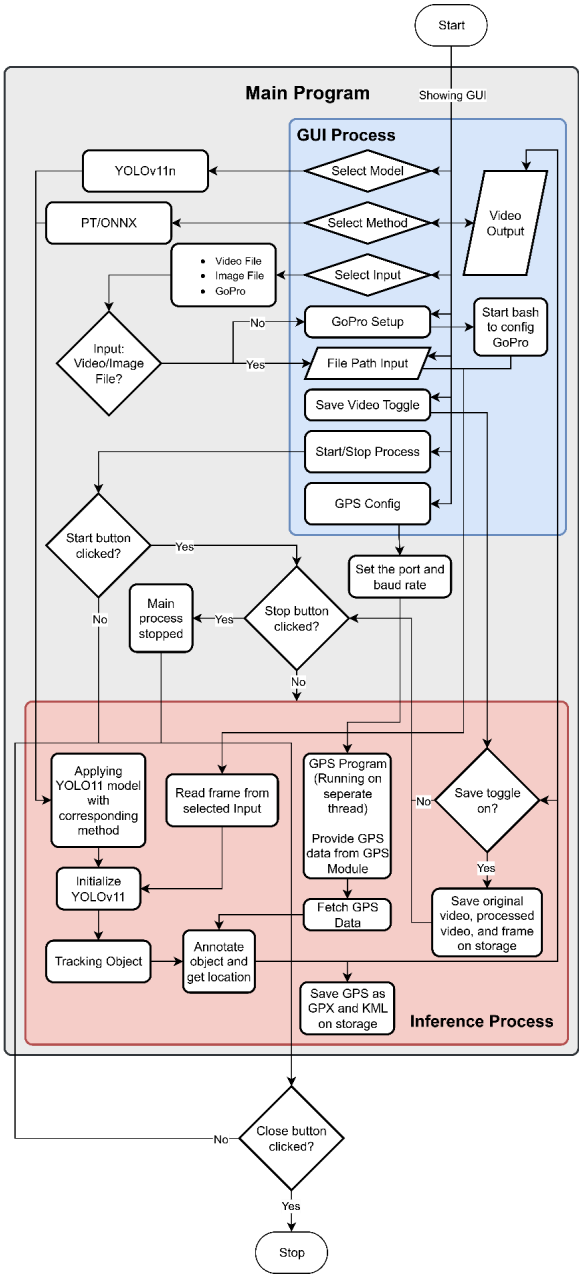


Fig.4 Software flowchart

2.4 Dataset and Model Training

ISSN 3025-3780 (online)
ISSN 3025-4590 (printed)

The detection model was trained on a combined dataset sourced from the public CRDDC 2022 [9] and the 2023 Bina Marga Road Day Competition, ensuring diverse data from various countries, including Indonesia. The dataset was manually cleaned, standardized, and annotated to cover four distinct damage classes: longitudinal cracks (D00), transverse cracks (D10), alligator cracks (D20), and potholes (D40). The final dataset was split into training (70%), validation (20%), and testing (10%) sets. The YOLOv11-nano (YOLOv11n) model was trained using PyTorch on an NVIDIA RTX 3060, with a batch size of 16, targeting 1000 epochs. An early stopping mechanism with a patience of 100 epochs was implemented to prevent overfitting. The distribution of the combined dataset is shown in **Fig. 6**.

2.5 Model Optimization for Edge Computing

To achieve real-time performance on the resource-constrained Raspberry Pi 5, the trained PyTorch (.pt) model was converted to the ONNX (Open Neural Network Exchange) format. This conversion enables the use of the highly optimized ONNX Runtime, which performs static graph optimizations (e.g., layer fusion) and utilizes compute kernels optimized for ARM Central Processing Unit (CPU) cores. This step was critical for reducing inference latency and stabilizing the frame rate, aiming to move from an unstable baseline to a reliable real-time system.

2.6 System Validation and Performance Metrics

The fully integrated system was validated through field tests in a vehicle. Three distinct routes were selected to represent different operational environments: semi-rural, rural/village, and dense urban. During these tests, the system's performance was evaluated based on a set of key quantitative metrics:

- Accuracy: Mean Average Precision (mAP), the primary metric for object detection, which calculates the average Precision-Recall curve. It is defined as:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \quad (1)$$

  where AP_i is the Average Precision for class i and N is the total number of classes. We primarily use mAP@50 (Intersection over Union (IoU) threshold of 0.5).
- Speed: Frames Per Second (FPS) and inference latency (ms).
- Efficiency: CPU load (%), Random Access Memory (RAM) consumption (%), and CPU temperature (°C).

System metrics were logged using the `psutil` library and custom scripts, while inference speed was captured directly from the Ultralytics model's profiler.

3. Result and Discussion

The system's performance was evaluated in two key areas: model accuracy and operational system performance (speed, efficiency, and stability).

3.1 Model Accuracy and Training

The YOLOv11n model was trained as described in the methodology. The training process successfully converged, automatically halting at 289 epochs due to the early stopping mechanism, which prevented overfitting. The training and validation loss curves (**Fig. 5**) demonstrate stable convergence.

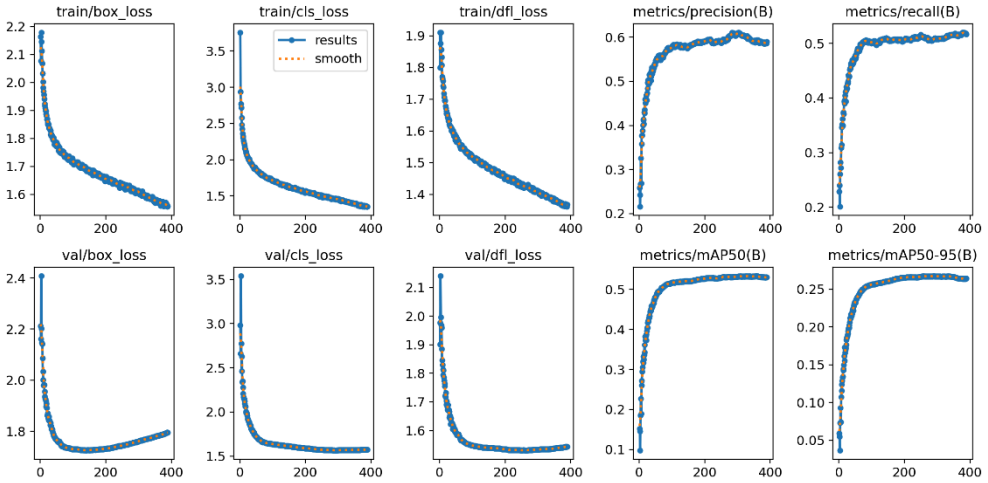


Fig.5 Training and Validation Loss Curves

The final model was evaluated on the 10% test split, achieving a mean Average Precision (mAP) at an IoU of 0.5 (mAP@50) of 0.554, and an mAP@50-95 of 0.278. The dataset distribution for this evaluation is shown in **Fig. 6**. Performance varied by class, with alligator cracks (D20) being the most accurately detected (mAP@50 = 0.679), while longitudinal cracks (D00) were the most challenging (mAP@50 = 0.493).

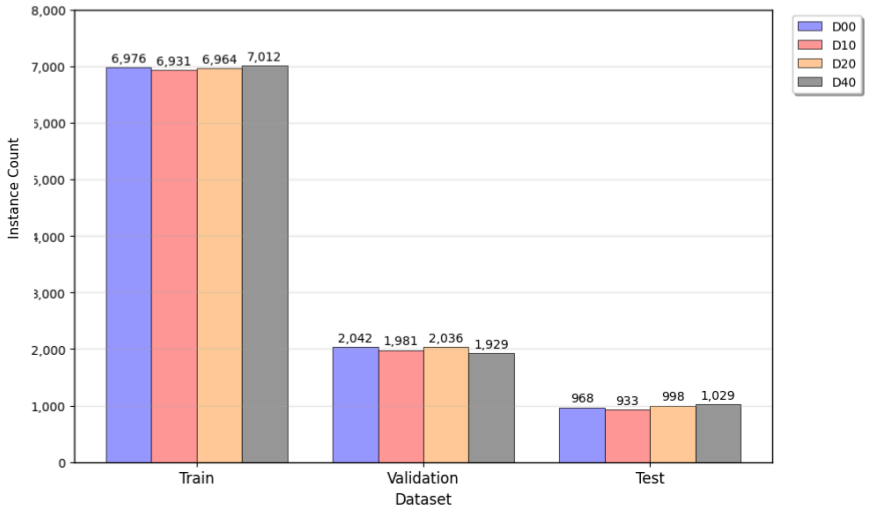


Fig.6 Instance Distribution for Each Class

A key finding from the confusion matrix (**Fig. 7**) revealed that the primary source of error was not misclassification between damage types, but a high rate of false negatives—the model often classified a real instance of damage as "background."

ISSN 3025-3780 (online)
ISSN 3025-4590 (printed)

Fig.7 Confusion Matrix

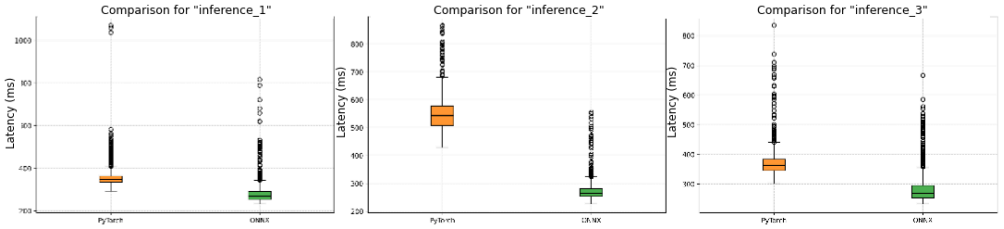
3.2 System Performance: Optimization for Real-Time Edge Computing

A critical objective of this research was to validate the feasibility of real-time processing on a low-cost edge device. To quantify this, we compared the performance of the baseline PyTorch (.pt) model against the optimized ONNX (.onnx) model on the Raspberry Pi 5.

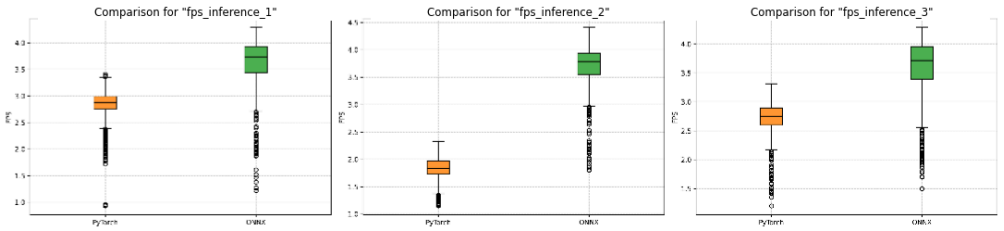
The results, summarized in **Table 1** (and detailed in **Fig. 8**), show a dramatic improvement. The baseline PyTorch model was unstable, with performance dropping as low as 1.85 FPS, failing to meet the 2 FPS target. In contrast, the ONNX model achieved a stable average processing speed of 3.66 FPS with an average latency of 277.69 ms. This optimization yielded a 47.6% increase in processing speed, successfully exceeding the system's performance target.

Table 1. PyTorch vs. ONNX Model Performance Statistics

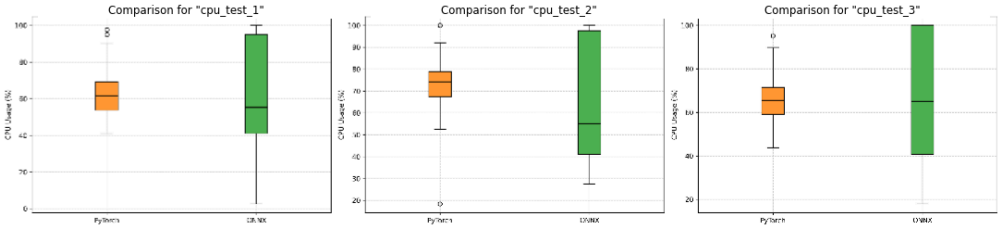
No	Parameter	Model	Average (3 Test)	Standard Deviation
1.	Latency (ms)	PyTorch (`.pt`)	424.11	49.71
		ONNX (`.onnx`)	277.69	43.01
2.	FPS	PyTorch (`.pt`)	2.48	0.23
		ONNX (`.onnx`)	3.66	0.41
3.	CPU (%)	PyTorch (`.pt`)	66.67	9.98
		ONNX (`.onnx`)	65.05	26.74
4.	RAM (%)	PyTorch (`.pt`)	42.44	2.5
		ONNX (`.onnx`)	45.06	1.81
5.	Temperature (˚C)	PyTorch (`.pt`)	65.25	0.96
		ONNX (`.onnx`)	71.79	2.15
6.	Power (Watt)	PyTorch (`.pt`)	14.6	0.9
		ONNX (`.onnx`)		



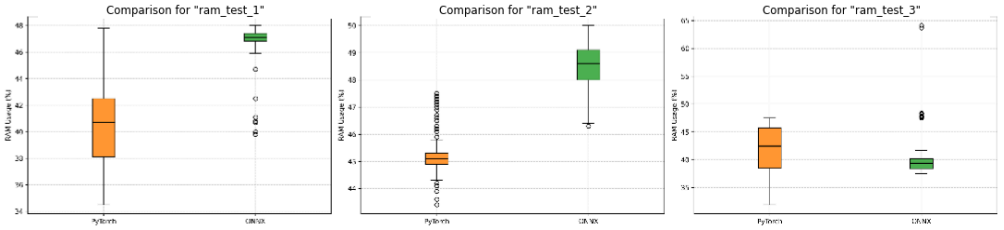
A. Latency PyTorch vs. ONNX



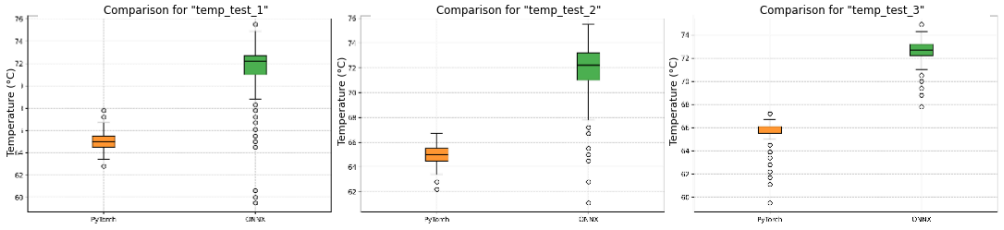
B. FPS PyTorch vs. ONNX



C. CPU Usage PyTorch vs. ONNX



D. RAM Usage PyTorch vs. ONNX



E. Temperature PyTorch vs. ONNX

F. Power Usage

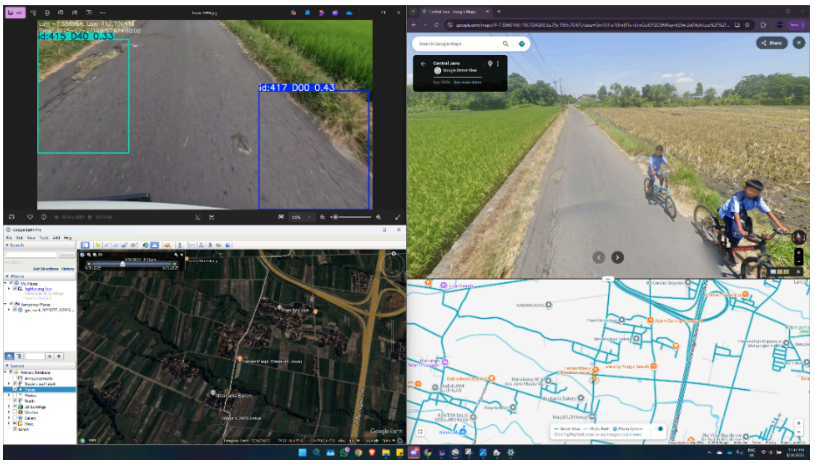
Fig.8 PyTorch vs. ONNX usage

Furthermore, all system efficiency targets were met. During operation, the system consumed an average of 45.06% of RAM and 65.05% of the CPU. The average CPU temperature was 71.79°C, well below the 85°C thermal throttle limit, and the system drew an average of only 14.6W of power. This confirms the system's ability to operate efficiently and stably on low-cost hardware.

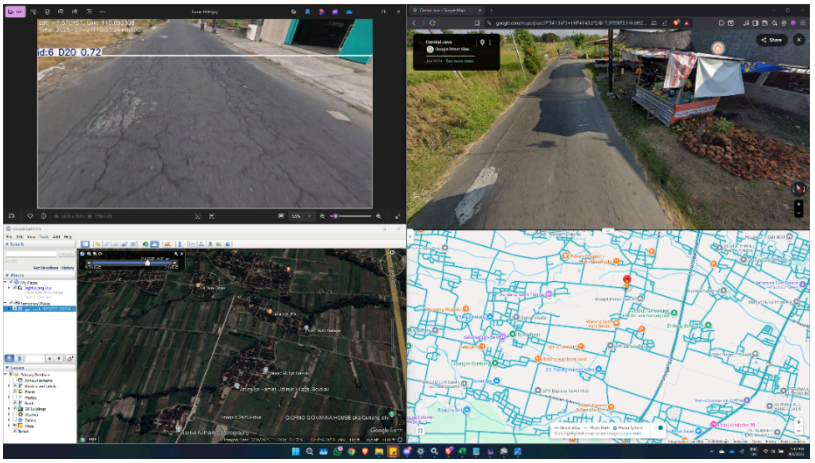
3.3 Field Validation and Geospatial Functionality

The system's geospatial functionality was validated in three distinct field tests (semi-rural, rural, and urban). In all scenarios, the GPS module successfully acquired a satellite lock and accurately recorded the vehicle's path. The system demonstrated its core capability by correctly synchronizing the detected damage instances with their precise geographical coordinates, generating a functional KML track (**Fig. 9**) for post-survey mapping and analysis.

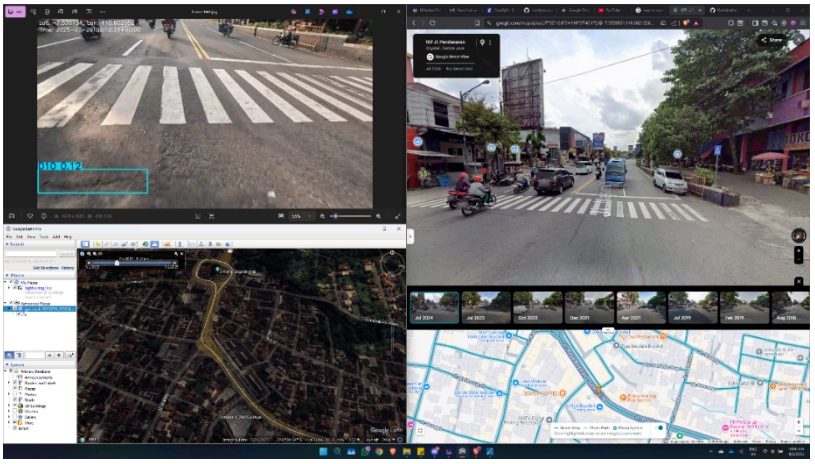
ISSN 3025-3780 (online)
ISSN 3025-4590 (printed)



A. Semi-rural Area



B. Rural Area



C. Urban Area

Fig.9 Geospatial testing

3.4 Discussion

The findings validate that the proposed system successfully addresses the gap identified in the literature. With an mAP@50 of 0.554 and a stable 3.66 FPS on a Raspberry Pi 5, this work demonstrates a balance between accuracy and efficiency on a low-cost platform.

Unlike cloud-dependent systems [2] or post-processing-based systems [3], our platform is fully autonomous and operates in real-time. While Zanevych et al. [6] achieved a higher mAP (0.72), their model was limited

to a single class (potholes). Our system, in contrast, provides a more comprehensive 4-class detection, which is more comparable to the scope of Sidqi et al. [4]. Notably, our mAP of 0.554, validated on the actual target hardware (Pi 5), surpasses the mAP of 0.497 reported by the lightweight YOLO-LWNet model [5], which was benchmarked on a high-power desktop GPU. This confirms the contribution of this research: a functional, low-cost, and real-time edge-computing platform for multi-class road damage monitoring.

4. Conclusion

This study successfully addressed the objective stated in the introduction: to design, implement, and validate an autonomous, real-time road damage detection system on a low-cost edge platform. The results confirm that the system, based on YOLOv11 and optimized with ONNX, is fully functional on a Raspberry Pi 5. It achieved reliable real-time performance, exceeding the target with 3.66 FPS, and successfully integrated precise GPS data. The model demonstrated its capability to detect four damage classes with an mAP@50 of 0.554, confirming the feasibility of this approach.

Based on the findings, the primary limitation identified was the model's high rate of false negatives (undetected damage). Future work should therefore focus on improving model accuracy by retraining the model with a more varied and augmented dataset. Furthermore, the system's prospects can be expanded by developing advanced functionalities, such as classifying damage severity and creating a web-based visualization dashboard to enhance data usability for maintenance stakeholders. Validating the system's performance at different vehicle speeds and exploring AI accelerators for further optimization also remain valuable avenues for future research.

Author Contribution:

All authors contributed equally as main contributors to this paper. All authors read and approved the final paper.

Funding:

This research received no external funding.

Conflicts of Interest:

The authors declare no conflict of interest.

References

[1] D. J. B. M. Kementerian Pekerjaan Umum, “Pedoman Sistem Pemeliharaan Jalan Kota (City Road Management System),” Feb. 2025. [Online]. Available: https://binamarga.pu.go.id/index.php/nspk/detail/02pbm2025-pedoman-sistem-pemeliharaan-jalan-kota-city-road-management-system

[2] C.-C. Hsieh, H.-W. Jia, W.-H. Huang, and M.-H. Hsih, “Deep learning-based road pavement inspection by integrating visual information and IMU,” *Information*, vol. 15, no. 4, p. 239, 2024, doi: 10.3390/info15040239.

[3] K. Mutijarsa, F. A. Alunjati, F. Hidayat, R. Kurnia, and Z. Arsyad, “Smart road damage survey system using machine vision and IoT technology,” *2023 3rd International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA)*, pp. 391–396, 2023, doi: 10.1109/icicyta60173.2023.10428992.

[4] A. J. Sidqi, F. A. Alunjati, U. Elviani, and F. Hidayat, “Development of longitudinal and transversal crack detection feature in automated road pavement condition survey system using YOLO algorithm,” *2024 International Conference on ICT for Smart Society (ICISS)*, pp. 1–5, 2024, doi: 10.1109/iciss62896.2024.10751400.

[5] C. Wu, M. Ye, J. Zhang, and Y. Ma, “YOLO-LWNet: A lightweight road damage object detection network for mobile terminal devices,” *Sensors*, vol. 23, no. 6, p. 3268, 2023, doi: 10.3390/s23063268.

[6] Y. Zanevych, V. Yovbak, O. Basystiuk, N. Shakhovska, S. Fedushko, and S. Argyroudis, “Evaluation of pothole detection performance using deep learning models under low-light conditions,” *Sustainability*, vol. 16, no. 24, p. 10964, 2024, doi: 10.3390/su162410964.

[7] R. Sapkota, M. Flores-Calero, R. Qureshi, C. Badgujar, U. Nepal, A. Poulose, P. Zeno, U. B. P.

ISSN 3025-3780 (online)
ISSN 3025-4590 (printed)

Vaddevolu, S. Khan, M. Shoman, H. Yan, and M. Karkee, "YOLO Advances to Its Genesis: A Decadal and Comprehensive Review of the You Only Look Once (YOLO) Series," *Artificial Intelligence Review*, vol. 58, Art. no. 274, 2025, doi: 10.1007/s10462-025-11253-3.

[8] S. Jana, A. I. Middya, and S. Roy, "Mobile Crowd Sensing for Road Anomaly Detection Using Deep Learning," *Spatial Information Research*, vol. 33, Art. no. 51, 2025, doi: 10.1007/s41324-025-00651-y.

[9] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, and Y. Sekimoto, "RDD2022: A Multi-National Image Dataset for Automatic Road Damage Detection," *Geoscience Data Journal*, 2024, doi: 10.1002/gdj3.260.

[10] D. Minott, S. Siddiqui, and R. J. Haddad, "Benchmarking Edge AI Platforms: Performance Analysis of NVIDIA Jetson and Raspberry Pi 5 with Coral TPU," in *2025 IEEE SoutheastCon*, 2025, pp. 1384–1389, doi: 10.1109/SoutheastCon56624.2025.10971592.