

**19L620 - INNOVATION PRACTICES
PAVEMENT ANAMOLY DETECTION**

ANIRUDH RAMKUMAR (22L207)

DHEERAJ RAJESH (22L214)

LAKSHMANAN A R (22L231)

NAHUL AARIYA D K (22L238)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

of Anna University



April 2025

**DEPARTMENT OF ELECTRONICS AND
COMMUNICATION ENGINEERING**

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

ACKNOWLEDGEMENT

We express our heartfelt gratitude to our Principal, Dr. K. Prakasan, for providing us with the opportunity and the necessary facilities to successfully carry out this project work.

We would also like to extend our sincere thanks to Dr. V. Krishnaveni, Professor and Head, Department of Electronics and Communication Engineering, for her constant encouragement and invaluable support throughout the project.

We are deeply grateful to our Programme Coordinator, Dr. M. Santhanalakshmi, Associate Professor, Department of ECE and our Tutor, Dr. K. Thiyagarajan, Assistant Professor (Senior Grade), Department of ECE, for their guidance, constructive feedback, and consistent approachability which greatly contributed to the progress of our work.

Our special thanks to Dr. K. Vasanthamani, Assistant Professor (Senior Grade), Department of ECE and Dr. S. Mohandass, Assistant Professor, Department of ECE, for their insightful suggestions and continuous encouragement, which played a key role in the completion of this project.

Finally, we wish to thank all our seniors and fellow students from the department for their support and cooperation, which helped us complete the project smoothly and without any hindrance.

ABSTRACT

Road and pavement safety plays a vital role in day-to-day travel experience. In India alone, the year 2020 recorded over 3,72,181 road accidents, averaging 42 accidents per hour. Among these, 5,626 accidents were caused by potholes. These incidents require proper detection and covering of potholes and manholes. However, manual inspection of potholes is time-consuming and labor-intensive, with cities like Mumbai and Delhi reporting over 900 and 1357 potholes respectively.

The proposed project focuses on automated detection and identification of road anomalies (potholes) to avoid this issue. The system utilizes Raspberry Pi 5 for real-time video streaming, pothole detection and GPS logging, with NodeMCU microcontroller handling the rover movements via the Blynk app for remote operation.

The pothole detection model is YOLOv11n which is trained on a custom dataset of 300 images annotated using Label Studio. The model is exported to NCNN format for faster inference on Raspberry Pi 5. The achieved average fps was between 6 and 7 as compared to 3 with YOLOv1n and accuracy of ~65%. The model includes a custom inference code to log the GPS coordinates using google client API and google services, whenever a pothole is detected.

The novelty of this project is that it eliminates the need of manual, time-consuming and labor-intensive inspection of roads. The project has potential integration on existing navigation tools such as Google maps to include the location of the potholes, which enables a better route determination and safe transportation system.

CONTENTS

| CHAPTER | Page No. |
|--|-------------|
| List of Figures | (i) |
| List of Tables | (ii) |
| 1. INTRODUCTION | 1 |
| 1.1. Scope of the project | 1 |
| 1.2. Need for the project | 1 |
| 1.3. Problem statement | 2 |
| 1.4. Objectives | 2 |
| 2. LITERATURE SURVEY | 3 |
| 2.1. Smart road safety system: detecting potholes, sharp turns, speed bumps, and signboards for driver safety | 3 |
| 2.2. Real-time object detection on a raspberry pi | 4 |
| 2.3. Review on road pothole detection mechanisms | 4 |
| 2.4. Smart detection and reporting of potholes via image-processing using raspberry-pi microcontroller | 5 |
| 2.5. Deep learning-based detection of potholes in Indian roads using YOLO | 5 |
| 2.6. Pothole detection using deep learning: a real-time and ai-on-the-edge Perspective | 6 |
| 2.7. Application of various yolo models for computer vision-based real-time pothole detection | 7 |
| 2.8. MCUBench: a comprehensive benchmark of yolo-based object detectors for microcontroller units | 7 |
| 2.9. Real-time pavement damage detection with damage shape | 8 |
| 2.10. Object detection using yolo: challenges, architectural successors, datasets, and applications | 8 |
| 2.11. Detection of potholes using machine learning and image processing | 9 |
| 2.12. Real-time detection of potholes using image processing | 10 |
| 2.13. IoT based real time pothole detection system using image processing techniques | 11 |

| | |
|---|-----------|
| 2.14. Inference | 12 |
| 3. METHODOLOGY | 14 |
| 3.1. Overview | 14 |
| 3.2. System level block diagram of road anomaly detection bot | 14 |
| 3.3. Hardware overview of bot | 15 |
| 3.4. Software algorithm for pothole detection | 15 |
| 4. IMPLEMENTATION & RESULTS | 17 |
| 4.1. System requirements | 17 |
| 4.1.1. Hardware requirements | 17 |
| 4.1.2. Software requirements | 17 |
| 4.2. Hardware implementation of rover | 18 |
| 4.3. Blynk app GUI for rover control | 19 |
| 4.4. Pothole detection – ML model | 20 |
| 4.4.1. Overview of YOLOv11 & NCNN format for Raspberry PI 5 | 20 |
| 4.4.2. Pothole dataset collection & annotation | 20 |
| 4.5. Training & testing the pothole detection model | 21 |
| 4.6. Logging of GPS coordinates | 22 |
| 4.7. Results observed | 23 |
| 5. CONCLUSIONS & FUTURE WORK | 25 |
| Bibliography | 26 |
| Appendix A | 29 |
| Appendix B | 32 |

LIST OF FIGURES

| Figure No. | Name | Page No. |
|------------|---|----------|
| 1 | Road anomaly detection bot - System block diagram | 14 |
| 2 | Hardware module for pothole detection | 15 |
| 3 | Hardware module for rover control | 15 |
| 4 | Flow of real-time pothole detection | 16 |
| 5 | Software for Rover control | 16 |
| 6 | Hardware setup of the rover for detecting potholes | 18 |
| 7 | Blynk App GUI for rover control | 19 |
| 8 | Sample image from the dataset | 21 |
| 9 | Results from testing pothole model using Google Colab | 21 |
| 10 | Sample pothole detected image from testing the model | 22 |
| 11 | Retrieving and logging GPS coordinates | 22 |
| 12 | Data logging in Google sheets | 22 |
| 13 | Training and validation results | 23 |
| 14 | Normalized Confusion matrix of the trained ML model | 23 |
| 15 | Testing of NCNN-YOLOv11n model on Raspberry Pi 5 | 24 |

LIST OF TABLES

| Table No. | Name | Page No. |
|-----------|--|----------|
| 1 | Hardware requirements for rover & object detection | 17 |
| 2 | Software requirements for GUI & ML model | 17 |
| 3 | L298N motor driver pin configuration | 18 |
| 4 | Value (Datastream) sent from Blynk app | 19 |

CHAPTER 1

INTRODUCTION

1.1 SCOPE OF THE PROJECT

The scope of this project involves the design, development, and demonstration of a semi-autonomous road anomaly detection system specifically focused on pothole identification using embedded hardware and AI-based computer vision. The system is centered around the Raspberry Pi 5 platform, which features a powerful 4-core ARM Cortex-A76 processor, 8GB RAM, and a VideoCore VI GPU capable of handling light to moderate real-time inference workloads. An external camera module is used to capture road surface video footage, which is analyzed using a custom-trained YOLOv11n object detection model. This model, optimized through the NCNN inference engine, allows for fast processing on the constrained hardware of the Raspberry Pi while maintaining decent accuracy and detection speed. The use of edge computing ensures that detection is done locally without the need for constant internet connectivity.

Complementing the visual detection module is a NodeMCU ESP8266-based control unit that manages rover mobility through an L298N motor driver and provides wireless connectivity via the Blynk app, enabling remote control of the robot. The scope also includes integrating GPS functionality for precise geolocation of detected anomalies, with automated logging of coordinates and timestamps to a Google Sheet using Google APIs. The system is scalable and modular, allowing it to be adapted for a range of road anomalies including cracks, manholes, or surface irregularities. It can be mounted on autonomous vehicles, municipal inspection bots, or public service fleets, forming a critical part of a smart infrastructure monitoring ecosystem. The project not only explores hardware-software integration but also demonstrates how AI and IoT can converge to offer a robust and affordable solution for civic road maintenance and transportation safety.

1.2 NEED FOR THE PROJECT

The need for this project stems from the alarming number of accidents, vehicle damages, and public safety hazards caused by poorly maintained roads. In countries like India, potholes have become a major concern, with thousands of accidents and fatalities reported annually due to unrepaired road damage. For example, in 2020 alone, over 5,600 accidents were attributed directly to potholes[1], demonstrating the critical impact such anomalies have on road safety. Municipalities struggle with outdated and inefficient road inspection mechanisms, which rely heavily on manual fieldwork and citizen complaints. These manual systems are not only slow and reactive but also fail to detect and respond to damages in a timely manner, leading to compounding infrastructure issues and increased repair costs.

While some cities have piloted sensor-based monitoring or vehicle-integrated crowdsourced data collection, these methods often require high-end hardware, constant

calibration, and robust backend infrastructure—making them impractical for broad deployment, especially in low-resource settings. As urban sprawl continues, the burden on road networks increases, intensifying the demand for a real-time, cost-efficient solution to detect and report anomalies. There is thus a significant gap between available solutions and practical, scalable implementation. The need for a lightweight, AI-enabled system that performs localized detection and automated reporting becomes apparent. This project directly addresses that gap by proposing a real-time pothole detection system that is autonomous, portable, and capable of integrating with existing digital infrastructure, ensuring safer commutes and better-maintained roadways for the public.

1.3 PROBLEM STATEMENT

India's road network, one of the largest in the world, plays a vital role in connecting people, goods, and services across its vast geography. However, poor road maintenance and lack of timely repairs have contributed significantly to traffic delays, vehicle wear and tear, and accidents. Among the various types of road damage, potholes are one of the most common and hazardous anomalies. Despite multiple civic bodies allocating substantial funds for road repairs, the manual detection and documentation of these potholes remain inefficient and sporadic. A report by the Times of India reveals that in the last five years, pothole-related incidents have claimed at least 13 lives in Bengaluru alone, with BBMP (the local governing body) spending over ₹215 crores on roadworks during the same period[1]. Another report from DNA India states that nearly every third road in Chennai suffers from pothole issues[2], highlighting the scale of the problem.

The definition of the problem lies in the gap between road damage occurrence and its timely detection and resolution. Traditional inspections are carried out periodically, meaning newly formed potholes may go unnoticed for days or weeks. Additionally, reliance on human feedback via complaint portals or hotlines results in underreporting and data inconsistency. Hence, there is a clear need for a proactive, data-driven approach to detect potholes as soon as they appear. An automated, mobile system equipped with real-time object detection and GPS logging capabilities can bridge this gap by constantly monitoring road surfaces, detecting anomalies, and instantly reporting them to a centralized database or dashboard. The challenge is to build such a system that is affordable, reliable, and efficient enough to be deployed widely, including in semi-urban and rural regions.

1.4 OBJECTIVES

The objective of this project is to develop a semi-autonomous mobile rover system that can detect road anomalies, particularly potholes, in real time using embedded systems and artificial intelligence. The core processing is handled by a Raspberry Pi 5, which utilizes a lightweight YOLOv11n object detection model, trained on a custom dataset and exported to the NCNN format for optimized inference on low-power hardware. The system processes video data from a mounted camera to identify potholes while maintaining a balance between accuracy (~65%) and speed (6–7 FPS). In addition to detection, the system is equipped with GPS logging functionality that captures the coordinates of each identified pothole and uploads the data to Google Sheets using Google API services, enabling centralized tracking and review. A NodeMCU microcontroller, paired with an L298N motor driver and controlled via the Blynk mobile app, manages the movement of the rover, making it remotely operable and adaptable to various terrains. This project aims to create a cost-effective, scalable, and modular solution for road maintenance authorities to automate pothole detection and reporting, ultimately improving commuter safety and enabling timely repairs through smarter infrastructure monitoring.

CHAPTER 2

LITERATURE SURVEY

2.1 SMART ROAD SAFETY SYSTEM: DETECTING POTHOLES, SHARP TURNS, SPEED BUMPS, AND SIGNBOARDS FOR DRIVER SAFETY[3]

According to this paper, the authors have used YOLOv8 to build a smart system that seamlessly fits into vehicles, providing real-time detection and understanding of road features. The system consists of a user interface and application interface. The user interface deals with capturing the visual data (image) and alert notification, whereas, the application system deals with processing, extraction, identification, and decision making. The features or parameters that have been considered are sign boards, speed bumps, potholes and sharp turns. A custom dataset of 15,025 annotated road images was created, featuring traffic signs, potholes, speed bumps, and sharp turns under varying lighting and weather conditions. The end result achieved by using the state-of-the-art YOLOv8 model, the mAP (mean Average Precision) is 96% for potholes and 94% altogether.

Methods used:

The work proposed in this paper consist of capturing visual data, processing, extraction identification and decision making. They've used pi camera as the eye of the project and it captures the visual data. Roboflow, a platform for training computer vision models and ultralytics – YOLOv8, a real-time object detection and image segmentation model, was used as the primary software and framework.

Metrics used for evaluation:

- mAP: It is the aggregate precision of the model considering all cases.
- Precision: The model's ability to prevent false alarms is indicated by the ratio of true positive detections to total detections.
- Recall: This metric measures the proportion of detections of true positives to the overall amount of ground truth objects.

Challenges in implementation:

The main challenged faced as mentioned in the paper is the moderate level of accuracy in localizing and classifying sign boards with mAP of 82.2%. However, the precision is 91.2% and recall is 82%. This indicates room for improvement in both precision and recall for Sign Boards detection.

2.2 REAL-TIME OBJECT DETECTION ON A RASPBERRY PI[4]

This thesis investigates the feasibility of implementing real-time object detection on a Raspberry Pi 3 Model B+, an embedded device with limited processing power. It evaluates SSD (Single Shot Detector with MobileNetV2 backbone) and YOLOv3-tiny models, analysing speed (inference time, frames per second) and accuracy under different input sizes and conditions.

Key Features and Findings

- Models Evaluated: SSD (MobileNetV2) and YOLOv3-tiny, both pre-trained on the COCO dataset.
- Performance Analysis:
 - SSD outperformed YOLOv3-tiny in accuracy and speed.
 - Larger input sizes improved accuracy but reduced speed.
 - Neither model achieved sufficient speed for high-performance real-time applications.
- Experiments Conducted:
 - Throughput and inference time for varying input resolutions.
 - Accuracy tests for detecting a person at different distances, relevant for home security and surveillance.

Challenges

The Raspberry Pi's limited processing power restricts its ability to achieve high-speed object detection, especially for applications requiring high real-time performance.

2.3 REVIEW ON ROAD POTHOLE DETECTION MECHANISMS[5]

This paper presents a comparative analysis of various pothole detection systems implemented using embedded platforms, image processing, and deep learning models. Emphasis is placed on affordable and scalable systems suited for developing regions. The study compares Pi-based microcontroller platforms, mobile applications, and AI-powered approaches to improve road safety and automate road anomaly detection.

Methods Used:

Raspberry Pi and Embedded Systems: Pi + Image Processing - Utilizes a camera mounted on the vehicle windshield. Captures images at 8 fps, preprocesses with object removal, and applies Canny Edge Detection. Pothole data (including GPS) is sent via Wi-Fi to a web server.

ARM Controller-Based System: Incorporates infrared and ultrasonic sensors with an Android interface. Detects pothole depth, hump height, and shares the location data via GPS.

Smartphone & App-Based Detection: Smartphone Sensors: Mounted on car dashboard. Uses accelerometer and gyroscope to detect road issues. Apps collect sensor data and manually log potholes for training and validation.

Deep Learning & AI-Based Systems: Applies convolutional layers for feature extraction and fully connected layers for classification. CNN, R-CNN, F-CNN, and YOLO for recognizing potholes, humps, faded road signs, etc., from smartphone images.

Challenges in Implementation:

- Night and adverse weather detection limitations in conventional cameras

- Internet dependency for cloud-based reporting
- Power and performance trade-offs in Pi-based and embedded systems
- Model generalization for various road types and regions
- Manual annotation and sensor calibration for accurate training data

2.4 SMART DETECTION AND REPORTING OF POTHOLE VIA IMAGE-PROCESSING USING RASPBERRY-PI MICROCONTROLLER[6]

A low-cost, portable, and effective real-time pothole detection and reporting system is presented in this paper. Using a Raspberry Pi microcontroller with a camera and GPS module, the system captures and processes road images to detect potholes, then sends the data (image + GPS location) to a web server without relying on smartphones. The system was tested under both ideal and real-world driving conditions and achieved high accuracy.

Methods Used:

The system utilizes a Raspberry Pi microcontroller, powered via the vehicle's lighter socket, as the central processing unit. A camera module, mounted behind the rear-view mirror, captures video at 8 frames per second for continuous monitoring of the road surface. Image processing is implemented using Python and OpenCV. The process involves removing irrelevant elements such as pedestrians and sidewalks, followed by pothole detection through Canny edge detection, contour extraction, and analysis based on colour and area. For automatic reporting, a GPS module retrieves the location of the detected pothole, and the image along with the coordinates is uploaded via Wi-Fi to Dropbox, which subsequently syncs the data to a web server. The detection and upload status is indicated using onboard LEDs, which also serve to alert the driver. Additionally, a web interface is developed to display the pothole reports with images and location details for remote monitoring.

Metrics Used for Evaluation:

Detection accuracy was found to be 100% under ideal conditions and 87.45% in non-ideal, real-world environments. Sensitivity, which measures the system's ability to correctly identify actual potholes, averaged 96.61%, while specificity, indicating the system's effectiveness in avoiding false positives, averaged 92.85%. The reporting mechanism showed a 100% success rate, reliably uploading all detected pothole data to the server. In terms of processing efficiency, the average response time per frame was measured at 0.1123 seconds when no pothole was present, and 0.1244 seconds when a pothole was detected, demonstrating real-time capability with minimal delay.

Challenges in Implementation:

False Positives: Reflections on windshield

- Environmental Noise: Vehicles and road clutter affected non-ideal testing accuracy
- Stability: Camera placement was critical for consistent detection
- Internet Dependency: Reporting requires stable Wi-Fi for real-time operation

2.5 DEEP LEARNING BASED DETECTION OF POTHOLE IN INDIAN ROADS USING YOLO[7]

According to this paper, the authors developed a custom dataset of 1,500 annotated images focused on Indian roads. This dataset was used to train and compare the performance of YOLO models, including YOLOv3, YOLOv2, and YOLOv3-tiny. The

evaluation metrics included mean Average Precision (mAP), precision, and recall, with the model demonstrating reasonable accuracy in detecting potholes across various images. The study highlighted potential future applications, such as real-time implementation using Raspberry Pi and GPS integration. However, challenges such as complex model architectures, lengthy training times, risks of overfitting, and limited generalization capabilities were noted.

Methods used:

The flow of detecting potholes as mentioned in the paper is as follows:

Dataset creation → Annotation → Train set → K – means++ clustering algorithm →

Tuning architecture → Training → Test set → Validation and evaluation metrics.

Assumptions made:

The work assumes that a dedicated GPU such as GTX GeForce 1060 (used in the work) is available to decrease the training time and to increase the computation speed.

Metrics used:

To annotate the following parameters are used: class ID, centre X, centre Y, width and height. To compare the YOLO models the following metrics are used: mAP, IoU, Precision and Recall.

Challenges in implementation:

The main challenge faced is the variety of dataset, i.e., the potholes aren't of fixed size and hence, the dataset used to train the model should contain images of potholes taken under different angles, distances, light and climate condition.

2.6 POTHOLE DETECTION USING DEEP LEARNING: A REAL-TIME AND AI-ON-THE-EDGE PERSPECTIVE[8]

The main goal of this paper is to investigate the potential of deep learning models and deploy three superior deep learning models on edge devices for pothole detection. They have used an AI kit (OAK-D) on a Raspberry Pi single-board computer as an edge platform for pothole detection. This paper claims that among various pavement failures, there are numerous studies on pothole detection because they are harmful to cars and passengers and could result in an accident. They have proposed a model that uses morphological processing to first assemble wavelet energy for pothole detection, and then the Markov random field model to segment the detected pothole images and extract the pothole edge. The experiment is conducted on a dataset of images with potholes in various road conditions. More than 120 MATLAB pavement images were tested and trained using this methodology. The technique has achieved 86point 7 percent accuracy, 83point 3 percent precision, and 87point 5 percent recall. The creation of edge devices, YOLOv4 as the most accurate pothole detection model, and Tiny-YOLOv4 as the most accurate pothole detection model for real-time pothole detection with 90% detection accuracy.

Methods used:

Before being sent to deep learning models like the YOLO family and SSD for custom model training, the annotated data is divided into training and testing data. The weights acquired following training aid in assessing the model's performance using test data. After that, the custom weights are transformed into the OpenVino IR format so that the OAK-D and Raspberry Pi can detect in real time.

Challenges in implementation:

The system faced challenges from varying lighting, shadows, and weather, affecting detection accuracy. Limited computational power on edge devices restricted model complexity and real-time performance. High variability in pothole appearance hindered generalization. Ensuring reliable wireless communication and power efficiency during deployment was also difficult.

2.7 APPLICATION OF VARIOUS YOLO MODELS FOR COMPUTER VISION-BASED REAL-TIME POTHOLE DETECTION[9]

This paper claims that three cutting-edge object detection frameworks, i.e., using the image set, experiments are conducted to evaluate the performance of YOLOv4, YOLOv4-tiny, and YOLOv5s in terms of real-time responsiveness and detection accuracy.

Methods used:

The flow of detecting potholes as mentioned in the paper is as follows:

Dataset creation → Train set → image conversion to 416 x 416 pixels → validated until cost function attained a stable state → Training → Test set → Validation and evaluation metrics Assumptions made: It is assumed that the device to be worked on must comprise of Tesla K80 GPU with 12 GB of memory.

Metrics used:

- mAP: It is the aggregate precision of the model considering all cases.
- Precision: The ratio of detections of true positive to the overall number of detections.
- Recall This metric measures the proportion of detections of true positive to the overall number of ground truth objects, reflecting the model's capability to identify all relevant objects within the input data

Challenges in implementation:

The main challenged faced was the different types of datasets, as the potholes comprises of variable sizes and length, which made it difficult to obtain the required accuracy.

2.8 MCUBENCH: A COMPREHENSIVE BENCHMARK OF YOLO-BASED OBJECT DETECTORS FOR MICROCONTROLLER UNITS (MCUS)[10]

MCUBench is a benchmark that evaluates over 100 YOLO-based object detection models optimized for Microcontroller Units (MCUs). The models were tested on the PASCAL VOC dataset across seven distinct MCU platforms, measuring metrics like mean Average Precision (mAP), latency, RAM, and Flash usage. The study addresses challenges in deploying deep learning models on resource-constrained hardware and explores optimization techniques such as advanced detection heads and scaling parameters (resolution, depth, width).

Key Contributions

- Pareto-Optimal Analysis: Balancing accuracy and latency for YOLO models.
- Trade-Off Insights: Highlighting the impact of hardware constraints (memory, latency) on mAP.
- Compact YOLO Variants: Designed for ultra-low-power edge applications.

Challenges:

The main challenge lies in deploying deep learning models on MCUs with limited memory and processing power while maintaining acceptable accuracy and latency.

2.9 REAL-TIME PAVEMENT DAMAGE DETECTION WITH DAMAGE**SHAPE[11]**

Intelligent detection of pavement damage is crucial for road maintenance, enabling timely repairs that extend road lifespan. Existing detection methods struggle to balance accuracy with speed. This study introduces the Fast Pavement Damage Detection Network (FPDDN), designed for real-time, high-accuracy detection of pavement defects. FPDDN integrates a deformable transformer for geometric adaptability, a D2f block to lighten the network and increase speed, and an SFB module to enhance small object detection. The model was tested on the RDD2022 dataset, where it surpassed state-of-the-art models like YOLO v8x in detection accuracy while maintaining a swift inference speed of 1.8ms per image.

Methods used:

- Visual Data Capture: Utilized a pi camera to capture road images.
- Processing and Detection: Employed the FPDDN model which includes:
 - Deformable Transformer: For handling geometric variations in road defects.
 - D2f Block: Based on depth wise separable convolutions to reduce parameters and increase inference speed.
 - SFB Module: For minimizing information loss in small object detection.
 - Software and Framework: Used PyTorch for the deep learning framework.

Metrics used for evaluation:

- F1 Score: Harmonic mean of precision and recall, indicating model performance balance.
- mAP50: Mean Average Precision at IoU threshold of 0.5, measuring detection accuracy.
- mAP50-95: Mean Average Precision across IoU thresholds from 0.5 to 0.95 for detailed accuracy assessment.
- Inference Time: Time taken to process one image, critical for real-time applications.
- Parameters (Params) and FLOPs: To assess model complexity and computational efficiency.

Challenges in implementation:

- Complex Environments: Detection accuracy decreases under conditions like heavy snow or moisture.
- Dataset Limitations: Existing datasets often lack images from diverse, real-world scenarios which can impact model generalization.
- Model Efficiency: Despite increased complexity with SFB and DAttention modules, FPDDN manages to keep inference times low, suitable for real-time applications.

2.10 OBJECT DETECTION USING YOLO: CHALLENGES, ARCHITECTURAL SUCCESSORS, DATASETS, AND APPLICATIONS[12]

Detection of objects is one of the major challenges in CV, evolving significantly with deep learning over the last decade. This paper provides a comprehensive review focused on

single-stage object detectors, particularly the YOLO (You Only Look Once) series, their architectural developments, performance metrics, and applications. Single-stage detectors like YOLO are noted for their speed, although traditionally they lag behind two-stage detectors in accuracy. However, advancements in YOLO's iterations have shown significant improvements in both speed and accuracy. The review discusses the regression formulation of YOLO, its architectural evolution from YOLO v1 to v4, and performance comparisons with both single and two-stage detectors. It also highlights applications where YOLO's real-time capabilities are leveraged.

Methods used:

Regression Approach: YOLO reframes detection of objects as a regression analysis.

Architectural Evolution:

- YOLO v1: Utilizes a network inspired by GoogLeNet with 24 convolutional layers, focusing on predicting class probabilities per grid cell.
- YOLO v2: Introduces concepts like batch normalization, high-resolution classifiers, and anchor boxes, improving both speed and accuracy.
- YOLO v3: Incorporates residual connections and feature pyramid networks for multi-scale detection, enhancing small object detection.
- YOLO v4: Employs Cross-stage partial connections (CSP), Path Aggregation Network (PAN), and Spatial Pyramid Pooling (SPP) for better feature extraction and speed-accuracy balance.

Metrics used for evaluation:

- Detection Accuracy: Measured through mean Average Precision (mAP), which reflects both precision and recall across various IoU thresholds.
- Inference Time: Critical for real-time applications, measured in frames per second (fps).
- Parameters (Params) and FLOPs: To assess model complexity and computational efficiency.

Challenges in implementation:

Small Object Detection: Initial versions of YOLO struggled with detecting smaller objects due to their single-scale approach.

Background Detection: Errors in identifying background can impact detection accuracy, especially for objects with varied appearances.

2.11 DETECTION OF POTHOLE USING MACHINE LEARNING AND IMAGE PROCESSING[13]

The paper proposes a system that automates the detection of potholes on roads using image processing and machine learning. The system aims to assist local municipal authorities by identifying road surface defects early, reducing manual inspection efforts, and improving maintenance efficiency. It uses a convolutional neural network (CNN)-based model for accurate detection and classification of potholes from road images.

Methods used:

The methodology involves a multi-step pipeline that combines hardware and software components to identify road anomalies. Initially, a camera installed on a moving vehicle captures video footage of the road surface. This video is broken down into individual

frames, which are subjected to a series of image preprocessing steps. The images are first converted to grayscale to simplify analysis and reduce computational load. Denoising filters such as Gaussian filters are applied to remove noise, and image downsampling is used to speed up processing. Morphological operations are used to clean up binary images. Edge detection algorithms like the Canny edge detector are employed to highlight potential boundaries and contours of potholes. Feature extraction techniques are used to identify shapes and patterns typical of potholes, which are then fed into a classification algorithm. A convolutional neural network (CNN) is trained on labeled image data—consisting of both pothole and non-pothole images—allowing the model to learn distinguishing features. The CNN includes convolutional layers, ReLU activations, and pooling layers to process the spatial features of input images. Once trained, the model is capable of identifying potholes in real-time image feeds. The detected potholes are then logged along with their GPS coordinates, which are uploaded to a cloud platform for further review and action by relevant authorities.

Metrics used:

To evaluate the performance of the proposed pothole detection model, the authors consider several standard metrics used in image classification tasks. Accuracy is the primary metric mentioned, reflecting the overall percentage of correct predictions made by the model. While specific numeric values for accuracy, precision, and recall are not detailed in the paper, the authors suggest that their system achieved satisfactory classification accuracy on their test dataset. The confusion matrix is mentioned as a tool used to analyze the number of true positives (correct pothole detections), false positives (incorrectly identified potholes), and false negatives (missed potholes). These metrics provide insights into the model's reliability in real-world conditions. However, the evaluation appears to be qualitative rather than deeply quantitative, indicating a potential area for further research and more rigorous benchmarking in future iterations.

Challenges in implementation

The implementation of the pothole detection system comes with several challenges, both technical and practical. One of the major issues is the variability in environmental conditions such as lighting, shadows, and weather, which can significantly affect image quality and the accuracy of pothole detection. For instance, bright sunlight, low-light conditions, or reflections on wet roads may lead to misclassification. Another challenge is the presence of visual noise in the input images—such as road markings, oil spills, or debris—which can confuse the algorithm and lead to false positives. The authors also point out that their dataset was limited in size and diversity, which may affect the generalizability of the model to different road types or geographic locations. Furthermore, deploying the system in real-time with sufficient frame rates and responsiveness requires optimized code and hardware, which can be resource-intensive. Ensuring seamless integration of the camera, Raspberry Pi, Arduino, and cloud components adds to the complexity. Finally, achieving a low false positive rate while maintaining high recall remains a critical concern, as unnecessary alerts may reduce the system's credibility and usefulness.

2.12 REAL-TIME DETECTION OF POTHOLE USING IMAGE PROCESSING[14]

The paper proposes a system for real-time pothole detection on roads using image processing and sensor integration. The focus is on developing an affordable and efficient solution for identifying potholes to improve road safety and maintenance, especially in regions where roads tend to deteriorate quickly. The system employs a Raspberry Pi camera to capture images of the road and uses a combination of image processing techniques, including Gaussian filtering to remove noise and edge detection via the Canny algorithm to identify potholes. The system also integrates ultrasonic and infrared

sensors, enhancing the pothole detection process by providing additional context. Notably, the system avoids the need for complex machine learning models, making it both cost-effective and relatively simple to implement. The real-time capability is a key feature, making this system suitable for practical applications where timely pothole detection is crucial for safety.

Method used:

The proposed system captures road images using a Raspberry Pi camera and processes them through Gaussian filtering to eliminate noise. Edge detection is then applied using the Canny edge detection algorithm, which helps in identifying potholes by highlighting significant features like edges in the image. The system also incorporates ultrasonic and infrared sensors, which complement the image processing by helping detect potholes that may not be easily visible due to poor lighting or partial coverage. The system operates in real-time, ensuring that potholes are detected and flagged promptly as vehicles move along the road. This approach avoids the complexity of training machine learning models, making it a simpler, more cost-efficient solution.

Metrics used:

While the paper does not specifically mention traditional evaluation metrics like accuracy, precision, recall, or F1-score, the system is implicitly evaluated based on its ability to detect potholes in real-time under various environmental conditions. The key performance indicator is the system's real-time processing capability, which allows it to detect potholes as the vehicle moves along the road. The effectiveness of the edge detection algorithm (Canny edge detection) in identifying clear potholes is a significant evaluation point, as well as how well the ultrasonic and infrared sensors contribute to detecting potholes that might be missed in the image processing step.

Challenges in implementation:

There are several challenges in implementing this pothole detection system. One of the primary challenges is the variability of lighting and weather conditions. Changes in daylight or adverse weather such as rain or fog can degrade image quality and interfere with the detection process. Additionally, the ultrasonic and infrared sensors, while helpful, have limitations in detecting small or shallow potholes, especially in low-visibility situations. The computational limitations of the Raspberry Pi also present challenges, particularly when processing high-resolution images or handling multiple sensors. These constraints may result in delays in real-time processing or decreased accuracy, especially in complex road environments.

2.13 IOT BASED REAL TIME POTHOLES DETECTION SYSTEM USING IMAGE PROCESSING TECHNIQUES[15]

The paper addresses the critical issue of road safety by proposing an IoT-based system for real-time pothole detection. The system aims to provide early warnings to drivers and assist authorities in road maintenance. It employs a combination of image processing and sensor technologies to detect potholes, alert drivers, and log the location data for further action.

Method used:

The proposed system utilizes a camera mounted on the vehicle to capture real-time images of the road. These images are processed using MATLAB to detect potholes based on shape and dimension. Additionally, an ultrasonic sensor measures the distance to the road surface, identifying irregularities that indicate potholes. Upon detection, the system alerts the driver via a blinking LED and stores the GPS coordinates of the pothole

using a Bluetooth interface connected to a smartphone. This data is then uploaded to a cloud database using Wi-Fi or 4G connectivity, making it accessible to the public and relevant authorities through mapping platforms like Google Maps or OpenStreetMap.

Metrics used:

The paper does not explicitly mention specific performance metrics such as accuracy, precision, recall, or F1-score. However, the system's effectiveness is evaluated based on its ability to detect potholes in real-time, alert drivers promptly, and log accurate location data for further action. The integration of image processing and sensor data processing is assessed qualitatively through the system's functionality and responsiveness.

Challenges in implementation:

The implementation of the proposed IoT-based pothole detection system faces several challenges. One of the primary issues is the variability of environmental conditions, such as changes in lighting and weather, which can significantly impact the quality of the captured images, making pothole detection less accurate. Additionally, the ultrasonic sensor has limitations in detecting all types of potholes, especially if the road surface is uneven or the sensor alignment is not optimal. The real-time processing requirement for both image and sensor data also places a significant computational burden on the system, which may be difficult to handle with embedded systems that have limited resources. Furthermore, the system's reliance on stable internet connectivity for uploading data to the cloud is a challenge, as areas with poor or no network coverage could hinder the system's functionality. Lastly, integrating the image processing algorithms, sensors, and communication modules in a seamless manner presents technical difficulties, as each component must work reliably together for effective pothole detection and reporting. Despite these challenges, the system remains a promising solution for enhancing road safety and enabling efficient road maintenance.

2.14 INFERENCE

Road anomaly detection bot requires an edge device for running the deep learning model and custom inference application such as user alertness and GPS data logging. However, microcontrollers such as NodeMCU and ESP32 are inefficient for running DL models due to limited computation resources and single board computer such as Nvidia Jetson Nano makes the system expensive. Hence, running DL models, Raspberry Pi 5 provides a better trade-off between performance and cost.

Deep learning models consist of single stage detector such as YOLO and other detectors such as FPDDN providing high accuracy and faster inference on dedicated GPU platforms such as systems with Nvidia RTX GPUs. However, as edge devices such as Raspberry Pi are limited in computational resources, running YOLO and FPDDN models are ineffective. Hence, these models after training have to be exported to lightweight architectures such as tensorflow lite or NCNN to provide faster inference with accuracy around 70 to 80%.

The chosen methodology is well-justified based on insights from the literature, which emphasize the limitations of deploying full-scale deep learning models like YOLOv8 on resource-constrained edge devices such as the Raspberry Pi. To address these challenges, the project employs a lightweight NCNN-based YOLOv11n model, ensuring faster inference and real-time performance without compromising significantly on accuracy. The use of Raspberry Pi 5 offers a cost-effective balance between processing

capability and affordability, as opposed to high-end solutions like Jetson Nano or low-capability microcontrollers. Additionally, integrating GPS-based logging and real-time anomaly reporting through Google Sheets enhances system utility and aligns with proven practices in previous studies. The modular architecture—dividing video capture, detection, location logging, and navigation—enables efficient task handling, while the GUI-based semi-autonomous control via Blynk App provides flexibility and manual override when needed, making the methodology both robust and scalable for real-world road anomaly detection applications.

CHAPTER 3

METHODOLOGY

3.1. OVERVIEW

The proposed system for road anomaly detection is divided into four stages: video capture, pothole detection, GPS logging, and rover control. In the first stage, the rover's camera captures video frames in real-time, which are then analyzed by a machine learning model (NCNN-YOLOv11n) to identify potholes of varying sizes, shapes, and positions. If a pothole is detected, the system logs the GPS coordinates and sends this data to road authorities. The final stage involves controlling the rover's navigation using a custom GUI, which allows for precise speed and direction control through a NodeMCU that communicates with the motor driver and motors.

The hardware setup consists of two primary components: pothole detection and rover control. The pothole detection hardware includes a Raspberry Pi 5 and a camera for capturing and processing video, while the rover control hardware includes a NodeMCU, L298N motor driver, two motors, and a 12V battery. The software architecture includes a pothole detection algorithm that processes video frames and logs GPS data using the Google Client API. For rover control, the Blynk app is used to send speed and direction commands to the NodeMCU, which decodes and relays the instructions to the motor driver, enabling precise movement of the rover.

3.2. SYSTEM BLOCK DIAGRAM OF ROAD ANOMALY DETECTION BOT

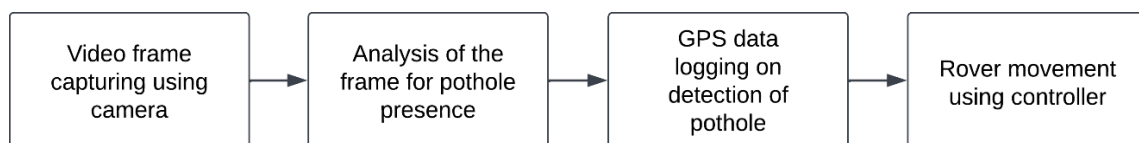


Fig 3.1 Road anomaly detection bot - System block diagram

The proposed system for road anomaly detection bot comprises of four stages as shown in Fig 3.1. The first stage involves the capturing of video frame using the external camera. This is done in real-time with an average frame per second of 6. The second stage involves the analysis of the frame. The frame is inspected whether a pothole is present or not. This is performed using ML model for identification of potholes of different size, shape and position with better accuracy. The third stage involves logging of GPS coordinates. If the model detects a pothole in the frame, the coordinates of the rover is sent to the respective road authorities. The final stage involves navigating the rover along the road using the given GUI for precise speed and direction control.

3.3. HARDWARE OVERVIEW OF ROVER

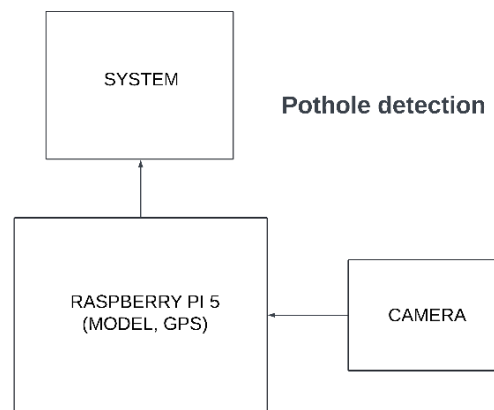


Fig 3.2 Hardware module for pothole detection

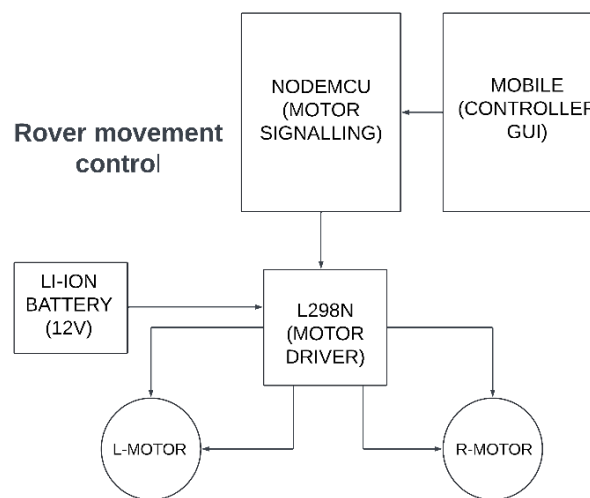


Fig 3.3 Hardware module for rover control

The hardware setup consists of two parts: pothole detection and rover control. As in Fig 3.2, the hardware for pothole detection comprises of Raspberry Pi 5, camera and a system (Laptop/PC/Mobile). The camera is for capturing the video frames for analysis of presence of potholes using ML model present in the Raspberry Pi 5. The system is used to display the video streaming from the rover to the end user for proper navigation. The hardware setup for rover control consists of NodeMCU, L298N motor driver, two motors and a 12V Li-ion battery as in Fig 3.3. The NodeMCU acts as the central control signal receiving device from GUI in the mobile. The L298N motor driver is used to drive and control the left and right motors based on the speed and direction signals from the NodeMCU. The motor driver is powered by a 12V Li-ion battery necessary to drive the motors.

3.4. SOFTWARE ALGORITHM FOR POTHOLE DETECTION

The software algorithm and flow are divided into two parts corresponding to the hardware setup. The first part is for real-time pothole detection. As in Fig 3.4, the pothole detection is performed using the NCNN-YOLOv11n^[25] high performance object detection Neural Network model. The video frames captured by the camera is processed by the ML model to identify the presence of the pothole in the captured frame. On detecting the pothole, the GPS coordinates of the rover is sent to a Google sheet

maintained by the respective road authorities. The GPS data logging is achieved using the Google Client API available in Python packages. The software for rover control comprises of custom designed GUI using Blynk App and motor driver control code using NodeMCU. The Blynk app is used to send the speed and direction control signals to the NodeMCU via Wi-Fi. The NodeMCU decodes the signal and relays them to the L298N motor driver for desired navigation.

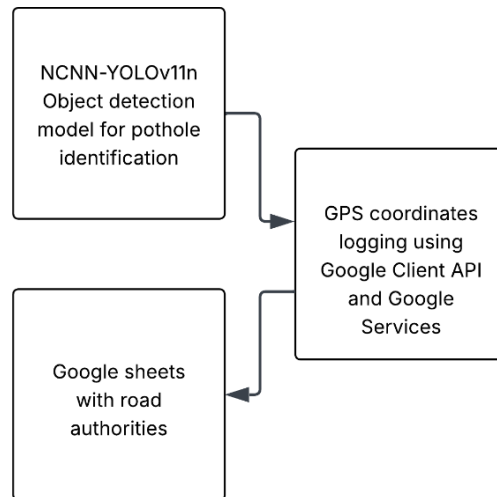


Fig 3.4 Flow of real-time pothole detection

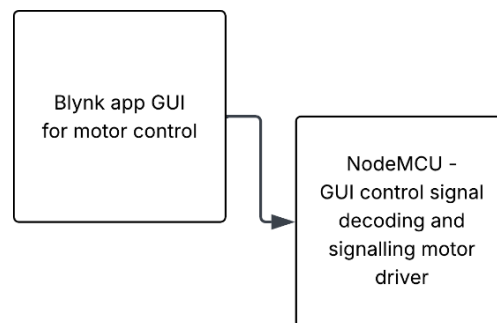


Fig 3.5 Software for Rover control

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 SYSTEM REQUIREMENTS

4.1.1 HARDWARE REQUIREMENTS

The hardware requirements for implementing the rover and pothole detection model includes the microcontroller, power supply, drivers and other accessories.

Table 4.1 Hardware requirements for rover & object detection

| S. No. | Item |
|--------|-----------------------------|
| 1 | Raspberry Pi 5 |
| 2 | NodeMCU (ESP8266) |
| 3 | Motor Driver (L298N) |
| 4 | Voltage regulator (LM2596S) |
| 5 | Rover kit |
| 6 | Web camera |
| 7 | Li-ion batteries (12V) |

4.1.2 SOFTWARE REQUIREMENTS

The software requirements for implementing the ML Model, rover control GUI and controller includes application for training the ML, building the GUI and libraries used.

Table 4.2 Software requirements for GUI & ML model

| S. No. | Software |
|--------|------------------------------|
| 1 | Raspberry Pi OS |
| 2 | Arduino IDE (C language) |
| 3 | Thonny IDE (Python language) |
| 4 | Libraries |
| a | ESP8266WIFI |
| b | BlynkSimpleESP8266 |
| c | Ultralytics |
| d | CV2 |
| e | GoogleAPIClient |
| 5 | Google Colab |
| 6 | Blynk App |

4.2 HARDWARE IMPLEMENTATION OF ROVER

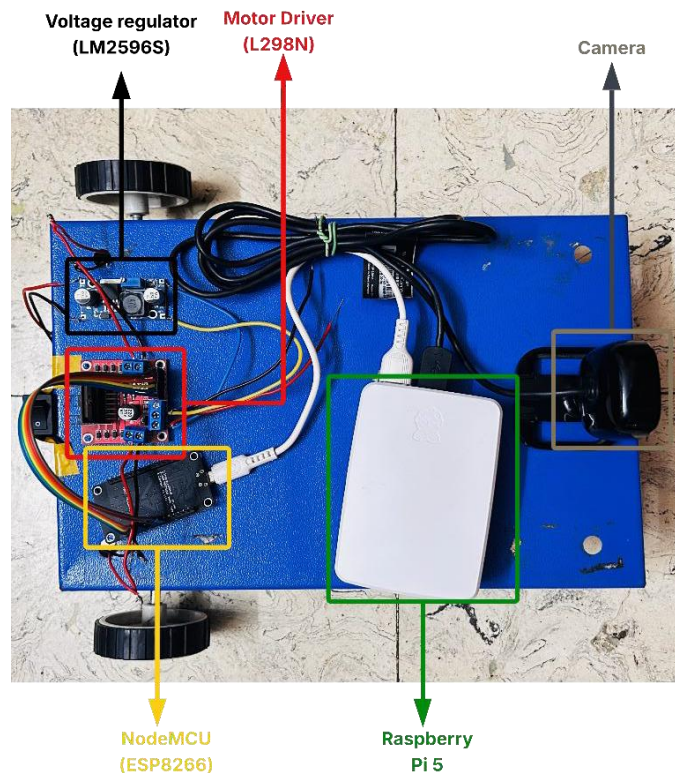


Fig 4.1 Hardware setup of the rover for detecting potholes

The hardware implementation of the road anomaly detection bot is as shown in Fig 4.1. it includes the Raspberry Pi 5 for running the NCNN-YOLOv11n pothole detection model and NodeMCU, L298N motor driver for controlling the rover.

The web camera for video frame capturing is interfaced with the Raspberry Pi 5 using USB 3.0 connectivity. The Raspberry Pi 5[18] is powered (not shown in Fig 4.1) using a 5.1V, 5A, 27W power supply adapter from Raspberry Pi. The Raspberry Pi 5 along with the camera is responsible for capturing and analyzing the video frame for presence of pothole using ML.

NodeMCU along with L298N motor driver powers the precise navigation of the rover. The NodeMCU is used to connect with the mobile hosting the control GUI (Blynk App) via Wi-Fi. It receives and decodes the control signals from the mobile and send it to the motor driver to move the rover in specified direction and speed. Table 4.3 summaries the pins of motor driver used and the interfacing with NodeMCU.

Table 4.3 L298N motor driver pin conFIGuration

| S. No. | Pin | Purpose | NodeMCU pin |
|--------|-----|------------------------------------|-------------|
| 1 | IN1 | Direction control of right motor | D2 |
| 2 | IN2 | | D3 |
| 3 | ENA | Speed control of right motor (PWM) | D1 |
| 4 | IN3 | Direction control of left motor | D4 |
| 5 | IN4 | | D5 |
| 6 | ENB | Speed control of left motor (PWM) | D6 |

The motor driver connected to the NodeMCU receives control signals through four input pins: IN1, IN2, IN3, and IN4. To move the motor in the forward direction, the NodeMCU sets IN1 and IN3 to 3.3V (HIGH) while IN2 and IN4 remain at 0V (LOW). For backward motion, IN2 and IN4 are set to 3.3V, and IN1 and IN3 are held at 0V. To turn the motor towards the right, only IN1 is set to 3.3V, while the remaining inputs (IN2, IN3, and IN4) are kept at 0V. Conversely, a left turn is achieved by setting IN3 to 3.3V and maintaining IN1, IN2, and IN4 at 0V. To stop the motor, all four input pins—IN1, IN2, IN3, and IN4—are set to 0V. These control signal combinations allow precise directional control of the motor via the NodeMCU.

4.3 BLYNK APP GUI FOR ROVER CONTROL

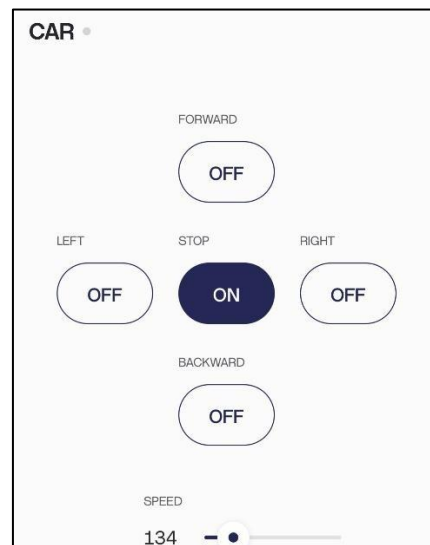


Fig 4.2 Blynk App GUI for rover control

The road anomaly detection bot is a semi-autonomous rover with manual control over its direction and speed. The control keys are provided by the custom designed GUI using Blynk app. As shown in Fig 4.2, there five direction and one speed control keys. The direction signal is sent as an integer (0 to 5) as tabulated in Table 4.5. The speed value is also sent as an integer value between 0 and 1023. The values that are sent from the Blynk app are called as 'Datastream'

Table 4.4 Value (Datastream) sent from Blynk app

| S. No. | Control | Value sent |
|--------|----------|------------|
| 1 | Stop | 0 |
| 2 | Forward | 1 |
| 3 | Right | 2 |
| 4 | Backward | 3 |
| 5 | Left | 4 |
| 6 | Speed | 0 to 1023 |

Table 4.4 summarizes the value sent by Blynk app via Wi-Fi is received by the NodeMCU connected to the same Wi-Fi. The NodeMCU decodes the received value and sends appropriate signal to the motor driver.

4.4 POTHOLE DETECTION – ML MODEL

4.4.1 OVERVIEW OF YOLOv11 & NCNN FORMAT FOR RASPBERRY PI 5

YOLO or You Only Look Once is a state-of-the-art single stage detector designed for high accuracy and fast inference real-time object detection. YOLOv11 is the second to latest YOLO version with higher accuracy and faster inference due to the modification in its architecture to include C3k2 (Cross Stage Partial with kernel size of 2).

However, as mentioned in section 2.3, YOLO in spite of its high accuracy, is only suitable for systems with dedicated GPUs for fast inference. The solution to this is to export the YOLOv11 pytorch model to NCNN format.

NCNN[25] is a lightweight framework designed especially for edge devices such as Raspberry Pi 5. It has quantization support to which reduces the precision of the weights. This results in a trade-off between accuracy and inference time.

To convert a YOLOv11n model trained on a custom pothole dataset to the NCNN format using Google Colab, first set up the environment by installing the necessary dependencies and cloning the NCNN repository. Build NCNN from source by following the instructions in the repository. After the environment is prepared, export the YOLOv11n model to the ONNX format using the YOLOv11 export script. This step involves either cloning the YOLOv11 repository or directly using your custom-trained model. Once the model is in ONNX format, convert it to the NCNN format using the `onnx2ncnn` tool, which will generate the required `.param` and `.bin` files. After conversion, verify the model by running inference tests within the NCNN framework to ensure the conversion was successful. Finally, download the converted `.param` and `.bin` files from Google Colab, which are now ready for deployment in an embedded system or other NCNN-supported environments.

4.4.2 POTHOLE DATASET COLLECTION & ANNOTATION

The YOLO model isn't pre-trained to detect potholes by default. Hence, training the YOLOv11 model with custom dataset is required. The dataset was collected from Kaggle. It consists of 300 images with annotation provided in PASCAL VOC format[15]. However, to train YOLO model, the annotations must be changed to YOLO annotation style. The annotation in YOLO style was performed using Label-Studio, an online annotation tool. The YOLO annotation style includes a box bounding with five parameters: label number, box's center coordinates, its height and width. Since there is only one object to be detected, there is only one label – 0, which indicates potholes. For example, the annotation for the pothole in Fig 4.3 is as follows: 0 0.4805 0.5307 0.6723 0.528. The first '0' indicates the label number, next two are the coordinates of the center of the bounding box and last two are the width and height of the bounding box.



Fig 4.3 Sample image from the dataset

4.5 TRAINING & TESTING THE POTHOLE DETECTION MODEL

The YOLOv11n model was trained using Google Colab[18][19] with Tesla T4 Virtual GPU. The dataset was split into train and validation sets with 270 images (90%) in training set and 30 images (10%) in validation set. The YOLOv11n model was trained with the following parameters:

- No. of epochs = 100
- Image resized to 640x640

The trained model gave an accuracy of 76.6% (mAP50) with average confidence level of 83.2%. The trained model was then validated using the 30 images in validation set to give an accuracy of 78.5% (mAP50) with an average confidence level of 79.1%. The validation resulted in a reduced model size (181 to 100 layers). The model was then tested again with the validation set to give an average processing time of 34ms and was able to detect up to 16 potholes per image.

```
Ultralytics 8.3.111 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
YOLO11n summary (fused): 100 layers, 2,582,347 parameters, 0 gradients, 6.3 GFLOPs

image 1/30 /content/data/validation/images/00428708-potholes49.png: 640x640 1 potholes, 8.9ms
image 2/30 /content/data/validation/images/01ad7453-potholes277.png: 352x640 16 potholes, 70.0ms
image 3/30 /content/data/validation/images/076a0300-potholes105.png: 480x640 6 potholes, 67.1ms
image 4/30 /content/data/validation/images/1fcc247b-potholes260.png: 448x640 2 potholes, 63.5ms
image 5/30 /content/data/validation/images/2882f3a9-potholes181.png: 352x640 (no detections), 11.9ms
image 6/30 /content/data/validation/images/3125e92e-potholes269.png: 640x640 2 potholes, 11.8ms
image 7/30 /content/data/validation/images/3e41f93f-potholes82.png: 448x640 3 potholes, 11.5ms
image 8/30 /content/data/validation/images/483664da-potholes85.png: 640x640 1 potholes, 13.2ms
image 9/30 /content/data/validation/images/48ac160d-potholes18.png: 288x640 1 potholes, 64.6ms
image 10/30 /content/data/validation/images/564c598b-potholes255.png: 384x640 2 potholes, 62.7ms
image 11/30 /content/data/validation/images/5ea6ddd9-potholes7.png: 640x640 1 potholes, 13.4ms
image 12/30 /content/data/validation/images/662932f0-potholes263.png: 480x640 1 potholes, 10.9ms
image 13/30 /content/data/validation/images/7dee3a3a-potholes2.png: 640x640 1 potholes, 11.0ms
image 14/30 /content/data/validation/images/8228f0c7-potholes221.png: 352x640 1 potholes, 10.9ms
image 15/30 /content/data/validation/images/8da76551-potholes197.png: 448x640 1 potholes, 10.9ms
image 16/30 /content/data/validation/images/93e151e4-potholes155.png: 640x640 1 potholes, 11.9ms
image 17/30 /content/data/validation/images/a10f2a74-potholes162.png: 448x640 2 potholes, 11.1ms
image 18/30 /content/data/validation/images/a334ffc5-potholes59.png: 448x640 1 potholes, 12.5ms
image 19/30 /content/data/validation/images/af03ae74-potholes163.png: 640x640 2 potholes, 11.8ms
image 20/30 /content/data/validation/images/af7e7f7c-potholes252.png: 288x640 (no detections), 14.2ms
image 21/30 /content/data/validation/images/b351df3a-potholes159.png: 640x640 2 potholes, 11.5ms
image 22/30 /content/data/validation/images/b78ed56b-potholes179.png: 640x640 1 potholes, 13.9ms
image 23/30 /content/data/validation/images/bff7f8bc-potholes146.png: 448x640 1 potholes, 16.4ms
image 24/30 /content/data/validation/images/da4528e1-potholes284.png: 640x640 1 potholes, 16.6ms
image 25/30 /content/data/validation/images/dd8e1539-potholes205.png: 448x640 3 potholes, 17.7ms
image 26/30 /content/data/validation/images/e139320e-potholes188.png: 384x640 2 potholes, 18.3ms
image 27/30 /content/data/validation/images/ec4d97c7-potholes240.png: 544x640 6 potholes, 73.4ms
image 28/30 /content/data/validation/images/f2814a36-potholes92.png: 640x640 3 potholes, 12.8ms
image 29/30 /content/data/validation/images/f51d6d22-potholes244.png: 480x640 2 potholes, 13.4ms
image 30/30 /content/data/validation/images/fedf8364-potholes123.png: 640x640 3 potholes, 11.2ms
Speed: 3.4ms preprocess, 23.6ms inference, 7.0ms postprocess per image at shape (1, 3, 640, 640)
```

Fig 4.4 Results from testing pothole model using Google Colab

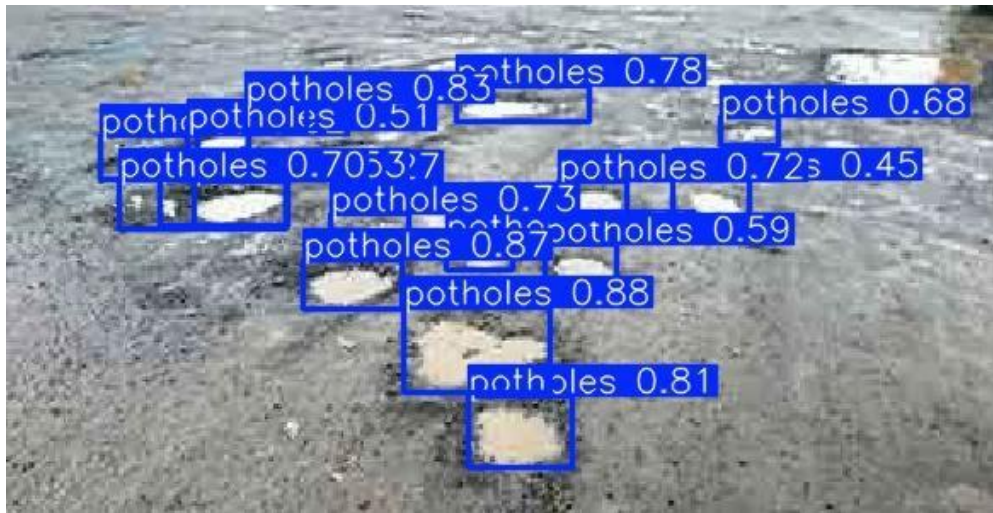


Fig 4.5 Sample pothole detected image from testing the model

4.6 LOGGING OF GPS COORDINATES

The logging of GPS coordinates was done using ipinfo package, Google Client API and Google services. Upon valid detection of a pothole, the ML model inference code determines the location of the rover based on the IPv4 address of the Raspberry Pi. The flow of retrieving the location is shown in Fig 4.6.

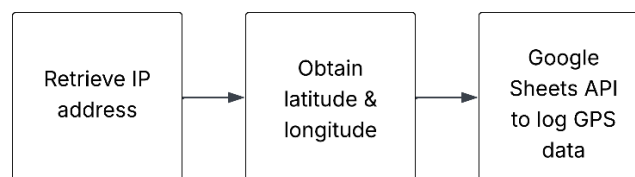


Fig 4.6 Flow diagram of retrieving and logging GPS coordinates

First, the python script retrieves the public IP address of Pi using ipify API. Then it uses the ipinfo[26] service to obtain the latitude and longitude corresponding to the IP address. Then the script uses Google Sheets API[27] to log each detection with the following information recorded: timestamp, latitude and longitude. The google sheets details are as follows:

- Sheet ID: 1xCEepv-e0w2VMd9gBpSLOrFNSeRhaiBPPwqq2qwUc
- Target range: Sheet1!A:C

| A | B | C |
|---------------------|----------|-----------|
| TIMESTAMP | LATITUDE | LONGITUDE |
| 2025-04-09 15:55:43 | 13.0878 | 80.2785 |
| 2025-04-09 15:55:47 | 13.0878 | 80.2785 |
| 2025-04-09 15:55:50 | 13.0878 | 80.2785 |
| 2025-04-09 15:55:54 | 13.0878 | 80.2785 |
| 2025-04-09 15:55:57 | 13.0878 | 80.2785 |
| 2025-04-09 15:56:01 | 13.0878 | 80.2785 |

Fig 4.7 Google sheets – data logging

4.7 RESULTS OBSERVED

The trained YOLOv11n model achieved an accuracy of 78.5% (mAP50) with confidence level of 79.1%. The model has 100 layers, 25,82,347 parameters with an average inference time of 5.3ms as shown in Fig 4.8. From the normalized confusion matrix in Fig 4.9, it can be inferred that true positive (potholes predicted as potholes) is 76% and true negative (potholes not detected as potholes) is 24%. However, false positive (background detected as potholes) is 100% which indicates that any dark background will be detected as an pothole and will result in wrong data logging.

```

Epoch      GPU_mem    box_loss    cls_loss    dfl_loss    Instances    Size
100/100      4.09G      0.8286      0.5372      1.015       25           640: 100% 17/17 [00:04<00:00, 3.48it/s]
Class      Images    Instances    Box(P      R      mAP50    mAP50-95): 100% 1/1 [00:00<00:00, 2.41it/s]
all        30        59          0.832      0.661      0.766      0.442

100 epochs completed in 0.194 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 5.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 5.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.111 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
YOLO11n summary (fused): 100 layers, 2,582,347 parameters, 0 gradients, 6.3 GFLOPs
Class      Images    Instances    Box(P      R      mAP50    mAP50-95): 100% 1/1 [00:00<00:00, 2.54it/s]
all        30        59          0.791      0.712      0.785      0.489
Speed: 0.3ms preprocess, 3.5ms inference, 0.0ms loss, 1.6ms postprocess per image

```

Fig 4.8 Training and validation results

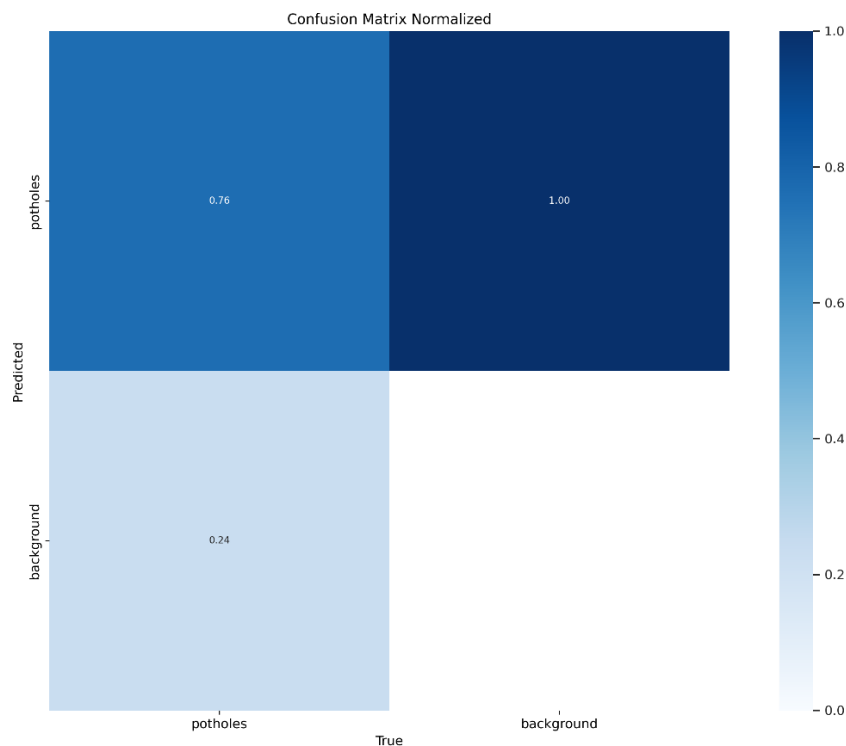
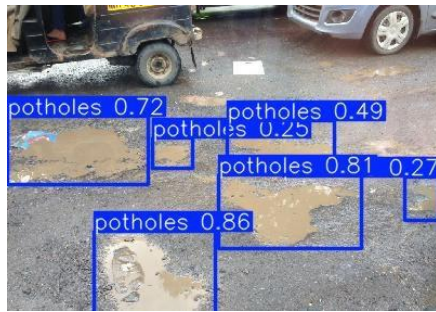


Fig 4.9 Normalized Confusion matrix of the trained ML model

The NCNN-YOLOv1nn pothole detection model achieved an average fps of 6 and an inference latency of ~150 to 200ms per frame. From Fig 4.10, it can be inferred that, in spite of false positive detection being 100% on training, on actual testing using Raspberry Pi 5, dark backgrounds in Fig 4.10 (d) weren't detected as pothole. Also, the model proved well for small sized potholes (Fig 4.10 (b)) and large no. of potholes per frame (Fig 4.10 (a)).



(a)



(b)



(c)



(d)

Fig 4.10 Testing of NCNN-YOLOv11n model on Raspberry Pi 5

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The project 'Road Anomaly Detection Bot' successfully detects potholes of various size and shape with high accuracy and confidence level using the custom trained YOLO model. By exporting the YOLOv11n model to NCNN framework, the pothole detection system functioned effectively with an average fps of 6 with a confidence level of 60 to 70%. Through the use of Raspberry Pi 5 and NCNN framework, the system displayed a better trade-off between performance and cost, making it suitable for real world implementation at lost cost. Overall, the project demonstrated a semi-autonomous pothole detection and GPS coordinate logging system, creating the ability to detect potholes without human intervention. It also opens up pathways for future improvements to making the system more accurate and fully autonomous.

The scope of this project involves upgrading the current semi-autonomous system into a fully autonomous drone capable of independently navigating roadways and detecting road anomalies. The drone will be equipped with a custom, lightweight deep learning model optimized for edge devices, specifically designed to detect potholes, cracks, and relevant road signs. This model will be optimized to minimize inference latency, enabling real-time detection of anomalies and the identification of pothole-free routes, which will enhance the travel experience. The integration of a GPS module and a distance estimation algorithm will provide precise localization of detected anomalies, improving the accuracy and reliability of the system.

In addition to anomaly detection, a feedback mechanism will be incorporated to verify whether detected potholes have been covered or repaired. The drone will monitor the same locations over time, providing real-time feedback to update its database on the status of road repairs. This continuous monitoring will ensure that the system accurately tracks the resolution of road anomalies, offering valuable data to road authorities. This enhancement will improve the efficiency of road maintenance, offering a reliable, autonomous solution for detecting and reporting potholes and other road issues.

BIBLIOGRAPHY

1. TNN, "Potholes caused 13 deaths in 5 years in Bengaluru; Rs 215 crore spent on repair," *The Times of India*, Feb. 28, 2022. <https://timesofindia.indiatimes.com/city/bengaluru/potholes-caused-13-deaths-in-5-years-in-bengaluru-rs-215-crore-spent-on-repair/articleshow/89881027.cms>
2. I. Sharma, "Over 5,000 People Killed In Road Accidents Caused By Potholes In 2018-20: Transport Ministry," *Indiatimes*, Aug. 22, 2022. <https://www.indiatimes.com/trending/social-relevance/over-5000-people-died-by-potholes-accidents-between-2018-2020-577821.html>
3. V. Mourya, S. Shaikh, S. Sayyed, S. Shivasharan, and N. Costa, "Smart Road Safety System: detecting potholes, sharp turns, speed bumps, and signboards for driver safety," *International Journal of Computer Applications*, pp. 12–13, Jul. 2024, [Online]. Available: <https://www.ijcaonline.org/archives/volume186/number34/mourya-2024-ijca-923884.pdf>
4. D. J, S. D. V, A. S A, K. R and L. Parameswaran, "Deep Learning based Detection of potholes in Indian roads using YOLO," 2020 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 2020, pp. 381-385, doi: 10.1109/ICICT48043.2020.9112424. keywords: {Deep learning;Shape;Roads;Object detection;Maintenance engineering;Feature extraction;Market research;ADAS;CNN;Deep Learning;Potholes Detection;R-CNN;Vision based approach;YOLOv2;YOLOv3},
5. Asad, Muhammad Haroon, Khaliq, Saran, Yousaf, Muhammad Haroon, Ullah, Muhammad Obaid, Ahmad, Afaq, Pothole Detection Using Deep Learning: A Real-Time and AI-on-the-edge Perspective, *Advances in Civil Engineering*, 2022, 9221211, 13 pages, 2022. <https://doi.org/10.1155/2022/9221211>
6. Park, S.-S.; Tran, V.-T.; Lee, D.-E. Application of Various YOLO Models for Computer Vision-Based Real-Time Pothole Detection. *Appl. Sci.* 2021, 11, 11229. <https://doi.org/10.3390/app112311229>
7. Sah, S., "MCUBench: A Benchmark of Tiny Object Detectors on MCUs", <i>arXiv e-prints</i>, Art. no. arXiv:2409.18866, 2024. doi:10.48550/arXiv.2409.18866.
8. Y. Zhang and C. Liu, "Real-Time Pavement Damage Detection With Damage Shape Adaptation," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 11, pp. 18954-18963, Nov. 2024, doi: 10.1109/TITS.2024.3416508. keywords:{Accuracy;YOLO;Transformers;Sensors;Roads;Three-dimensional displays;Detection algorithms;Non-destructive testing;transformer;damage detection;real-time detection},
9. Diwan, T., Anirudh, G. & Tembhurne, J.V. Object detection using YOLO: challenges, architectural successors, datasets and applications. *Multimed Tools Appl* 82, 9243–9275 (2023). <https://doi.org/10.1007/s11042-022-13644-y>
10. T. V. Sai, B. Aditya, A. M. Reddy, and Y. Srinivasulu, "Real time object detection using Raspberry Pi," *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 1, pp. 834–838, Jan. 2023, doi: 10.22214/ijraset.2023.48549
11. M. M. Garcillanosa, J. M. L. Pacheco, R. E. Reyes and J. J. P. San Juan, "Smart Detection and Reporting of Potholes via Image-Processing using Raspberry-Pi Microcontroller," 2018 10th International Conference on Knowledge and Smart Technology (KST), Chiang Mai, Thailand, 2018, pp. 191-195, doi: 10.1109/KST.2018.8426203. keywords: {Roads;Microcomputers;Automobiles;Web servers;Cameras;Global Positioning System;Testing;Cloud storage;global positioning system;image-processing;internet;microcomputer;smart devices;webserver}, Conference on Knowledge and Smart Technology (KST), Chiang Mai, 2018, pp. 191-195.

12. V. Pereira, S. Tamura, S. Hayamizu and H. Fukai, "A Deep Learning-Based Approach for Road Pothole Detection in Timor Leste," 2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), Singapore, 2018, pp. 279-284, doi: 10.1109/SOLI.2018.8476795. keywords: {Roads;Machine learning;Feature extraction;Training;Support vector machines;Convolutional neural networks;Task analysis;Potholes;Deep Learning;Convolutional Neural Network;Image Classification},
13. B. Prakash and S. Nayak, "A REVIEW ON ROAD POTHOLE DETECTION MECHANISMS," 1361. Accessed: Apr. 21, 2025. [Online]. Available: https://www.irjmets.com/uploadedfiles/paper/volume2/issue_9_september_2020/3899/1628083154.pdf
14. S. C. Shekar, S. Naqhi, S. R., and S. P., "Real Time Detection of Potholes using Image Processing," *International Journal of Advances in Engineering and Management (IJAEM)*, vol. 4, no. 6, pp. 369–375, Jun. 2022. [Online]. Available: https://ijaem.net/issue_dcp/Real%20Time%20Detection%20of%20Potholes%20using%20Image%20Processing.pdf
15. Vijayalakshmi B, Kiran P, Kishor Jadav B, Madhusudhan GR and Manoj KS, "Detection of potholes using machine learning and image processing," *Proceedings of the National Conference on Advancements in Information Technology (NCAIT)*, vol. 8, no. 15, Special Issue, 2020. [Online]. Available: <https://www.ijert.org/research/detection-of-potholes-using-machine-learning-and-image-processing-IJERTCONV8IS15039.pdf>
16. K. Singh, S. Hazra, C. Mukherjee, S. G., and S. Gowda, "IoT-Based Real-Time Potholes Detection System Using Image Processing Techniques," *International Journal of Scientific & Technology Research*, vol. 9, no. 2, pp. 785–789, Feb. 2020. [Online]. Available: <https://ijstr.org/final-print/feb2020/Iot-Based-Real-Time-Potholes-Detection-System-Using-Image-Processing-Techniques.pdf>
17. S. N. Rao, "YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy," *Medium*, Oct. 22, 2024. <https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71>
18. *Raspberrypi.org*, 2025. <https://projects.raspberrypi.org/en/pathways/raspberry-pi-beginners>
19. M. Lab, "Interface L298N DC Motor Driver Module with ESP8266 NodeMCU," *Microcontrollers Lab*, Jan. 09, 2022. <https://microcontrollerslab.com/l298n-dc-motor-driver-module-esp8266-nodemcu/>
20. "Pascal VOC XML Annotation Format," *Roboflow.com*, 2021. <https://roboflow.com/formats/pascal-voc-xml>
21. Team, "A Basic Guide for Getting Started with Raspberry Pi 5 Programming Languages - SmashingApps.com," *SmashingApps.com*, Apr. 18, 2024. <https://www.smashingapps.com/guide-raspberry-pi-5-programming-languages/>
22. R. Khandelwal, "COCO and Pascal VOC data format for Object detection," *Medium*, Dec. 07, 2019. <https://medium.com/data-science/coco-data-format-for-object-detection-a4c5eaf518c5>
23. Evan Juras, "Train YOLO Models", *Google Colab*, Jan. 03, 2025. https://colab.research.google.com/github/EdjeElectronics/Train-and-Deploy-YOLO-Models/blob/main/Train_YOLO_Models.ipynb#:~:text=This%20notebook%20uses%20Ultralytics%20to%20train%20YOLO11%2C%20YOLOv8%2C,Pi.%20Custom%20YOLO%20candy%20detection%20model%20in%20action%21
24. Edje Electronics, Bozeman, United States. *How to Train YOLO Object Detection Models in Google Colab (YOLO11, YOLOv8, YOLOv5)*. (Aug. 1, 2025). [Online Video]. Available: <https://www.youtube.com/watch?v=r0RspILG260&t=547s>
25. Edje Electronics, Bozeman, United States. *How to Run YOLO Object Detection Models on the Raspberry Pi*. (May 2, 2025). [Online Video]. Available: <https://www.youtube.com/watch?v=z70ZrSZNi-8>
26. "DNA Special: The pothole problem of Indian cities and its fatal side effects," *DNA India*, 2022. <https://www.dnaindia.com/analysis/report-dna-special-the-pothole-problem-of-indian-cities-and-its-fatal-side-effects-2966530>
27. Ni, H., & The ncnn contributors. (2017). ncnn [Computer software]. <https://github.com/Tencent/ncnn>

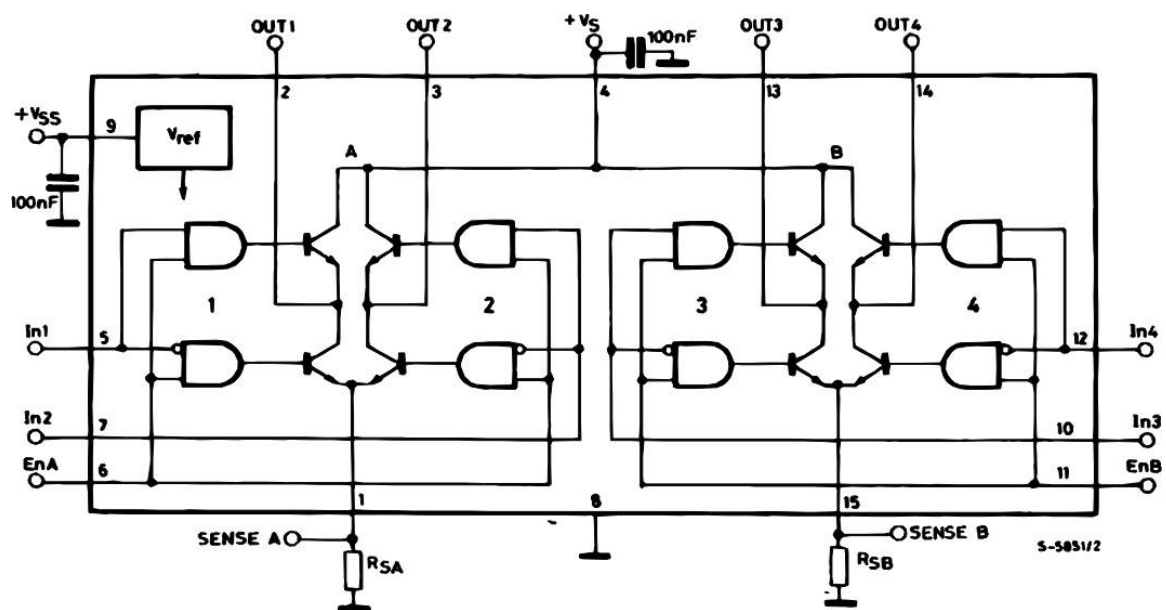
28. R. Khanam and M. Hussain, "YOLOV11: AN OVERVIEW OF THE KEY ARCHITECTURAL ENHANCEMENTS," 2024. Available: <https://arxiv.org/pdf/2410.17725>
29. Larxel, "Pothole Detection," Kaggle.com, 2020. <https://www.kaggle.com/datasets/andrewmvd/pothole-detection/data>
30. "How to Get Geolocation in Python?," *Ipinfo.io*, 2025. <https://ipinfo.io/blog/geolocation-api-python>
31. SB Social, "Log Raspberry Pi data to Google Sheets," *SB Components Ltd*, Jul. 25, 2019. <https://shop.sb-components.co.uk/blogs/posts/log-raspberry-pi-data-to-google-sheets>

APPENDIX A

DATASHEET

L298N MOTOR DRIVER

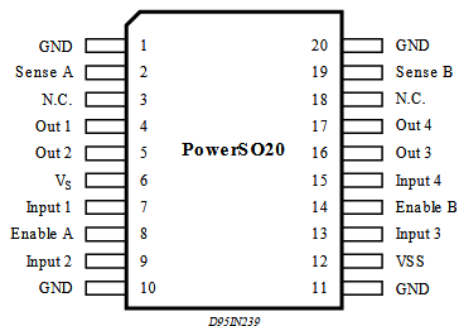
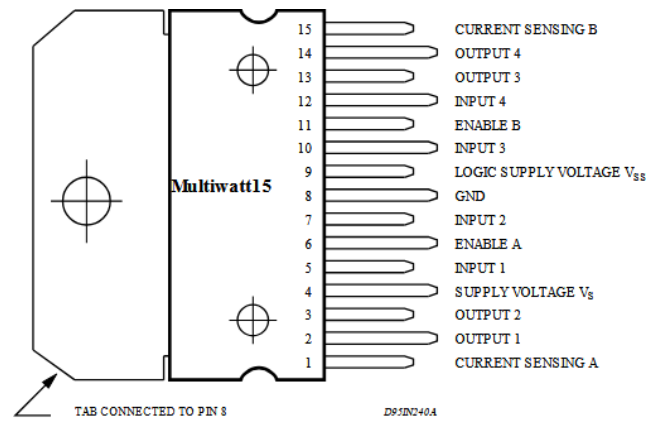
Block Diagram



Electrical Characteristics

| Symbol | Parameter | Value | Unit |
|----------------|---|------------|--------------------|
| V_S | Power supply | 50 | V |
| V_{SS} | Logic supply voltage | 7 | V |
| V_I, V_{en} | Input and enable voltage | -0.3 to 7 | V |
| I_O | Peak output current (each channel): | | |
| | • Non repetitive ($t = 100\text{ ms}$) | 3 | A |
| | • repetitive (80% on -20% off; $t_{on} = 10\text{ ms}$) | 2.5 | A |
| | • DC operation | 2 | A |
| V_{sens} | Sensing voltage | -1 to 2.3 | V |
| P_{tot} | Total power dissipation ($t_{case} = 75\text{ }^{\circ}\text{C}$) | 25 | W |
| T_{op} | Junction operating temperature | -25 to 130 | $^{\circ}\text{C}$ |
| T_{stg}, T_j | Storage and junction temperature | -40 to 150 | $^{\circ}\text{C}$ |

Pin configuration

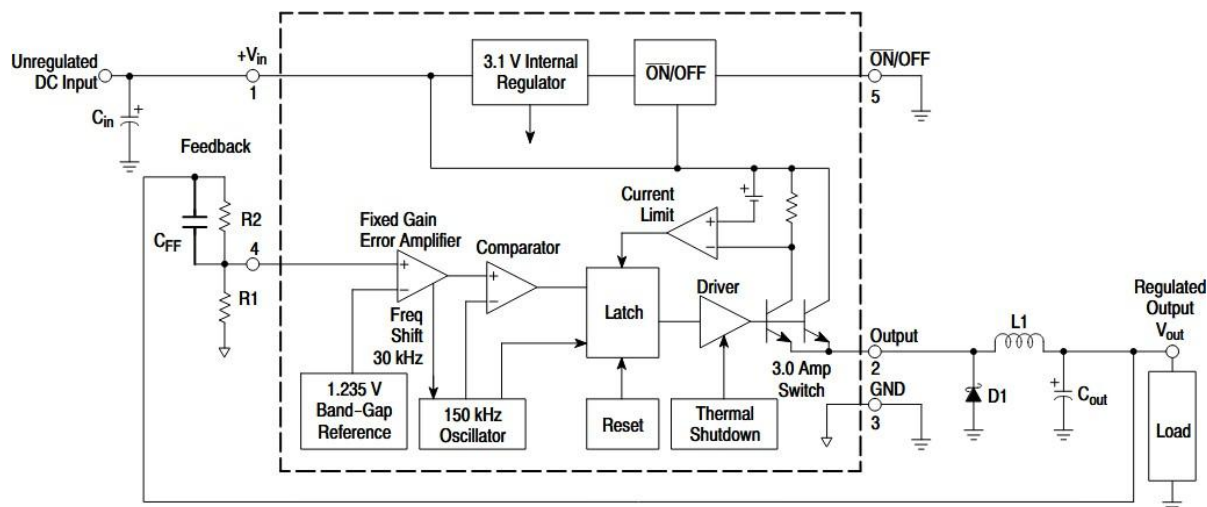


Pin function

| MW.15 | Power SO | Name | Function |
|--------|---------------|--------------------|---|
| 1, 15 | 2, 19 | Sense A, Sense B | Between this pin and ground is connected the sense resistor to control the current of the load. |
| 2, 3 | 4, 5 | Out 1, Out 2 | Outputs of the bridge A; the current that flows through the load connected between these two pins is monitored at pin 1. |
| 4 | 6 | V _S | Supply voltage for the power output stages. A non-inductive 100nF capacitor must be connected between this pin and ground. |
| 5, 7 | 7, 9 | Input 1, Input 2 | TTL compatible inputs of the bridge A. |
| 6, 11 | 8, 14 | Enable A, Enable B | TTL compatible enable input: the L state disables the bridge A (enable A) and/or the bridge B (enable B). |
| 8 | 1, 10, 11, 20 | GND | Ground. |
| 9 | 12 | VSS | Supply voltage for the logic blocks. A 100nF capacitor must be connected between this pin and ground. |
| 10, 12 | 13, 15 | Input 3, Input 4 | TTL compatible inputs of the bridge B. |
| 13, 14 | 16, 17 | Out 3, Out 4 | Outputs of the bridge B. The current that flows through the load connected between these two pins is monitored at pin 15. |
| – | 3, 18 | N.C. | Not connected |

VOLTAGE REGULATOR (LM2596S)

Block diagram



Electrical characteristics





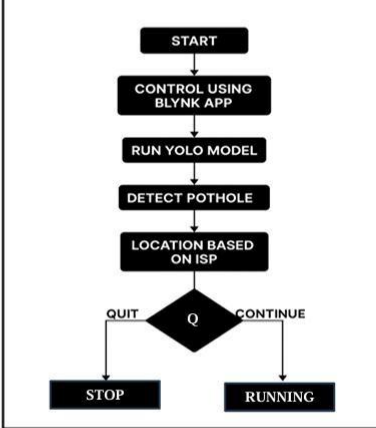
| Rating | Symbol | Value | Unit |
|--|-----------------|-------------------------------------|----------------------|
| Maximum Supply Voltage | V_{in} | 45 | V |
| ON/OFF Pin Input Voltage | – | $-0.3\text{ V} \leq V \leq +V_{in}$ | V |
| Output Voltage to Ground (Steady-State) | – | -1.0 | V |
| Power Dissipation | | | |
| Case 314B and 314D (TO-220, 5-Lead) | P_D | Internally Limited | W |
| Thermal Resistance, Junction-to-Ambient | $R_{\theta JA}$ | 65 | $^{\circ}\text{C/W}$ |
| Thermal Resistance, Junction-to-Case | $R_{\theta JC}$ | 5.0 | $^{\circ}\text{C/W}$ |
| Case 936A (D ² PAK) | P_D | Internally Limited | W |
| Thermal Resistance, Junction-to-Ambient | $R_{\theta JA}$ | 70 | $^{\circ}\text{C/W}$ |
| Thermal Resistance, Junction-to-Case | $R_{\theta JC}$ | 5.0 | $^{\circ}\text{C/W}$ |
| Storage Temperature Range | T_{stg} | -65 to +150 | $^{\circ}\text{C}$ |
| Minimum ESD Rating (Human Body Model: C = 100 pF, R = 1.5 k Ω) | – | 2.0 | kV |
| Lead Temperature (Soldering, 10 seconds) | – | 260 | $^{\circ}\text{C}$ |
| Maximum Junction Temperature | T_J | 150 | $^{\circ}\text{C}$ |

Pin function

| Pin | Symbol | Description (Refer to Figure 1) |
|-----|----------|--|
| 1 | V_{in} | This pin is the positive input supply for the LM2596 step-down switching regulator. In order to minimize voltage transients and to supply the switching currents needed by the regulator, a suitable input bypass capacitor must be present (C_{in} in Figure 1). |
| 2 | Output | This is the emitter of the internal switch. The saturation voltage V_{sat} of this output switch is typically 1.5 V. It should be kept in mind that the PCB area connected to this pin should be kept to a minimum in order to minimize coupling to sensitive circuitry. |
| 3 | GND | Circuit ground pin. See the information about the printed circuit board layout. |
| 4 | Feedback | This pin is the direct input of the error amplifier and the resistor network R2, R1 is connected externally to allow programming of the output voltage. |
| 5 | ON/OFF | It allows the switching regulator circuit to be shut down using logic level signals, thus dropping the total input supply current to approximately 80 μA . The threshold voltage is typically 1.6 V. Applying a voltage above this value (up to $+V_{in}$) shuts the regulator off. If the voltage applied to this pin is lower than 1.6 V or if this pin is left open, the regulator will be in the "on" condition. |

APPENDIX B

POSTER

| | | | |
|--|---|--|---|
|  | 19L620 Innovation Practices - Project Expo | Date:11/04/25 Venue: Devices Lab |  |
| <h3 style="margin: 0;">ROAD ANOMALY DETECTION BOT</h3> | | | |
| <div style="display: flex;">   </div> <ul style="list-style-type: none"> 🔍 Automatically detect road anomalies like cracks and potholes using real-time video. 📍 Log GPS coordinates of detected damages for precise location-based reporting. 📱 Control mobility via NodeMCU-operated rover using the Blynk mobile app. 🧠 Deploy a lightweight YOLOv11n model, exported to NCNN for efficient edge inference on Raspberry Pi. 🌐 Offer a low-cost, scalable, and cloud-free solution for urban and rural road monitoring. | |  <pre> graph TD START([START]) --> CONTROL[CONTROL USING BLYNK APP] CONTROL --> RUN[RUN YOLO MODEL] RUN --> DETECT[DETECT POTHOLE] DETECT --> LOCATION[LOCATION BASED ON ISP] LOCATION --> Q{Q} Q -- QUIT --> STOP([STOP]) Q -- CONTINUE --> RUNNING([RUNNING]) </pre> | |
| ANIRUDH RAMKUMAR(22L207) | | DHEERAJ RAJESH(22L214) NAHUL AARIYA D K(22L238) | |
| LAKSHMANAN A R(22L231) | | | |