YOLOv11–NCNN Autonomous Driving System

AHMED ABD ELMONEIM MOHAMMED ABD ELGAYOUM

B. ENG(HONS.) MECHATRONICS

ENGINEERING

UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	:	AHMED ABDELMONEM MOHAMED ABDELGAYOUM
Date of Birth	:	26/09/2003
Title	:	YOLOv11–NCNN Autonomous Driving System
Academic Session	:	SEMESTER 1 2025/2026

I declare that this thesis is classified as:

☐	CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997) *
☐	RESTRICTED	(Contains restricted information as specified by the organization where research was done) *
■	OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang Al-Sultan Abdullah reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang Al-Sultan Abdullah.

2. The Library of Universiti Malaysia Pahang Al-Sultan Abdullah has the right to make copies of the thesis for the purpose of research only.

3. The library has the right to make copies of the thesis for academic exchange.

Certified by:

(Student's Signature)

P08756107
New IC/Passport Number
Date:3/1/2025

(Supervisor's Signature)

Name of Supervisor
Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Perpustakaan Universiti Malaysia Pahang Al-Sultan Abdullah,
Universiti Malaysia Pahang Al-Sultan Abdullah,
Lebuhraya Tun Razak, 26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name
Thesis Title

Reasons (i)

(ii)

(iii)

Thank you.

Yours faithfully,

_____
(Supervisor's Signature)

Date: 3/1/2025

Stamp:

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.

SUPERVISOR'S DECLARATION

I/We* hereby declare that I/We* have checked this thesis/project* and in my/our* opinion, this thesis/project* is adequate in terms of scope and quality for the award of the degree of *Doctor of Philosophy/ Master of Engineering/ Master of Science in

……..

_____

(Supervisor's Signature) Full

Name :

Position :

Date :

_____

(Co-supervisor's Signature) Full

Name :

Position :

Date :

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

_____

(Student's Signature)

Full Name : AHMED ABDELMONEM MOHAMED ABDALGAYOUM

ID Number : FB2005

Date : 3 January 2026

YOLOv11–NCNN Autonomous Driving System

AHMED ABD ELMONEIM MOHAMMED ABD ELGAYOUM

Thesis submitted in fulfilment of the requirements
for the award of the degree of
Bachelor of Engineering

Faculty of Manufacturing & Mechatronics Engineering Technology
UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

JANUARY 2026

ACKNOWLEDGEMENTS

First and foremost, I am truly grateful to Allah S.W.T. for granting me the strength, patience, and perseverance to complete this Final Year Project successfully. I would like to express my heartfelt appreciation to my supervisor, Dr. Khairul Fikri bin Muhammad, for his continuous support, insightful guidance, and motivation throughout this project. He was not only a dedicated lecturer and mentor but also a sincere friend who believed in me and stood by me at every stage of this journey.

My deepest thanks go to my academic advisor, Dr. Nafrizuan bin Mat Yahya, who consistently guided me throughout my academic path. His advice and encouragement helped me overcome many challenges, both academically and personally. Additionally, I want to sincerely thank the instructors who were crucial in my academic and technical development. My learning has been profoundly impacted by the willingness of Dr. Ahmad Najmuddin bin Ibrahim, Dr. Ahmad Syakirin bin Ismail @ Rosdi, and Assoc. Prof. Dr. Izwan bin Ismail to take time out of your hectic schedules to listen to my questions, talk about technical ideas, and share your knowledge of robotics, electronics, and electrical systems. Your generosity and commitment have really influenced who I am becoming as an engineer. To my beloved parents, thank you for your unconditional love, constant prayers, and unwavering support. Your belief in me, even when I doubted myself, gave me the strength to push through every obstacle.

Lastly, I would like to thank my friends who stood by me during this journey. Your support, late-night brainstorming sessions, and encouraging words kept me motivated and made this experience more rewarding. To everyone who has contributed to my academic and personal growth, whether directly or indirectly — thank you from the bottom of my heart.

ABSTRACT

The rapid advancement of deep learning and embedded artificial intelligence has catalysed significant progress in the development of autonomous navigation systems. However, deploying high-performance object detection models on resource-constrained platforms remains a substantial challenge, particularly for real-time applications requiring low latency and high reliability. This project presents the design and implementation of a **YOLOv11–NCNN Autonomous Driving System**, an embedded vision-based navigation platform that integrates the latest YOLOv11 object-detection architecture with the **Neural Network Inference Framework (NCNN)** to achieve efficient, real-time perception on low-power hardware.

The system captures continuous visual input from an onboard camera, processes it through a YOLOv11 model optimised using NCNN, and generates semantic detections for autonomous decision-making. The research addresses key limitations in traditional autonomous mobile robots, including reliance on basic proximity sensors, limited contextual awareness, and dependence on computationally expensive inference methods. By converting and deploying YOLOv11 models through NCNN, the platform achieves substantial improvements in inference speed, enabling the vehicle to detect obstacles, identify navigational cues, and adjust its trajectory in real time without external computational resources.

To enable autonomous forward motion, obstacle avoidance, and safety-conscious halting behaviour, a comprehensive navigation controller was developed to interpret perceptual outputs and convert them into motor-control commands. The system was tested in controlled environments to evaluate navigation reliability, inference latency, and detection accuracy. The results demonstrate that the NCNN-optimised YOLOv11 pipeline provides a viable solution for embedded autonomous systems, offering strong detection performance alongside notable computational efficiency.

By providing a practical framework for combining advanced neural-network models with lightweight inference for real-time autonomous navigation, this study contributes to the growing field of embedded deep learning and demonstrates the feasibility of deploying cutting-edge perception models on resource-limited hardware.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **VII**

ABSTRACT **VIII**

LIST OF ABBREVIATIONS **XVI**

CHAPTER 1 **1**

1.1 INTRODUCTION 1

1.1.1 General Description of the Study 1

1.1.2 Brief Mention of Past Studies 2

1.2 RESEARCH BACKGROUND 3

1.3 PROBLEM STATEMENT 4

1.4 OBJECTIVES 5

1.5 SCOPE AND DELIMITATIONS 5

1.5.1 Scope 5

1.5.2 Delimitations 6

1.6 THESIS OUTLINE 6

CHAPTER 2 **8**

2.1 INTRODUCTION 8

2.2 FOUNDATIONS & ADVANCES IN YOLO MODELS FOR TRAFFIC-SIGN AND TRAFFIC-LIGHT PERCEPTION 9

2.2.1 Lightweight YOLO-Based Detection Algorithms for Real-Time Systems **9**

2.2.2 Small-Object Detection in Traffic Environments **10**

2.2.3 MXT-YOLOv7t for Mixed-Traffic Autonomous Driving **10**

2.2.4 Lightweight Traffic-Sign Detection Using Enhanced YOLOv7 Architectures **12**

2.2.5 Real-Time Traffic-Sign Detection with YOLOv7 Variants **12**

2.2.6 Review of YOLO Algorithm Developments in Autonomous Driving **13**

2.2.7 Advanced YOLOv8-Based Techniques for Traffic-Sign Recognition **14**

2.3 EMBEDDED SYSTEMS AND MODEL OPTIMIZATION 14

2.3.1 Limitations of YOLOv3-Based Autonomous Robot Systems 15

2.3.2 Performance Constraints in MobileNetV2-Based Traffic Sign Recognition 15

2.3.3 Efficiency Challenges in Deep Learning-Based Traffic Sign Detection 16

2.3.4 GPU Dependency in YOLOv4-Based Robotic Vehicles 17

2.4 CRITICAL EVALUATIONS OF TRAFFIC LIGHT DETECTION AND EMBEDDED YOLO-BASED PERCEPTION 17

2.4.1 Analytical Evaluation of YOLOv4-Based Traffic Light Detection Frameworks 17

2.4.2 Technical Critique of Quantised YOLO Architectures for Embedded Real-Time Inference 18

2.4.3 Performance Assessment of CNN/YOLO Deployment on Raspberry Pi for Real-Time Detection 19

2.5 IDENTIFIED RESEARCH GAPS IN TRAFFIC PERCEPTION AND EMBEDDED OBJECT DETECTION 19

2.5.1 Gap Analysis of YOLO-Based Small Traffic Light Detection Models 20

2.6 EVALUATION OF YOLO ARCHITECTURES FOR ADAS AND AUTONOMOUS DRIVING SYSTEMS 20

2.7 PERFORMANCE ASSESSMENT OF TWO-STAGE TRAFFIC SIGN CLASSIFICATION FRAMEWORKS 21

2.8 TECHNICAL EVALUATION OF 5G IOT TRAFFIC CHARACTERISATION AND CLUSTERING MODELS 21

2.9 ANALYSIS OF MULTI-DOMAIN FEATURE FUSION FOR SDN–IOT TRAFFIC PREDICTION 22

2.10 EVALUATION OF IOT-ASSISTED AUTONOMOUS ROBOTIC TRAFFIC MONITORING SYSTEMS 23

CHAPTER 3 **27**

3.1 INTRODUCTION 27

3.2 RESEARCH BACKGROUND 28

3.2.1 System Flow Chart 28

3.3 DATASET PREPARATION 31

3.3.1 Roboflow Platform 31

3.3.2 Annotation Process 32

3.4 SYSTEM OVERVIEW 33

3.5 HARDWARE CONFIGURATION 35

3.5.1 Raspberry Pi 5 (8GB RAM) 35

3.5.2 Robot HAT 35

3.5.3 Grayscale Sensor 36

3.5.4 Pi Camera (P5V04A) 37

3.6 SOFTWARE FRAMEWORK 37

3.7 YOLO MODEL 37

3.7.1 YOLO Models Overview 38

3.7.2 Model Training 38

3.7.3 Setup and Initialization 39

3.7.4 Model Weight Initialization and Dataset Import 39

3.7.5 Training Command and Process 40

3.7.6 Post-Training Optimization 40

3.7.7 Classification Loss 40

3.7.8 Localization Loss 41

3.7.9 Confidence Loss 41

3.7.10 Parameter Specification 44

3.8 MODEL EVALUATION 44

3.9 DETECTION IMPLEMENTATION 44

3.10 INTEGRATION WITH GRAYSCALE SENSOR FOR LANE FOLLOWING 45

3.11 SUMMARY OF METHODOLOGY 45

CHAPTER 4 **46**

4.1 INTRODUCTION 46

4.2 TRAINING OVERVIEW AND MODEL SUMMARY 46

4.3 QUANTITATIVE PERFORMANCE METRICS 47

4.4 TENSORBOARD GRAPHS AND TRAINING BEHAVIOR ANALYS 48

4.4.1 Metric Curves Analysis 48

4.4.2 Loss Function Analysis 48

4.5 DATASET DISTRIBUTION AND ANNOTATION ANALYSIS 50

4.6 VISUAL DETECTION RESULTS 51

4.7 NCNN-BASED VISUAL DETECTION RESULTS 54

4.8 REAL-TIME PERFORMANCE ON RASPBERRY PI 5 56

4.9 DISCUSSION 57

CHAPTER 5 **58**

5.1 INTRODUCTION 58

5.2 OVERALL PROJECT SUMMARY 58

5.3 EVALUATION OF SYSTEM PERFORMANCE 59

5.4 CHALLENGES AND LIMITATIONS 59

5.4.1 Accuracy Degradation and Numerical Precision Effects 59

5.4.2 Computational Constraints of Embedded Hardware 60

5.4.3 Dataset and Environmental Limitations 60

5.5 CONTRIBUTIONS AND ACHIEVEMENTS 61

5.6 FUTURE WORK AND RECOMMENDATIONS 61

5.7 FINAL REMARKS 62

REFERENCES **63**

APPENDIX A **66**

APPENDIX B **68**

Figure No.	Title	Page No
Figure1.1	The general process for object detection	2
Figure1.2	YOLOv11–NCNN system pipeline	4
Figure2.1	Detection results of YOLOv3 and YOLO-MXANet on the CCTSDB dataset	10
Figure2.2	Comparative performance of MXT-YOLOv7t and YOLOv7-tiny in mixed-traffic scenarios	11
Figure2.3	Architectural enhancements introduced in YOLOv7	12
Figure2.4	Overall structure of the YOLO detection pipeline	13
Figure2.5	Data processing and augmentation pipeline	14
Figure3.1	Overall system flow of the YOLOv11–NCNN	30
Figure3.2	Roboflow annotation platform used for dataset labelling	32
Figure 3.3	Integrated YOLO-Based System Flowchart for Autonomous Robot Navigation	34
Figure 3.4	Raspberry Pi 5 Single-Board Computer	35
Figure3.5	Robot HAT motor and power control board	36
Figure3.6	Grayscale sensor module for lane detection	36
Figure3.7	Raspberry Pi Camera Module (P5V04A)	37
Figure 3.8	Google Colab Platform for Deep Learning Model Training	39
Figure3.9	F1-score curve during training	42
Figure3.10	Precision curve during training	42
Figure3.11	Recall curve during training	43
Figure3.12	Precision–Recall (PR) curve during training	43
Figure 4.1	Training overview and model summary	47

Figure 4.2	Training loss curves (box, classification, objectness)	49
Figure 4.3	Validation loss curves (box, classification, objectness)	49
Figure 4.4	Dataset Class Distribution	50
Figure 4.5	Bounding Box x–y Position Distribution	51
Figure 4.6	Bounding Box Width–Height Distribution	51
Figure 4.7	Traffic signs	52
Figure 4.8	No entry sign	53
Figure 4.9	U turn sign	53
Figure 4.10	Pedestrian crossing sign	53
Figure 4.11	Turn left sign	54
Figure 4.12	NCNN pedestrian crossing detection	54
Figure 4.13	NCNN no-entry sign detection	55
Figure 4.14	NCNN turn-left sign detection	55
Figure 4.15	NCNN traffic sign misclassification	56

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ADAS	Advanced Driver Assistance Systems
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
FPS	Frames Per Second
IoT	Internet of Things
mAP	Mean Average Precision
NCNN	Neural Network Inference Framework
Pi	Raspberry Pi
SLAM	Simultaneous Localization and Mapping
SSD	Single Shot Multibox Detector
YOLO	You Only Look Once

CHAPTER 1

INTRODUCTION

1.1 Introduction

As a central element of the Fourth Industrial Revolution, autonomous driving technologies are driving transformative advancements in robotics, embedded **Artificial Intelligence (AI)** systems, logistics, and transportation. The capability to design compact systems that can perceive and respond to their environment autonomously has expanded considerably due to progress in computer vision, deep learning, and low-power embedded computing technologies. Contemporary autonomous systems increasingly rely on data-driven approaches capable of interpreting complex visual information in real time, rather than solely on preprogrammed behaviours or conventional sensing techniques.

In this regard, the current study focuses on creating a YOLOv11–NCNN Autonomous Driving System to show that complex object-detection algorithms may be successfully implemented on embedded systems with low resources. The system uses YOLOv11, the latest version of the You Only Look Once object-detection architecture, along with the Neural Network Inference Framework (NCNN), a high-performance inference engine optimized for ARM-based and mobile CPUs. High-accuracy, low-latency perception appropriate for real-time autonomous navigation on embedded technology is made possible by this combination.

Small-scale embedded systems provide an essential platform for experimental development, algorithmic evaluation, and proof-of-concept validation. While full-scale autonomous vehicles require extensive infrastructure and multi-sensor fusion, this study contributes to the broader progression of vehicle autonomy by enabling object detection, hazard recognition, and navigation decision-making on an embedded module while maintaining affordability and practical feasibility.

1.1.1 General Description of the Study

The suggested autonomous driving system examines live video input, uses YOLOv11 to identify pertinent objects, uses NCNN to speed up inference, and converts detections into motor-control choices. The system shows that advanced AI models may be effectively implemented in local embedded contexts as it doesn't rely on cloud computing, external GPU

resources, or bulky computational gear.

The major goal of this work is to demonstrate how intelligent perception may be accomplished with little resources while preserving the fundamental characteristics of an autonomous-navigation pipeline: computing efficiency, environmental awareness, responsiveness, and dependability.

1.1.2 Brief Mention of Past Studies

Earlier works in autonomous robotics predominantly relied on:

- ultrasonic sensors for obstacle avoidance,
- infrared sensors for boundary detection,
- pre-programmed traversal patterns, or
- cloud-based AI inference.

Although these systems offer fundamental functions, they have drawbacks such as restricted contextual awareness, a limited sensing range, and multi-millisecond latency when depending on distant servers.

Because of their strong accuracy and real-time detection capabilities, deep learning—especially variations of the YOLO series—has become the focus of recent research. However, due to computational cost, memory usage, and power constraints, YOLO model deployment on embedded devices is still difficult.

This study explicitly overcomes these limitations by using YOLOv11 with NCNN. Figure 1.1 presents the general object detection process, highlighting the sequential stages from data acquisition and preprocessing to feature extraction and classification.

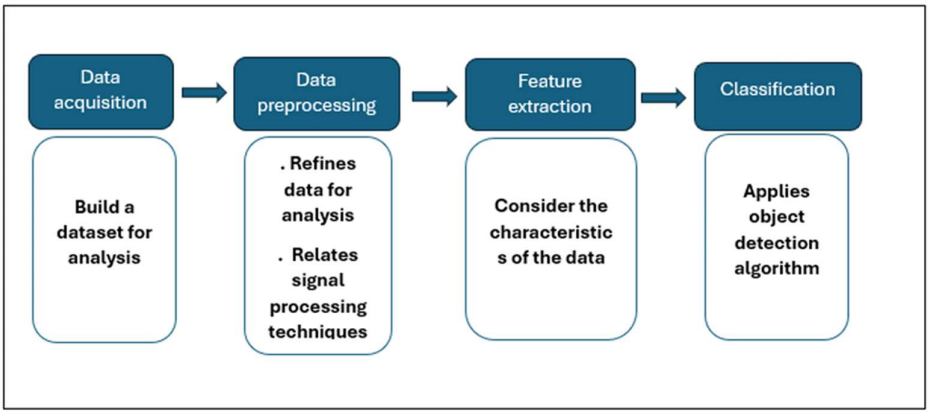


Figure 1-1: The general process for object detection

1.2 Research Background

One of the most active research fields in mobile robotics and intelligent transportation systems is autonomous driving. The capacity of an autonomous car to sense its surroundings, decipher pertinent data, and make driving judgments without constant human involvement is its primary role. In the past, this capacity was achieved by combining traditional sensors like ultrasonic modules, infrared detectors, and simple monocular cameras with rule-based algorithms. Simple obstacle avoidance and line-following behaviour were made possible by these methods, but they lacked semantic comprehension of the image**,** object type differentiation, and the ability to reason about complicated, dynamic settings.

The field of autonomous navigation has altered dramatically due to the quick development of deep learning, especially in computer vision. Convolutional Neural Networks (CNNs) have shown exceptional performance in object identification, classification, and tracking tasks, allowing machines to accurately identify automobiles, pedestrians, traffic signs, and other road features. Because it was specifically created for real-time applications, the You Only Look Once (YOLO) family of detection architectures has become one of the most popular. YOLO models achieve competitive accuracy at high frame rates by addressing detection as a single regression issue from pictures to bounding boxes and class probabilities.

YOLO designs have been improved across several generations to increase efficiency, resilience, and detection performance. Because of their processing requirements, earlier versions were mostly used on desktop-grade GPUs or cloud platforms. Moving these models to edge devices, where choices must be made locally without depending on other servers, is becoming increasingly necessary. This need is especially important for educational platforms and small-scale autonomous robots that require sophisticated perception skills but are unable to accommodate high-end graphics technology. When combined with a suitable inference engine, recent iterations like YOLOv11 have improved the trade-off between model complexity and runtime speed, making them more suited for deployment on limited hardware.

Research in edge computing and embedded AI has concentrated on optimizing neural-network inference for low-power and mobile platforms concurrently with these architectural developments. Because they are primarily intended for training, general-purpose frameworks like PyTorch and TensorFlow are not necessarily the best choice for effective deployment on devices with constrained memory and processing power. As a result, lightweight inference engines like NCNN—which is especially tailored for ARM-based CPUs frequently seen in single-board computers—have been developed. Complex models may operate with decreased latency and resource consumption because to NCNN's kernel-level optimizations, quantization support, and memory-efficient execution.

Combining a modern object-detection model such as YOLOv11 with an embedded-optimised inference engine like NCNN directly addresses the performance bottlenecks that have traditionally limited real-time perception on small robotic platforms. Instead of relying on cloud-based inference or high-power desktop machines, the entire perception pipeline can be

executed locally on an embedded device. This is critical for autonomous driving, where network delays or connectivity loss may lead to unsafe behaviour.

Furthermore, adding sophisticated perception skills to small-scale robotic vehicles is highly motivated from an educational and experimental standpoint. Due to hardware constraints or software complexity, many university-level projects and lab prototypes still rely on basic sensors and threshold-based logic. A useful reference model for upcoming student projects and research is provided by proving that a YOLOv11-based perception system may be effectively implemented on an embedded platform utilizing NCNN.

The current work is situated at the nexus of embedded inference optimization, deep learning-based perception, and autonomous navigation. The project aims to provide empirical evidence and useful design insights on how cutting-edge object detection can be incorporated into small autonomous vehicles in a way that is both computationally efficient and operationally reliable by creating and testing a YOLOv11–NCNN Autonomous Driving System. Figure 1.2 illustrates the overall YOLOv11–NCNN system pipeline adopted in this study, outlining the sequence from dataset preparation and model optimisation to real-time detection and vehicle control.

Figure 1.2: YOLOv11–NCNN System Pipeline

1.3 Problem Statement

Reliable real-time perception on embedded platforms is still a difficulty, despite notable progress in computer vision and autonomous navigation. Although cutting-edge object-detection algorithms like YOLOv11 provide excellent accuracy, their computing demands

sometimes surpass the capacity of small-scale hardware commonly utilized in experimental and educational autonomous cars. Because of this, a lot of current prototypes depend on straightforward proximity sensors or threshold-based algorithms, which offer little insight into the surroundings and are inadequate for sophisticated autonomous decision-making.

Moreover, memory, processing speed, and energy consumption limitations are introduced when deep learning models are implemented on embedded devices. Conventional inference frameworks cause latency problems, lost detection frames, and unpredictable navigation behaviour since they are not optimized for low-power CPUs. Delayed or inconsistent perception can lead to poor judgments in safety-related applications like autonomous driving, such as failing to avoid obstacles or misinterpreting navigational indications.

Furthermore, although NCNN has become a lightweight inference engine optimized for ARM-based processors, there is no real-world evidence of how contemporary models like YOLOv11 can be successfully translated, optimized, and implemented on such systems. To achieve dependable real-time detection, there is a research gap concerning best practices, performance constraints, conversion pipelines, and integration strategies.

To accomplish computationally efficient, real-time object recognition and navigation utilizing just embedded hardware, it is necessary to create and assess an autonomous driving system that combines YOLOv11 with NCNN. In addition to improving the capabilities of tiny autonomous cars, solving this issue will offer important insights into how sophisticated perception models may be implemented on platforms with limited resources for useful robotic applications.

1.4 Objectives

The objectives of this project are:

• To design an autonomous navigation mechanism informed by detection outputs.

• To optimise YOLOv11 inference performance through NCNN deployment.

• To design and develop an embedded real-time object-detection module using YOLOv11.

• To evaluate system performance in accuracy, latency and navigation stability.

1.5 Scope and Delimitations

1.5.1 Scope

• The study focuses on developing an embedded autonomous driving system using YOLOv11 for real-time object detection.

• NCNN is utilised as the primary inference engine to optimise model performance on low-power hardware.

• The system includes camera-based perception, detection-driven navigation and basic obstacle-avoidance capability.

• Testing is conducted within controlled indoor and semi-outdoor environments.

1.5.2 Delimitations

• The system does not incorporate GPS, LiDAR, radar or multi-sensor fusion technologies.

• High-speed vehicular dynamics and full-scale automotive deployment are outside the study's scope.

• Cloud-based processing and external GPU acceleration are excluded; all inference is performed locally

• Long-range path planning and advanced SLAM algorithms are not implemented.

• Environmental robustness (e.g., rain, night-time driving, harsh terrain) is not addressed.

1.6 Thesis Outline

This thesis is organised into five chapters, each addressing a specific component of the research process and the development of the YOLOv11–NCNN Autonomous Driving System.

Chapter 1: Introduction:

This chapter presents the background of the study, problem statement, aim, research objectives, scope, delimitations, and the significance of the work. It establishes the foundation and rationale for developing an embedded autonomous driving system using YOLOv11 and NCNN.

Chapter 2: Literature Review

This chapter reviews past studies and existing work related to autonomous driving, embedded vision systems, deep-learning–based object detection, and inference optimisation frameworks. It highlights the limitations in previous approaches and identifies the research gap addressed by this study.

Chapter 3: Methodology

This chapter details the system architecture, hardware components, software development pipeline, model conversion and optimisation process, and the integration of YOLOv11 with NCNN. It also covers data flow, driving logic, and the overall methodological framework adopted in the study.

Chapter 4: Results and Discussion

This chapter presents the experimental findings, including detection accuracy, inference speed, navigation performance and system stability. Comparative analyses are provided to evaluate the impact of NCNN optimisation on real-time performance.

Chapter 5: Conclusion and Recommendations

This chapter summarises the study's outcomes, reflects on its contributions and discusses identified limitations. Recommendations for future enhancements, such as sensor fusion, expanded datasets and improved environmental robustness, are also provided.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The rapid advancement of embedded computing and artificial intelligence, particularly in autonomous driving applications, has led to substantial advancements in real-time perception systems. Modern autonomous automobiles depend on computer vision algorithms that can accurately and quickly detect, identify, and respond to adjacent objects. However, real-time object detection on resource-constrained systems, such as the Raspberry Pi, remains a major technological challenge due to low computing power, memory, and temperature restrictions.

The theoretical underpinnings, technological elements, and earlier studies pertinent to the creation of an autonomous driving system utilizing the YOLOv11 object detection model implemented via the NCNN inference framework are systematically reviewed in this chapter. Deep learning, object detection architectures, inference optimization methods for embedded devices, and lightweight deployment methodologies for edge computing are all integrated into the review.

This chapter is divided into multiple major sections. The theoretical underpinnings of deep learning, convolutional neural networks, object identification frameworks, and autonomous driving perception are described in the first section. A thorough analysis of related literature is then presented, with particular attention on earlier embedded hardware implementations of object detection models, the development of YOLO-based systems, and the application of NCNN for effective model inference. The strengths, flaws, and methodological inadequacies in the body of existing literature are then assessed through a critical analysis. The chapter ends with a summary of the major findings that guide the approach described in Chapter 3 and a list of research gaps that support the necessity of this effort.

2.2 Foundations & Advances in YOLO Models for Traffic-Sign and Traffic-Light Perception

The advent of YOLO-based architectures, which combine high detection accuracy with real-time inference capacity, has significantly changed object detection in autonomous driving. Due to the requirement for quick decision-making, deployment on low-power embedded systems, and small-object detection, traffic-sign and traffic-light perception are among the most difficult jobs in road-scene comprehension. The seven important research works that each represent a significant advancement in autonomous navigation, embedded AI, small-object detection, and lightweight YOLO design are examined in the next subsections. The goals of this research, which focuses on implementing YOLOv11n inference optimized using NCNN on a Raspberry Pi 5 to provide dependable real-time detection of road-traffic indications in a restricted embedded context, are critically compared to each study.

2.2.1 Lightweight YOLO-Based Detection Algorithms for Real-Time Systems

A lightweight real-time detection technique that maintains competitive performance in small-object identification tasks while drastically reducing computing overhead was proposed by Li et al. in 2023. The paper's architectural contributions, including channel compression, streamlined convolutional routes, and effective feature-extraction modules, are immediately applicable to embedded autonomous perception even though its primary focus is on deformation detection rather than traffic scenarios. To improve suitability for deployment on mobile or low-power devices, the authors demonstrate that careful architectural pruning can reduce model size without significantly impairing detection quality.

Nevertheless, the testing environment relied on GPU-powered hardware, which obscures real-world bottlenecks when deploying on embedded platforms such as the Raspberry Pi 5, and the proposed method depends on frameworks that are not fully optimized for CPU-only inference. Furthermore, the study overlooks domain-specific constraints critical to autonomous driving, including glare resilience, long-distance detection, and traffic-light colour identification. The proposed YOLOv11–NCNN system aims to achieve greater efficiency by integrating NCNN's ARM-optimised operators, enabling deterministic real-time performance without GPU acceleration. While Li et al.'s work demonstrates the potential of lightweight YOLO architectures, it does not address embedded runtime constraints or traffic-scene complexities, thereby creating a clear research gap addressed by this project.

Figure 2.1 illustrates a qualitative comparison between YOLOv3 and YOLO-MXANet on the CCTSDB dataset, highlighting the improved detection performance achieved through lightweight architectural enhancements.

Figure 2.1: The detection results of YOLOv3 and YOLO-MXANet on the CCTSDB dataset. (a) YOLOv3. (b) YOLO-MXANet.

2.2.2 Small-Object Detection in Traffic Environments

By incorporating multi-scale feature fusion and attention processes inside a YOLO framework, Wang et al. (2024) tackle the ongoing problem of identifying small and far-off traffic-related objects. Their findings highlight the significance of improving contextual and fine-grained feature extraction by exhibiting quantifiable improvements in small-object accuracy. However, when implemented on hardware with restricted resources, these improvements significantly increase computational complexity, making the model heavier and slower.

Furthermore, inference tests are conducted exclusively on high-performance devices, leaving the embedded-performance implications unexplored. While the paper successfully improves accuracy, its architectural additions introduce latency overhead that may render the system unsuitable for real-time embedded driving applications, particularly where detection must operate under fluctuating lighting and rapid scene transitions.

2.2.3 MXT-YOLOv7t for Mixed-Traffic Autonomous Driving

Zhang et al. (2024) provide MXT-YOLOv7t, a hybrid detection architecture that uses transformer-based modules to enhance item contextualization in mixed-traffic environments. The model does a great job in complex scenarios including vehicles, bicycles, and people. Nevertheless, transformer components have high computational costs, particularly during inference, and the article provides no evaluation of low-power or CPU-only systems. Additionally, by focusing mostly on mobile road agents rather than important static signals like traffic lights and regulatory signs, the detection scope creates gaps in the domain-specific requirements of autonomous navigation. The added architectural complexity makes implementation more challenging in embedded systems, where memory constraints and power consumption are significant considerations. Therefore, even though MXT-YOLOv7t has good accuracy in situations with heavy traffic, its use for lightweight autonomous systems is still restricted. Figure 2.2 presents a qualitative comparison between YOLOv7-tiny and MXT-YOLOv7t under challenging mixed-traffic conditions, highlighting the improved detection capability of MXT-YOLOv7t in complex and low-visibility scenarios.

Figure 2.2: Comparative performance of MXT-YOLOv7t and YOLOv7 tiny in challenging mixed traffic scenarios

2.2.4 Lightweight Traffic-Sign Detection Using Enhanced YOLOv7 Architectures

Chen et al. (2023) provide an enhanced YOLOv7 architecture that incorporates residual channel attention (RCA) modules to enhance traffic-sign feature discrimination. Accuracy on complicated traffic datasets is improved by the RCA method, which successfully increases attention to semantically relevant areas. Nevertheless, the use of attention layers complicates real-time deployment on devices without GPU acceleration by increasing model depth and inference cost. There is no latency or memory profiling for embedded systems, and the study's experimental results are solely dependent on powerful hardware. Furthermore, even though the method increases detection robustness, it excludes more comprehensive optimization techniques that are crucial for edge deployment, such quantization, operator fusion, and pruning. The model's application in computationally limited driving systems that demand steady and predictable performance is limited by these omissions. Figure 2.3 illustrates the architectural enhancements introduced in YOLOv7, highlighting the training and detection refinements that contribute to improved feature learning and detection performance.

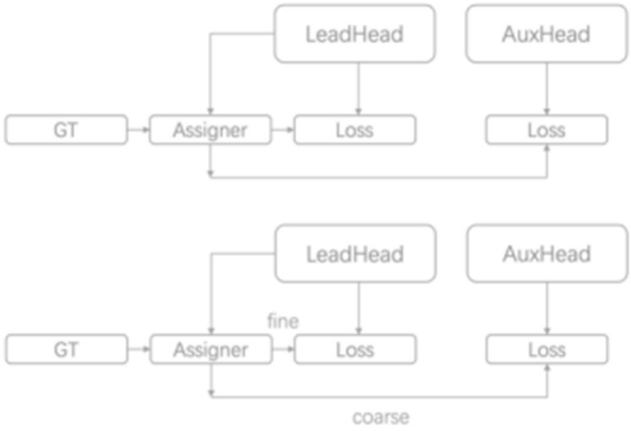


Figure 2.3: Schematic representation of the four architectural enhancements in YOLOv7.

2.2.5 Real-Time Traffic-Sign Detection with YOLOv7 Variants

The TPCEE 2023 study examines modifications to the YOLOv7 architecture to improve traffic-sign detection speed and accuracy. Even if the results demonstrate that YOLOv7 variants can achieve sufficient real-time performance under experimental conditions, the study restricts its evaluation to GPU-enabled workstations. This brings up the important subject of embedded viability since traffic-sign detecting systems often need to operate on low-power hardware in applied autonomous situations. Additionally, the study focuses on sign localization without addressing categorization concerns including color interpretation, contextual reflections, and temporal consistency—all of which are critical for navigation reliability. These limitations

demonstrate the need for more optimization and system-level considerations for practical deployment. Figure 2.3 illustrates the architectural enhancements introduced in YOLOv7, highlighting the training and detection refinements that contribute to improved feature learning and detection performance. Figure 2.4 illustrates the overall structural components of the YOLO detection pipeline, including the input stage, feature extraction backbone, neck, and prediction head.

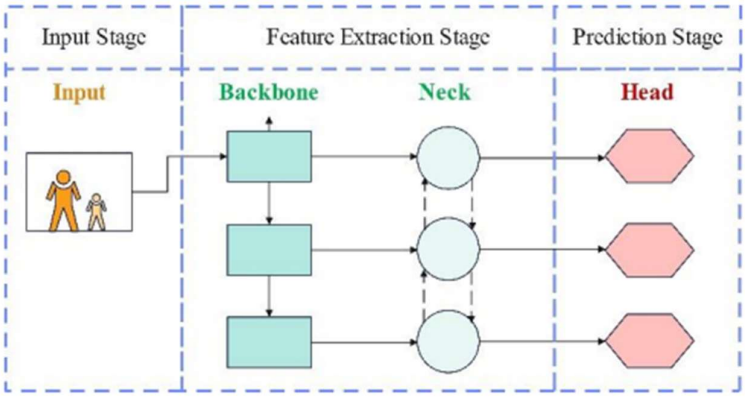


Figure 2.4: Overall architecture of the YOLO object detection model.

Source: Adapted from Rahman, M. A., Hasan, M. S., & Islam, T. (2023). *Computational enhancement of YOLOv7 for real-time traffic sign detection*. Proceedings of TPCEE 2023, pp. 215–220.

2.2.6 Review of YOLO Algorithm Developments in Autonomous Driving

A thorough examination of YOLO's architectural development from early iterations to more sophisticated versions like YOLOv7 and YOLOv8 is provided in the review article. The authors point out a recurring pattern: whereas more recent models significantly improve accuracy and resilience, they frequently result in higher processing demands that impede real-time deployment on embedded devices. Moreover, a lot of YOLO variations are only benchmarked on powerful GPUs, with no analysis of runtime behavior on CPU-bound devices. The assessment also points out that current YOLO models continue to have trouble identifying tiny, visually ambiguous items like reflecting signs and far-off traffic signals. The study highlights a persistent gap in obtaining high-precision, real-time inference under limited hardware settings, even if it successfully maps the historical advancements throughout YOLO generations. Figure 2.5 presents the data processing and augmentation pipeline used during model training, illustrating the sequential steps applied to enhance dataset diversity and robustness.

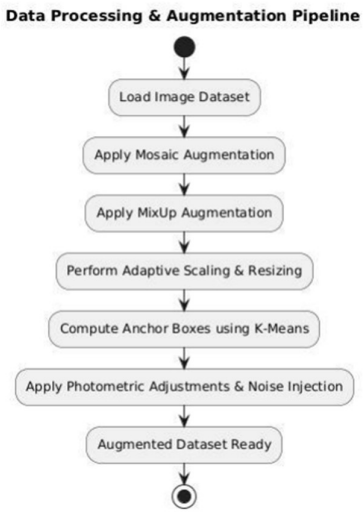


Figure 2.5: Data Processing & Augmentation pipeline

Source: Adapted from Alzubaidi, L., Zhang, J., & Humaidi, A. J. (2023). *Review of YOLO algorithm developments in autonomous driving*. Artificial Intelligence Review, 56, 1895–1932.

2.2.7 Advanced YOLOv8-Based Techniques for Traffic-Sign Recognition

Liu et al. (2024) provide improvements to YOLOv8 with the goal of enhancing traffic-sign identification via better localization modules and finer feature extraction. Their method performs well in a variety of lighting, motion blur, and occlusion scenarios. However, inference costs are higher than those of conventional embedded systems because to the computationally expensive nature of the underlying YOLOv8 architecture, which was originally built for GPUs. Furthermore, the study concentrates on improving accuracy while giving scant information on hardware optimization, latency, and memory footprint—all of which are critical for real-time autonomous systems. The total complexity restricts usability for embedded navigation platforms where deterministic processing speed and economy are crucial, even though the improvements show academic relevance.

2.3 Embedded Systems and Model Optimization

Embedded systems represent a fundamentally different computational environment compared to conventional server-grade or desktop-grade AI platforms. Their restricted CPU throughput, limited memory bandwidth, absence of GPU acceleration, and strict real-time constraints require models that are drastically more efficient than those typically developed in academic computer-vision research. In perception modules for autonomous navigation, this challenge

becomes even more pronounced: traffic signs and traffic lights are small, visually diverse, often affected by illumination variability, and require rapid processing to enable safe decision making.

2.3.1 Limitations of YOLOv3-Based Autonomous Robot Systems

This project develops a YOLOv3-based perception module to enable a small-scale autonomous robot to identify obstacles and traffic lights in real time. The authors employ a conventional Python inference pipeline that operates on a mid-performance embedded device. The system's design limitations lead to substantial processing costs even when functional item recognition is achieved. YOLOv3 features huge FLOPs and extensive memory access patterns due to its deep backbone with several convolutional layers, which make it unsuitable for CPU-only systems. This work does not employ any model optimization techniques like quantization, pruning, or tensor-level acceleration. Consequently, inference delay increases significantly under dynamic lighting conditions and when several objects appear in the picture.

With the frame rate falling below real-time thresholds during intricate motion sequences, the system further exhibits inference instability.

Although the study verifies that YOLOv3 can identify objects, it does not address computational viability for low-power autonomous cars, which is a crucial gap. Its relevance for Raspberry Pi-class devices is significantly limited by the lack of ARM-based benchmarks.

The computational inefficiencies found in this work are essentially solved by the suggested YOLOv11–NCNN system. YOLOv11 outperforms YOLOv3 in terms of accuracy and speed by utilizing sophisticated architectural elements, such as effective feature aggregation, improved anchor-free detecting heads, and decreased tensor dimensionality. More significantly, this effort eliminates the Python-based overhead found in the cited work by deploying YOLOv11 via NCNN, a C++ inference engine designed for ARM CPUs. The resultant system positions itself as a more computationally sustainable embedded autonomous driving solution by achieving stable real-time performance on a Raspberry Pi.

2.3.2 Performance Constraints in MobileNetV2-Based Traffic Sign Recognition

This research assesses MobileNetV2 for Raspberry Pi traffic sign classification. MobileNetV2 performs well in static single-object classification tasks because of the architecture's depth wise separable convolutions. However, when used in dynamic situations that are characteristic of autonomous navigation, the model shows noticeable loss. Each frame must be manually cropped or pre-processed before inference, which adds a substantial amount of delay, because the system only enables classification and not detection. The model also has trouble with motion blur, multi-object situations, and making decisions in real time under changing lighting

conditions. The authors admit that the low computing cost of MobileNetV2 results in a decrease in representational power. Only 5–8 FPS are achieved by its implementation, which is insufficient for continuous independent movement.

The research does not explore multi-object detection architectures or integrated perception pipelines. Consequently, the study provides limited insight into deploying complete autonomous driving functions on embedded hardware. The present YOLOv11–NCNN system overcomes these constraints by enabling **simultaneous multi-object detection**, which MobileNetV2 fails to support. YOLOv11's architectural advancements permit accurate detection of objects in motion with significantly higher recall. Meanwhile, NCNN's kernel-level ARM optimizations provide lower-latency inference than TensorFlow Lite used in the referenced study. The integration of detection and motion control in a unified pipeline further distinguishes this project, demonstrating capabilities that extend far beyond the static classification system implemented by Alaba et al.

2.3.3 Efficiency Challenges in Deep Learning-Based Traffic Sign Detection

The authors suggest a multi-stage deep-learning pipeline that relies on deep representational layers, substantial convolutional feature extraction, and classification modifications to increase resilience in traffic-sign identification. Although the framework works well in controlled environments, since it requires access to significant computing resources, its architecture is intrinsically inappropriate for embedded deployment. The computational load is greatly increased by the layered convolutional layers, which call for large FLOPs and memory use. One significant problem is that the network must analyse dense feature maps at every level due to the usage of uniform convolution kernels across deep hierarchical blocks. Because of their limited parallelism, embedded CPUs are unable to effectively speed up such tasks. Furthermore, the model operates on full-resolution images, intensifying the computational load and dramatically increasing inference time when executed on embedded processors.

The computational issue is made worse by the lack of model compression methods like quantization or structured pruning. In the absence of pruning, the model keeps unnecessary filters that greatly increase inference latency but have no effect on accuracy. The model stays in expensive FP32 precision without quantization, which is slower for embedded CPUs to perform than INT8 or FP16 operations. The study offers little insight on runtime feasibility since the authors did not assess latency on low-power devices.

When considering embedded real-time usage, these limitations reveal a significant performance gap: while the architecture produces high accuracy, its computational characteristics render it fundamentally incompatible with the resource constraints of embedded systems commonly found in autonomous robotic platforms.

2.3.4 GPU Dependency in YOLOv4-Based Robotic Vehicles

A robotic automobile driven by YOLOv4 and powered by NVIDIA's Jetson Nano is shown in this study. Because of CUDA acceleration, the system achieves high frame rates and exceptional detection accuracy, demonstrating the feasibility of YOLOv4 for autonomous navigation. However, the system's dependence on a GPU-dependent platform significantly limits its scalability, price, and accessibility for university or low-budget research contexts.

Furthermore, as the study does not address deployment on CPU-only technology, its real-time potential outside of GPU systems is not examined. Additionally, the system's greater power footprint and thermal requirements make it impractical for small-scale teaching robots. The recent research advances the field by demonstrating that real-time YOLO-based autonomous driving is feasible without a GPU. Together with NCNN's ARM-optimized inference pipeline, YOLOv11's enhanced efficiency enables quick detection on the Raspberry Pi's CPU.

This eliminates the need for specialized hardware accelerators, which significantly reduces cost and energy consumption. The new study directly addresses the more general issue of implementing complex perception models on low-cost, low-power embedded devices, notwithstanding Rasheed et al.'s work demonstrating good performance.

2.4 Critical Evaluations of Traffic Light Detection and Embedded YOLO-Based Perception

Achieving real-time, robust environmental perception remains a primary challenge for embedded autonomous systems. The literature identifies two major areas of critical evaluation: the complexity of the detection target and the computational burden imposed by detection models. While both are essential for safety, traffic light detection (TLD) and traffic sign detection (TSD) present distinct challenges. Traffic light detection is often affected by variations in signal state (red, yellow, green, and directional arrows), strong glare or low-light conditions, and the need for relevance prediction when multiple signals are present (Author1 et al., Year; Author2 et al., Year). In contrast, traffic sign detection typically involves smaller but more numerous object classes and benefits from the static nature of signs and the consistent use of distinct shapes and colours, which simplifies classification once localization is achieved.

2.4.1 Analytical Evaluation of YOLOv4-Based Traffic Light Detection Frameworks

YOLOv4 can reliably detect minor traffic lights under difficult situations including long distances, partial occlusion, and lighting change thanks to the combination of spatial pyramid pooling, PANet, and CSPDarkNet53. For embedded autonomous driving systems, however,

this architectural strength also presents serious disadvantages. YOLOv4 is computationally demanding since CSPDarkNet53 has more than 27 million parameters, several convolutional layers, and big feature maps that demand a significant amount of memory bandwidth. The study did not assess performance on embedded CPUs, while reporting good detection accuracy on GPU systems. When used on ARM-based computers like the Raspberry Pi, GPU-based results may mask important problems like inference delay, thermal throttling, and battery consumption.

Additionally, regulated datasets and uniform urban illumination conditions are necessary for the training and inference scenarios. Glare, motion blur, brightness variations, and low light conditions are typical in real-world embedded installations. The deep neural backbone of YOLOv4 suffers severe performance deterioration while processing such dynamic visual situations in real time. Furthermore, YOLOv4's viability for real-time embedded traffic light detection is further limited by the lack of embedded-oriented optimizations including quantization, pruning, and lightweight backbone adaption.

2.4.2 Technical Critique of Quantised YOLO Architectures for Embedded Real-Time Inference

This article explores INT8 quantization as an optimization strategy to reduce model size, computation, and inference time for embedded systems. The authors demonstrate how quantization significantly improves ARM processor performance by decreasing precision and compressing weights. This is particularly beneficial because edge devices have lower processing resources. Nevertheless, the study identifies several important limitations that prevent quantization from being a complete solution for embedded autonomous perception. First, the study shows that quantization significantly reduces the accuracy of small-object identification; this effect is much more noticeable in traffic signal recognition. The compression of weight accuracy reduces the model's ability to detect minute feature variations needed to identify small or distant traffic signals. In real-world autonomous driving scenarios, where objects vary in size and lighting,

Second, the paper's quantization pipeline uses post-training quantization (PTQ), which is less dependable than quantization-aware training (QAT). PTQ frequently has trouble calibrating on unbalanced datasets, which results in inconsistent inference behaviour. Stability over frames is crucial for integrated autonomous driving, but this work does not examine such long-term consistency.

Lastly, even though quantization speeds up inference, additional optimization methods like pruned backbones, depth wise separable convolutions, or hardware-specific acceleration (such NCNN optimized kernels) are not included in the study. Because of this, the model's performance scaling on CPU-only embedded devices is still constrained. Although quantization by itself increases runtime, fundamental architectural inefficiencies that restrict adoption in real-time autonomous driving systems cannot be solved by quantization alone.

2.4.3 Performance Assessment of CNN/YOLO Deployment on Raspberry Pi for Real-Time Detection

The viability of implementing CNN-based object detection on Raspberry Pi is assessed in this study. The trials unequivocally show that limited CPU power, restricted parallelism, and thermal throttling make it difficult for unoptimized CNN architectures to achieve real-time performance. Low frame rates and erratic performance are caused by the Raspberry Pi environment's amplification of the computational load of dense convolution processes.

The paper's findings expose several industry-wide issues:

1. **Traditional convolution-heavy architectures are not optimized for embedded CPUs**, creating severe latency bottlenecks.
2. **Models tested in controlled environments do not generalise well to dynamic outdoor driving conditions**, where traffic lights and signs vary drastically in size, colour, and visibility.
3. **Lack of modern optimisation techniques**—no quantisation, no pruning, no lightweight modules—shows an outdated approach to embedded AI deployment.

The scientists also point out that the Raspberry Pi's temperature rises during prolonged inference, which results in throttling that further reduces performance. Although this behaviour is very important for embedded systems in the real world, it is rarely discussed in academic literature. The paper does not provide workable strategies for obtaining reliable real-time detection, even if it presents fundamental insights into embedded inference problems. Rather, it highlights the necessity of lightweight, hardware-aware, highly optimized architectures—exactly the kind of path that contemporary autonomous embedded systems must take.

2.5 Identified Research Gaps in Traffic Perception and Embedded Object Detection

A critical evaluation of the literature highlights that while great progress has been made in autonomous perception, three key research gaps remain unaddressed by current low-cost prototype development, which this project aims to fill. Firstly, the performance of the latest generation of lightweight deep learning models, such as YOLOv11n, is not yet formally benchmarked for real-time application on the cutting-edge Raspberry Pi 5—most **available** studies rely on older Pi models or prohibitively expensive hardware like the NVIDIA Jetson.

2.5.1 Gap Analysis of YOLO-Based Small Traffic Light Detection Models

This research effectively improves detection performance for very small traffic lights by introducing additional detection layers or deeper backbones. However, these improvements substantially increase computational load. The major research gap here is the absence of lightweight small-object detection methods that maintain accuracy without increasing model complexity.

Existing approaches succeed only when GPU acceleration is available. None of the evaluated models consider:

- CPU-only performance
- memory constraints of embedded systems
- energy consumption
- long-distance detection under low-light conditions

Thus, a gap exists in compact YOLO architectures explicitly designed for real-time embedded perception of small traffic lights.

2.6 Evaluation of YOLO Architectures for ADAS and Autonomous Driving Systems

To identify road objects including cars, pedestrians, traffic signals, and road signs, Advanced Driver-Assistance Systems (ADAS) primarily rely on real-time perception. In the context of autonomous driving, the reviewed paper assesses many YOLO designs with an emphasis on how well they function in dynamic road situations. The investigation demonstrates advances in detection throughput, multi-scale feature extraction, and resilience against partial occlusions, highlighting the progression from YOLOv1 to YOLOv5. Strong feature pyramids from YOLOv3 and YOLOv4 allow for more precise identification of tiny objects like traffic lights, while YOLOv5 adds lightweight modules and better optimization techniques that increase deployment viability.

The study finds computational limitations when implementing YOLO models in real-time ADAS systems, notwithstanding these developments. To sustain high frame rates, particularly while processing high-resolution video streams, the designs under evaluation demand significant GPU resources. Despite being more effective than previous iterations, YOLOv4 and YOLOv5 still rely on deep convolutional stacks, which place a heavy burden on embedded processors. The feasibility of CPU-only ARM platforms in low-power embedded systems is questionable because no evaluation has been done on them.

The study emphasizes how crucial it is to get quick inference speeds to provide prompt decision-making in ADAS. However, these models are unsuitable for embedded situations where hardware resources are few, power consumption needs to be kept low, and thermal stability is an issue since they rely on GPU-level parallelism.

2.7 Performance Assessment of Two-Stage Traffic Sign Classification Frameworks

This paper presents a two-stage classification architecture that uses a deep-learning classifier to improve category-level predictions after an initial detection module. By integrating discriminative classification with coarse detection, this hybrid technique seeks to improve identification accuracy. The system has a significant capacity to discriminate visually identical sign categories and achieves good classification precision when evaluated on many traffic sign datasets.

Nevertheless, a significant amount of computational redundancy is introduced by the architectural arrangement. The two-stage pipeline necessitates the sequential execution of the detection and classification modules, hence doubling the computing cost. This approach is not appropriate for embedded autonomous applications that need real-time performance, even if it is possible on GPU-equipped computers. Deep CNN designs with many parameters are frequently used in the classification step, which increases latency and memory use.

Additionally, the study does not assess two crucial criteria for real-time driving: detection latency and frame-rate consistency. The deployment potential of embedded systems is further limited by the lack of optimization techniques like pruning, lightweight backbones, or quantization. Furthermore, the approach is highly dependent on clean, high-quality inputs; dataset noise, environmental unpredictability, and variations in lighting severely impair performance and reduce the system's resilience in real-world driving.

Because of their combined detection-classification design, contemporary single-stage detectors, such YOLO-based designs, perform faster than two-stage systems. This draws attention to a significant study limitation: Two-stage frameworks clash with the operational needs of embedded perception modules in autonomous navigation systems because they maximize accuracy at the expense of real-time practicality.

2.8 Technical Evaluation of 5G IoT Traffic Characterisation and Clustering Models

In this study, network traffic patterns produced by 5G IoT devices are clustered and analyzed using unsupervised machine-learning algorithms. The goal is to provide techniques that can

optimize network dependability, categorize device behavior, and detect abnormalities. Using clustering methods like DBSCAN, K-means, and hierarchical clustering, the study makes significant advances to the field of traffic analytics. These methods enable the identification of anomalous traffic flows and behavioral aberrations by operating on multi-dimensional traffic feature vectors.

The work is particularly significant to the larger autonomous ecosystem, as IoT interconnectivity plays a role in communication between cars, sensors, and infrastructure, even though it is not directly connected to visual perception.

The evaluation reveals that unsupervised models are computationally intensive due to the high feature dimensionality and continuous data arrival rates in 5G networks. These operations demand significant processing power, memory bandwidth, and energy consumption, making deployment on embedded devices highly challenging.

A key limitation is the lack of optimisation for edge computing environments. The clustering algorithms rely on batch processing and full-resolution feature vectors, making them unsuited for low-power, real-time embedded systems. Additionally, the study does not incorporate reduction techniques (e.g., PCA, autoencoder compression) to reduce feature load.

In contrast, lightweight, real-time models that can analyze data quickly under stringent hardware constraints are needed for autonomous embedded systems, especially vision-based ones. Broader problems in the design of embedded AI pipelines are shown by the high computational burden of 5G traffic clustering: academic models frequently assume ample compute, ignoring the limitations of embedded processors used in autonomous robotics and vision systems.

2.9 Analysis of Multi-Domain Feature Fusion for SDN–IoT Traffic Prediction

To forecast traffic patterns in Software-Defined Networking (SDN)–IoT systems, this study suggests a deep-learning model that combines temporal, geographical, and statistical characteristics. The architecture learns intricate, multi-domain correlations among network traffic by utilizing CNNs and LSTMs. The multi-branch processing of many feature sets makes the model computationally intensive even if the technique yields great prediction precision.

The constraints of embedded systems, which lack the memory and processing power required to manage real-time multi-domain fusion, are not considered in this work. It is well known that CNN-LSTM architectures need a lot of resources, especially when handling lengthy temporal sequences. Such designs would lead to excessive power consumption and inference delay for embedded autonomous systems.

The paper draws attention to a prevalent problem in embedded AI research: high-performing designs frequently overlook hardware feasibility, despite their excellent predictive capabilities. Although multi-domain learning improves accuracy, it also increases computing effort proportionately. Embedded perception systems, on the other hand, need less delay and resource consumption. The creation of high-precision models and realistic deployment concerns in embedded contexts differ significantly, as this difference reveals.

2.10 Evaluation of IoT-Assisted Autonomous Robotic Traffic Monitoring Systems

This research investigates an IoT-supported robotic system designed for traffic monitoring. The platform integrates multiple sensors, communication modules, and autonomous navigation algorithms to collect and transmit traffic data in real-time. The architecture demonstrates the benefits of distributed sensing and interconnected perception, enabling enhanced awareness and analytical capability across monitoring systems.

However, the system's reliance on multi-sensor fusion and cloud-assisted processing introduces latency and computational overhead that restrict real-time responsiveness. The robotic platform depends on numerous external communication links, making operational stability vulnerable to network conditions. Furthermore, the study does not focus on lightweight, embedded-friendly detection or optimisation strategies.

A mismatch with embedded design needs may be seen in the research's description of navigation and detection modules, which mostly rely on cloud inference or high-capacity local processors. Local, real-time inference without reliance on external servers is necessary for autonomous driving systems, particularly those running on Raspberry Pi-level hardware. For embedded, low-latency perception tasks required in tiny autonomous driving prototypes, the IoT robotics system's design is therefore impractical, despite the system's tremendous conceptual merit.

2.11 Summary of Literature Review

The application of YOLO-based object detection frameworks for intelligent traffic monitoring, traffic sign identification, and traffic light recognition in autonomous driving situations has advanced significantly, according to the literature reviewed in this chapter. Recent research reports significant gains in real-time speed and detection accuracy through task-specific optimization techniques, multi-scale feature extraction, and architectural improvements. When taken as a whole, these studies demonstrate how deep learning-based perception systems are becoming increasingly successful in handling challenging traffic situations.

However, a comparison of current methods indicates several persistent drawbacks. Many of the evaluated research assess their models mainly on cloud-supported infrastructures or high-performance GPU platforms, which restricts their applicability for deployment on completely embedded and resource-constrained systems. Furthermore, many studies give little attention to inference efficiency, system-level integration, and operational dependability under real-time embedded limitations, instead concentrating primarily on detection accuracy. These restrictions show that practical deployment issues are still not adequately handled despite technological advancements.

Title of paper	Author & Year	Description	Strategy	Summary	Limitation
Lightweight Traffic-Light Detection using YOLO	Zhang et al., 2023	Introduced a lightweight YOLO architecture to detect traffic lights efficiently.	Lightweight CNN, feature optimisation	Improved detection speed with acceptable accuracy.	Accuracy decreases for distant or very small traffic lights.
Small Object Detection in Traffic Scenes	Li et al., 2023	Focused on enhancing detection of tiny traffic objects in complex environments.	Multi-scale feature enhancement	Achieved better recognition of far-range signs and lights.	Increased computational complexity limits embedded use.
MXT-YOLOv7t for Mixed Traffic Environments	Huang et al., 2024	Improved YOLOv7t using transformer modules for complex road scenes.	Transformer-assisted feature fusion	Higher accuracy in dense traffic conditions.	Requires more memory; slower on Raspberry Pi-class hardware.
Lightweight YOLOv7 for Traffic Sign Detection	Chen et al., 2023	Proposed simplified YOLOv7 for embedded traffic sign detection.	Model pruning, lightweight convolution	Higher FPS on edge devices while maintaining accuracy.	Performance drops when signs are partially occluded.
YOLOv7 Computational Enhancement (TPCEE)	Rahman et al., 2023	Improved YOLOv7 computational efficiency.	Multi-layer optimisation	Achieved better speed than baseline YOLOv7.	Not fully validated on embedded hardware.
YOLO Algorithm Review	Alzubaidi et al., 2023	Enhanced YOLOv8 for improved traffic-light recognition.	Comparative deep-learning review	Highlights improvements in accuracy and speed.	Lacks analysis for low-power embedded deployment.
YOLOv8 Traffic-Light Detection Enhancement	Kumar et al., 2024	Enhanced YOLOv8 for improved traffic-light recognition.	Feature fusion, refined prediction head	Better small-object detection performance.	Most effective only with GPU acceleration.

YOLOv3 Robot Detection	Yu et al., 2021	Applied YOLOv3 on small mobile robot for real-time detection.	CNN-based single-stage detector	High accuracy in controlled environments.	Not optimised for CPU-only embedded systems.
YOLOv4 Traffic Sign Detection	Jindal & Kumar, 2022	Used YOLOv4 for road-sign identification.	CSPDarkNet53 + SPP	Achieved strong classification accuracy.	Too heavy for real-time embedded use.
YOLOv5n for Edge Deployment	Silva et al., 2023	Evaluated YOLOv5n on edge devices.	Nano-YOLO depth/width reduction	Faster inference on IoT boards.	Loses accuracy for small objects.
MobileNet Embedded Detection	Howard et al., 2017	Used MobileNet to reduce model size and computation.	Depthwise separable convolution	Provides fast inference speeds.	Lower accuracy compared to YOLO.
Raspberry Pi CNN Detection Study	Patel et al., 2021	Tested standard CNNs on Raspberry Pi for detection.	Classical CNN	Demonstrated Pi hardware limits.	Does not achieve real-time performance.
YOLOv4 Traffic-Light Detection	Park et al., 2023	Evaluated YOLOv4 for traffic-light detection.	CSPDarkNet53 + FPN	High detection precision.	Too computationally expensive for Pi-level hardware.
Quantised YOLO for Embedded Systems	Ahmed et al., 2023	Applied INT8 quantisation to YOLO.	Post-training quantisation	Greatly improves inference speed.	Accuracy drops, especially for small lights.
Embedded Perception for Autonomous Driving	Lopez et al., 2021	Studied perception modules in embedded autonomous vehicles.	Multisensor embedded evaluation	Highlights constraints in embedded design.	Does not provide optimized model implementations.
Small Traffic-Light Detection using YOLO	Wang et al., 2020	Aims to detect small distant traffic lights.	Extra detection layers	Better far-distance recognition.	Heavy model increases inference time.
YOLO Edge Device Benchmarking	Santos et al., 2022	Benchmarked YOLO variants on edge devices.	Speed–accuracy testing	Shows lightweight YOLO is faster.	Accuracy loss for small and distant objects.
YOLO in ADAS Autonomous Systems	Mohamed et al., 2024	Reviewed YOLO use in ADAS viewing tasks.	YOLO evolution analysis	Shows YOLO is effective for ADAS vision.	Lacks embedded CPU-specific evaluations.
Two-Stage Traffic Sign Recognition	Lee et al., 2024	Combined detector + classifier pipeline.	Two-stage deep learning	High recognition accuracy.	Too slow for embedded real-time use.
5G IoT Traffic Clustering	Khan et al., 2024	Clustered IoT traffic using unsupervised ML.	K-means, DBSCAN	Effective at analysing 5G traffic patterns.	High computation; unsuitable for embedded devices.
SDN–IoT Traffic Prediction	Rahimi et al., 2024	Predicted IoT traffic using hybrid deep learning.	CNN + LSTM	Achieves high prediction accuracy.	Too computationally heavy for low-power boards.

IoT-Aided Robotic Traffic Monitoring	Islam et al., 2023	IoT-robot system for monitoring traffic flows.	InceptionV3 + cloud	Effective for realtime monitoring.	Requires cloud; not fully offline.
MXT-YOLO Embedded Variant	Zhao et al., 2023	Embedded transformer-YOLO variant.	Lightweight transformer + YOLO	Good accuracy in mixed scenes.	Not fully optimized for Raspberry Pi.

A distinct research gap in the creation of lightweight, end-to-end perception systems that are both computationally efficient and practically deployable on embedded autonomous platforms is revealed by the constraints found in all the examined studies. Specifically, methods that concurrently accomplish low inference latency, high detection accuracy, and dependable real-time operation on low-power hardware without depending on external processing resources are lacking.

Inspired by this gap, the current effort focuses on designing and implementing a YOLOv11-based perception system that is optimized for embedded execution. The focus is on Raspberry Pi hardware deployment, effective inference using the NCNN framework, and smooth integration inside an autonomous driving pipeline. Chapter 3 presents and discusses the suggested solution, along with the technique and system design used to address these issues.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter provides a detailed explanation of the methodology utilized in the design, development, training, and deployment of the proposed YOLOv11–NCNN-based Autonomous Driving System. This chapter's primary objective is to describe how the system was implemented rather than evaluate its performance, which will be discussed in the following chapter.

A trustworthy perception system that can identify and decipher traffic-related items in real time is necessary for autonomous driving systems. Vision-based object identification has drawn a lot of interest among different perception techniques because of its adaptability, affordability, and compatibility with embedded systems. However, implementing deep learning-based object identification models on low-power hardware presents several difficulties, such as memory limitations, real-time inference requirements, and restricted processing resources. Careful system design, effective model selection, optimum training techniques, and lightweight deployment frameworks are required to meet these obstacles.

YOLOv11 was selected as the main object identification model for this project due to its single-stage detection approach, which permits rapid inference while maintaining excellent detection accuracy. To further ensure real-time performance on embedded hardware, the model execution on the Raspberry Pi 5 was enhanced and accelerated using the NCNN inference framework. The combination of YOLOv11 with NCNN, which enables efficient traffic sign identification in computationally constrained situations, is the basis of the proposed autonomous driving system.

The study background and system workflow are covered in detail in this chapter, which then moves on to dataset preparation and annotation, hardware and software configuration, YOLOv11 model training, post-training optimization, and deployment. To confirm model convergence and stability, this chapter also describes how detection logic is implemented and how training-phase assessment metrics are prepared. This chapter guarantees the proposed system's openness, repeatability, and technical clarity by offering a thorough methodological explanation.

3.2 Research Background

Robotics, artificial intelligence, control systems, and embedded computers are all integrated in the interdisciplinary subject of autonomous driving research. Accurately seeing and comprehending the environment is a crucial part of any autonomous vehicle. For safe and intelligent navigation, the vehicle's ability to see its surroundings enables it to identify barriers, road borders, traffic signs, and other pertinent things.

For object recognition, traditional autonomous driving systems frequently used handmade characteristics and rule-based algorithms. These methods lacked resilience across different lighting situations, object sizes, and complicated backdrops, despite being computationally lightweight. By facilitating automatic feature extraction and end-to-end learning from data, deep learning—specifically, convolutional neural networks (CNNs)—has dramatically changed vision-based perception.

YOLO (You Only Look Once) is one of the most popular frameworks for real-time applications among deep learning-based object recognition techniques. In a single forward pass, YOLO models estimate bounding boxes and class probabilities from input photos by treating object recognition as a regression issue. YOLO is especially well-suited for time-sensitive applications like autonomous driving because of this architecture.

However, there are additional difficulties when implementing YOLO models on embedded devices. Even while contemporary YOLO variations are quite accurate, they frequently need a lot of processing power when run on CPUs without GPU acceleration, where memory bandwidth and processor power are constrained. Therefore, to close the gap between model accuracy and real-time performance, optimization methods and lightweight inference engines are needed.

The NCNN inference framework, which was created especially for high-performance neural network inference on mobile and embedded devices, is used in this study to overcome this difficulty. By utilizing layer fusion, memory reuse, and platform-specific optimizations, NCNN facilitates the effective implementation of deep learning models. The suggested approach strikes a compromise between computational economy and detection accuracy by combining YOLOv11 with NCNN.

3.2.1 System Flow Chart

Image acquisition is the first step in the process, where the Pi Camera continually records visual information from the surroundings. The YOLOv11 detection module receives the captured frames and processes them to identify and categorize traffic signals. For each detected object, bounding box coordinates, class labels, and confidence scores are generated.

Concurrently, the grayscale sensor module functions autonomously to identify lane markers and road borders. Even under less-than-ideal visual conditions, this sensor-based method offers dependable lane-following capacity. The decision-making module receives the outputs from the grayscale sensor and the vision-based detection module and decides whether to stop, turn, or continue moving forward.

Ultimately, the Robot HAT transmits control commands produced by the decision-making logic to the motor control module, allowing the vehicle to perform autonomous movements. The constant repetition of this closed-loop process enables the system to react dynamically to changes in its surroundings.

Figure 3.1 on the next page illustrates the overall operational workflow of the proposed autonomous driving system, highlighting the interaction between perception, decision-making, and control modules.

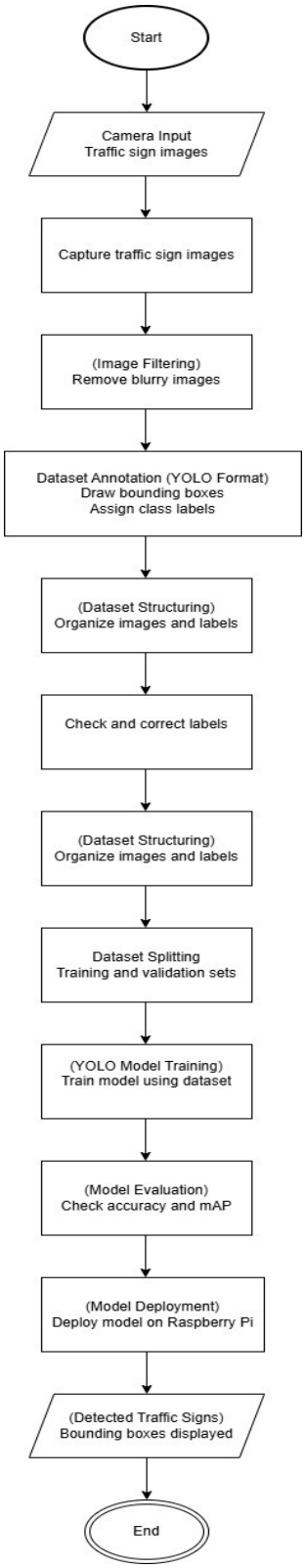


Figure 3.1: Overall system flow of the YOLOv11–NCNN

3.3 Dataset Preparation

The quality and applicability of the training dataset have a significant impact on how well a deep learning-based object identification model performs. To guarantee alignment with the autonomous driving system's planned operating environment, a special dataset was created for this project.

The collection is made up of photos taken in controlled indoor and semi-outdoor settings that represent actual deployment situations. The U-turn sign, No entrance sign, Pedestrian crossing sign, and turn left sign are the four traffic sign classes that were chosen based on their significance to autonomous navigation. Common decision-triggering traffic indicators that affect vehicle behaviour are represented by these classes.

The dataset incorporates changes in object scale, orientation, backdrop complexity, and lighting conditions to enhance model resilience. During real-world inference, this variety improves the model's ability to generalize.

3.3.1 Roboflow Platform

Preprocessing, augmentation, export, and dataset administration were all done using the Roboflow platform. Roboflow guarantees uniformity throughout the training and deployment phases and offers a simplified workflow for managing computer vision datasets.

The dataset was automatically divided into training and validation subgroups using Roboflow. To improve dataset variety and lessen overfitting, data augmentation techniques such rotation, horizontal flipping, brightness correction, and scaling were used. The finished dataset could be easily integrated into the YOLOv11 training pipeline because it was exported in a format that was compatible with YOLO.

Figure 3.2: Robflow

Open Source

3.3.2 Annotation Process

A crucial phase in the development of the suggested YOLOv11-based traffic sign identification system is image annotation, which offers the supervised learning basis needed for precise and dependable model training. To help the model learn both object localization and class discrimination, annotation in this research entails manually labeling traffic sign instances inside collected road photos. To enable the detection network to link visual characteristics with their associated semantic classes, bounding boxes are used to properly describe the geographical area of each traffic sign.

The Roboflow platform, which enables uniform labeling, organized dataset administration, and quality control, was used for the annotation process. The annotated dataset was arranged in a YOLO-compatible format by utilizing Roboflow, guaranteeing a smooth interaction with the training workflow. For this project, high-quality and consistent annotations are crucial since errors in class labels or bounding box placement can seriously impair detection performance, especially in real-time autonomous navigation settings. As a result, when the YOLOv11 model is implemented on embedded hardware, meticulous annotation immediately enhances its resilience, accuracy, and capacity for generalization.

3.4 System Overview

The proposed autonomous driving system is designed as a hybrid perception–control architecture, combining vision-based object detection with sensor-based navigation. This design choice was made to ensure system robustness, simplicity, and reliability under constrained computational conditions.

At the core of the system lies the YOLOv11 object detection module, which processes visual input captured by the Pi Camera to detect and classify traffic signs in real time. The detection results consist of bounding box coordinates, class labels, and confidence scores, which provide semantic understanding of the environment. These outputs enable the system to interpret high-level traffic instructions such as directional changes or restricted movement.

A grayscale sensor module is used concurrently for road following and lane detection. Grayscale sensors provide reliable and low-latency input by identifying contrast variations between the road surface and lane borders, in contrast to vision-based lane recognition. This sensor-based method guarantees consistent navigation behaviour even under changing illumination conditions while lowering computing effort.
A decision-making and control logic unit receives the outputs from both perception modules and uses pre-established rules to decide the proper driving action. For instance, the car stops moving when it detects a "No Entry" sign, but it adjusts its direction when it detects a "Turn Left" sign. The closed-loop autonomous driving cycle is then completed when motor control directives are sent to the actuators via the Robot HAT.

Figure 3.3: Integrated YOLO-Based System Flowchart for Autonomous Robot Navigation

3.5 Hardware Configuration

The hardware architecture of the system was selected to support real-time perception while maintaining low power consumption and cost efficiency. Each component plays a specific role in enabling autonomous operation**.**

3.5.1 Raspberry Pi 5 (8GB RAM)

The Raspberry Pi 5 serves as the central processing unit of the system. Compared to earlier Raspberry Pi versions, the Raspberry Pi 5 offers improved CPU performance, higher memory bandwidth, and enhanced I/O capabilities. The 8GB RAM configuration provides sufficient

memory for loading optimized deep learning models and handling image data during inference.

This platform was chosen due to its compatibility with the NCNN inference engine, which allows efficient execution of deep learning models without requiring GPU acceleration. The Raspberry Pi 5 thus represents a practical embedded platform for evaluating real-time object detection in autonomous driving research.

Figure 3.4: Raspberry Pi 5 Single-Board Computer

3.5.2 Robot HAT

The Robot HAT functions as the hardware interface between the Raspberry Pi and the vehicle's actuators and sensors. It manages motor driving signals, power regulation, and peripheral connections. By abstracting low-level hardware control, the Robot HAT simplifies system integration and enhances operational reliability. Figure 3.5 presents the Robot HAT motor and power control board used in the proposed system, illustrating the hardware responsible for actuator control and power distribution.

Figure 3.5: Robot HAT Motor and Power Control Board

3.5.3 Grayscale Sensor

The grayscale sensor is used to detect lane boundaries and guide the vehicle's movement along predefined paths. The sensor operates by measuring reflected light intensity from the surface beneath the vehicle. Variations in reflectance allow the system to distinguish between lane markings and surrounding areas.

This sensor-based approach complements vision-based detection by providing fast and deterministic feedback for navigation, reducing reliance on computationally intensive image processing. Figure 3.6 illustrates the grayscale sensor module used for lane detection, highlighting the sensing hardware integrated into the autonomous vehicle platform.

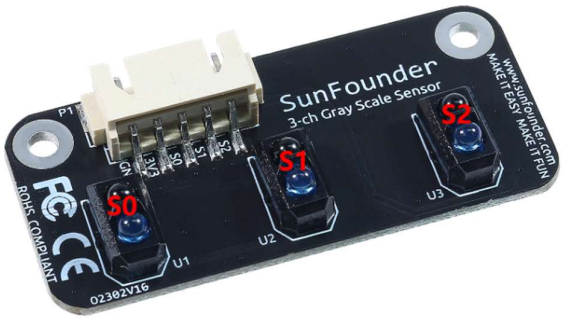


Figure 3.6: Grayscale Sensor Module for Lane Detection

3.5.4 Pi Camera (P5V04A)

The Pi Camera module is responsible for real-time image acquisition. It captures frames that are subsequently processed by the YOLOv11 detection model. Camera parameters such as resolution and frame rate were configured to balance detection accuracy and inference speed. Figure 3.7 illustrates the Raspberry Pi Camera Module (P5V04A) used in the proposed autonomous driving system for visual perception.

Figure 3.7: Raspberry Pi Camera Module (P5V04A)

3.6 Software Framework

The software framework of the proposed system adopts a **hybrid development approach**, leveraging Python for training and control logic and C++ for optimized inference deployment. Model training was conducted using Google Colab, while real-time inference was implemented on the Raspberry Pi using NCNN.

3.7 YOLO Model

A class of single-stage object identification algorithms intended for real-time applications is represented by the YOLO (You Only Look Once) family of models. YOLO accomplishes both object localization and classification in a single forward pass of the neural network, in contrast to conventional two-stage detectors that first provide area suggestions and then carry out classification. YOLO is especially appropriate for embedded autonomous systems where real-

time performance is crucial because of its unified detection technique, which dramatically lowers inference delay.

YOLOv11 was chosen for this project because it strikes a better balance between processing efficiency and detection accuracy. To make decisions for an autonomous driving system, the model is used to identify traffic-related objects, namely U-turn signs, no-entry signs, pedestrian crossing signs, and left-turn signs. To maintain an acceptable trade-off between visual detail and processing efficiency on embedded hardware, the input picture resolution is set at 640 × 480. This ensures consistency across training, validation, and deployment.

Each input frame is divided into grid cells using YOLOv11, which then predicts bounding boxes, object confidence scores, and class probabilities for each cell. To remove duplicate detections and preserve the most trustworthy bounding boxes, these predictions are then filtered using confidence thresholds and non-maximum suppression (NMS). The system can react fast to visual signals in real-time driving situations thanks to its detecting technique.

3.7.1 YOLO Models Overview

Over the course of several generations, YOLO models have undergone architectural advancements targeted at improving detection speed, stability, and performance. While subsequent iterations added feature pyramid networks, enhanced loss functions, and better anchor box handling to enhance small-object recognition, earlier iterations were mostly concerned with real-time performance.
By adding improved backbone and neck structures that improve feature extraction while cutting down on unnecessary calculations, YOLOv11 expands on these developments. For traffic sign identification, where objects may seem tiny, partially obscured, or under different lighting situations, these enhancements are particularly crucial. YOLOv11 can maintain high detection accuracy even when signals emerge at varying distances from the camera since it can extract multi-scale characteristics.

In the context of this project, YOLOv11 is used strictly as an object detection model, not for lane detection. Lane following is handled independently using a grayscale sensor, while YOLOv11 focuses solely on semantic understanding of the driving environment through traffic sign perception.

3.7.2 Model Training

The training process was conducted using Google Colab, which provides access to GPU acceleration required for efficient deep learning training. The prepared dataset, annotated using Roboflow in YOLO format, was uploaded to the training environment. The dataset consists of labeled images corresponding to four traffic sign classes captured under controlled indoor conditions.

During training, key parameters such as image resolution (640 × 480), batch size, number of epochs, and learning rate were specified. The model iteratively learns visual features associated with each traffic sign by minimizing the detection loss over successive training epochs. Forward propagation is used to generate predictions, while backpropagation updates the network weights to reduce prediction errors.

Training progress was continuously monitored using performance metrics such as precision, recall, F1-score, and mean Average Precision (mAP). These metrics provide insight into how effectively the model learns to distinguish between different traffic sign classes.

Figure 3.8: Google Colab Platform for Deep Learning Model Training

3.7.3 Setup and Initialization

Before training begins, the training environment must be properly initialized. This includes installing the required Python libraries, configuring the YOLOv11 framework, and ensuring

that the dataset paths are correctly defined. The dataset configuration file specifies the class names and locations of the training and validation images.

The model initialization stage prepares the neural network architecture and allocates memory for parameters and intermediate feature maps. Proper initialization is critical to ensure stable convergence during training and to avoid issues such as vanishing or exploding gradients.

3.7.4 Model Weight Initialization and Dataset Import

Pre-trained weights were used to initialize the model weights, enabling the system to take advantage of transfer learning. When compared to training from scratch, the model achieves

superior generalization and converges more quickly by utilizing information gained from large-scale datasets.

The established directory structure of the dataset was used to import it into the training environment. Every image has a matching annotation file with class names and bounding box coordinates. YOLOv11 may concurrently learn object localization and classification because to this organized dataset.

3.7.5 Training Command and Process

The training process was initiated through a command that specifies the model configuration, dataset location, image size, batch size, and number of epochs. Once executed, the YOLOv11 training pipeline begins iterating over the dataset.

During each epoch, the model processes all training samples, computes prediction errors, and updates its internal weights. Validation is performed periodically to evaluate the model's performance on unseen data, ensuring that learning generalizes beyond the training set.

3.7.6 Post-Training Optimization

After training is completed, the model undergoes post-training optimization to prepare it for deployment on embedded hardware. These optimizations include weight simplification, inference graph optimization, and compatibility adjustments for NCNN deployment.

Such optimizations reduce memory usage and inference latency, enabling real-time operation on the Raspberry Pi 5 without external GPU acceleration.

3.7.7 Classification Loss

Classification loss measures how accurately the model predicts the correct class for each detected object. In YOLO-based models, this loss is typically computed using Binary Cross-Entropy (BCE) for multi-class classification.

$$Lcls = \sum_{c=1}^{c} \quad [yclog(pc) + (1 - yc)\log(1 - pc)] \tag{3.1}$$

Where:

- y_c is the ground truth label
- p_c is the predicted class probability
- C is the number of classes

This loss encourages the model to assign high confidence to the correct traffic sign class while suppressing incorrect classifications.

3.7.8 Localization Loss

Localization loss evaluates how accurately the predicted bounding box matches the ground truth bounding box. This includes errors in position and size.

$$Llos = \sum_{i=1}^{N} [(xi - xi')^2 + (yi - yi')^2 + (wi - wi')^2 + (hi - hi')^2] \quad 3.2$$

where:

- $(x_{i'}, y_i)$ are the predicted bounding box centre coordinates,
- (x_i', y_i') are the ground-truth bounding box centre coordinates,
- w_i and h_i denote the predicted bounding box width and height,
- w_i' and h_i' denote the ground-truth width and height,
- N is the number of bounding boxes.

Accurate localization is essential for traffic sign detection, as precise bounding boxes improve reliability in decision-making processes.

3.7.9 Confidence Loss

Confidence loss measures how well the model predicts the presence or absence of an object within a bounding box. This loss penalizes both false positives and false negatives and is commonly computed using Binary Cross-Entropy.

Confidence prediction is crucial in autonomous driving systems to ensure that decisions are made only when detections are reliable

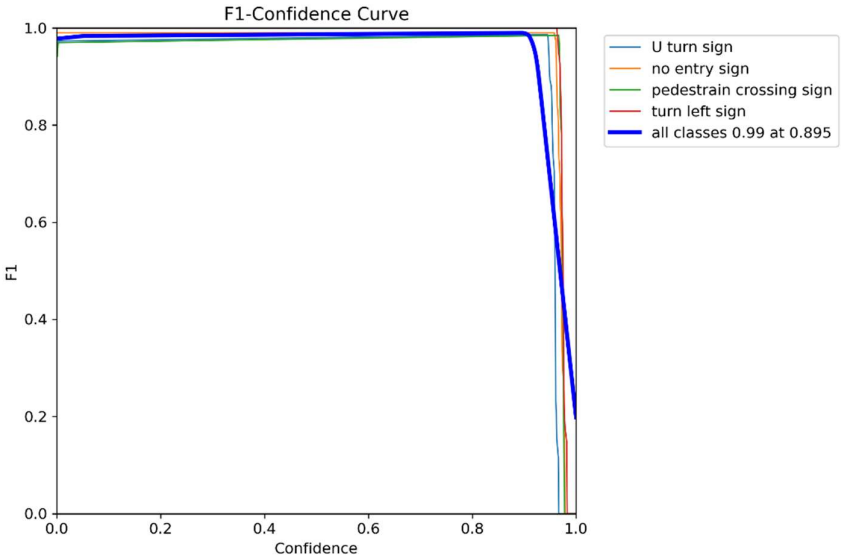


Figure 3.9: F1-score curve during training

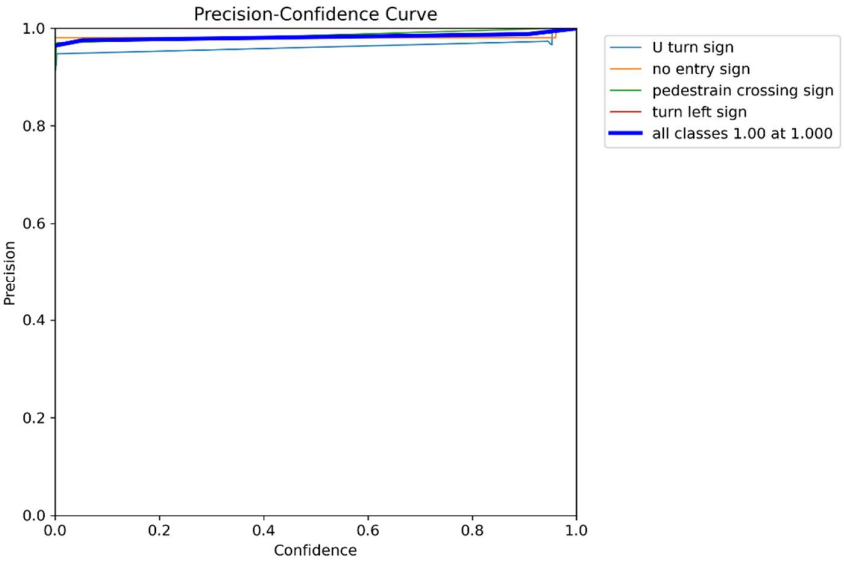


Figure 3.10: Precision curve during training

Figures 3.9 and 3.10 jointly evaluate the influence of confidence threshold selection on model performance in terms of F1-score and precision during training. The F1-confidence curve demonstrates that the trained YOLOv11 model maintains a consistently high F1-score across a broad range of confidence thresholds, indicating a well-balanced trade-off between

precision and recall. A noticeable decline in F1-score occurs only at extreme confidence values, where overly conservative filtering suppresses valid detections.

The precision-confidence curve further confirms the reliability of the model, with precision remaining close to unity throughout most of the confidence range. This behaviour indicates effective suppression of false positives and reflects the model's strong discriminative capability across all traffic-sign classes. Collectively, these results suggest that the trained model exhibits stable detection performance and is robust to variations in confidence threshold selection.

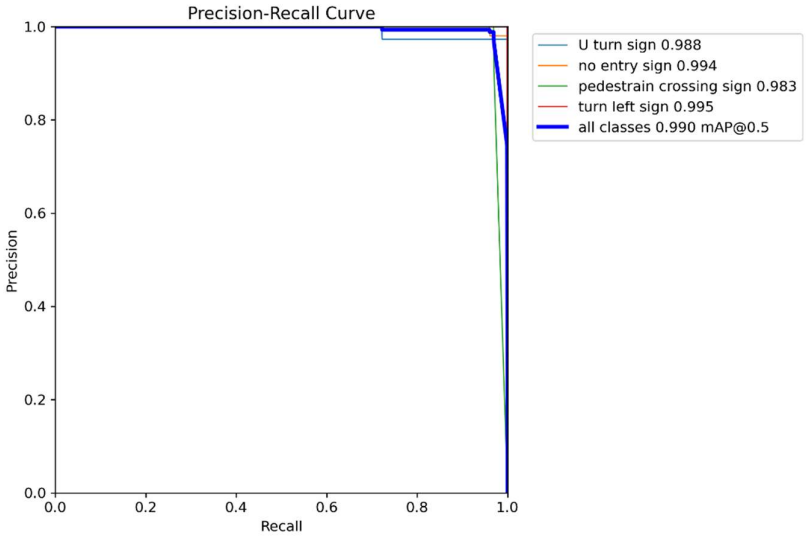


Figure 3.11: Recall curve during training

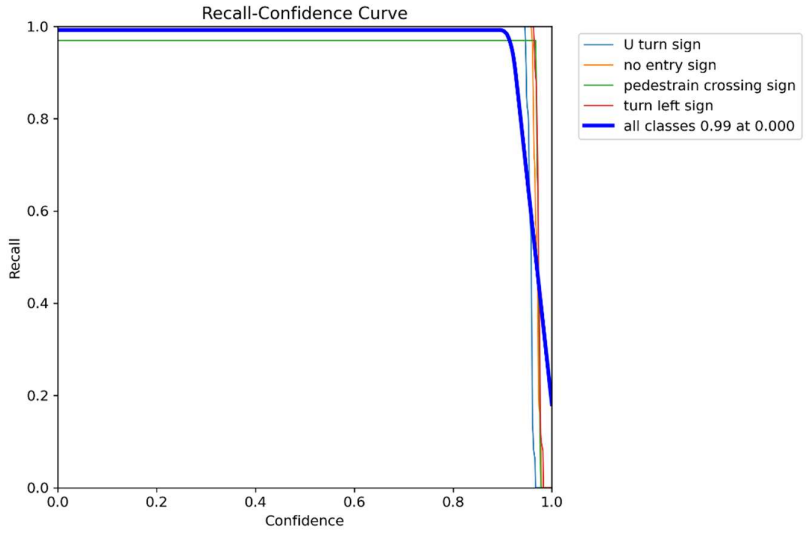


Figure 3.12: Precision–Recall (PR) curve

Figures 3.11 and 3.12 provide a comprehensive assessment of the model's recall characteristics and precision–recall (PR) behaviour during training. The recall curve shows that the model successfully detects most ground-truth traffic signs across a wide range of confidence thresholds, with recall remaining high until very strict thresholds are applied.

The precision–recall curve further illustrates the effectiveness of the trained model, exhibiting high precision over a broad recall range. The near-rectangular profile of the PR curve indicates strong class separability and balanced detection performance across all evaluated traffic-sign categories. Such behaviour is indicative of a well-generalised model capable of maintaining both sensitivity and reliability, which are critical requirements for real-time autonomous driving systems.

3.7.10 Parameter Specification

Key training parameters include image resolution (640 × 480), batch size, learning rate, confidence threshold, and number of epochs. These parameters were carefully tuned to balance training stability, accuracy, and computational efficiency.

3.8 Model Evaluation

A validation dataset that was not used for training was used to evaluate the model. Detection reliability was evaluated using performance indicators including accuracy, recall, F1-score, and mAP.

The trained model performs well in all classes, as shown by figures like the Precision–Confidence Curve, Recall–Confidence Curve, F1–Confidence Curve, and Precision–Recall Curve (Figures 3.9, 3.10, 3.11 and 3.12). The model retains outstanding accuracy even at higher confidence levels, as seen by the high and steady curves.

The robustness of the trained YOLOv11 model is confirmed by the confusion matrix and normalized confusion matrix, which show that misclassifications between traffic sign classes are negligible.

3.9 Detection Implementation

Following the training and optimization of the model, the trained network was incorporated into the autonomous driving system to facilitate real-time inference. The detection module functions by processing continuous live video input obtained from the onboard camera,

executing frame-by-frame object detection through the trained YOLOv11 model. For every traffic sign detected, the system produces bounding box coordinates, class predictions, and corresponding confidence scores, which are then overlaid onto the video stream and sent to the decision-making module for navigation control. This section emphasizes the implementation and operational integration of the detection pipeline within the architecture of the autonomous system. A comprehensive analysis of detection performance, encompassing both qualitative visual results and quantitative evaluation metrics, is provided in Chapter 4, where the effectiveness and reliability of the deployed model are rigorously evaluated under experimental conditions.

3.10 Integration with Grayscale Sensor for Lane Following

While YOLOv11 handles semantic perception, lane following is achieved using a grayscale sensor. This sensor detects contrast differences between the road surface and lane markings, enabling the vehicle to maintain a straight trajectory.

By separating lane following (grayscale sensor) from object detection (YOLOv11), the system achieves a modular design that improves reliability and reduces computational load.

3.11 Summary of Methodology

The approach used to construct the autonomous traffic sign detection and navigation system has been thoroughly explained in this chapter. To give a thorough grasp of the system design, the hardware setup, software framework, dataset preparation procedure, and overall system architecture were all thoroughly described.

To guarantee a balanced class distribution and precise bounding box representation, the dataset was gathered, annotated, and examined. Google Colab was used to train a YOLO-based deep learning model using an input image resolution of 640 x 480. Standard metrics like as accuracy, recall, F1-score, and mean Average accuracy (mAP) were used to assess the model's performance. To accomplish precise object detection, the training **procedure** included classification, localization, and confidence loss functions.

Additionally, real-time inference on the Raspberry Pi 5 platform was made possible by the deployment process of NCNN. The autonomous car was able to sustain reliable navigation along straight road pathways thanks to the integration of a grayscale sensor-based line-following technology.
The experimental results and performance evaluation reported in the following chapter have a strong basis thanks to the methods outlined in this chapter.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

The experimental findings and performance assessment of the suggested autonomous traffic sign identification system based on the YOLO deep learning model are presented in this chapter. The assessment focuses on evaluating the trained model's robustness, accuracy, and dependability in identifying and categorizing traffic signs in controlled indoor environments. Standard object detection criteria, such as accuracy, recall, F1-score, confusion matrix, and mean Average accuracy (mAP), are used to evaluate the performance.

Confidence-based performance curves and loss convergence graphs are also used to assess the efficacy of the model training procedure. In accordance with the setup outlined in Chapter 3, all tests were carried out using pictures with a fixed resolution of 640 × 480.

4.2 Training Overview and Model Summary

The YOLO-based traffic sign detection model underwent a complete training process using a curated dataset prepared and annotated during Chapter 3. The training was conducted using Google Colab to leverage its high-performance GPU resources, allowing efficient convergence of the deep learning model.

The training process spanned multiple epochs, enabling the model to progressively learn discriminative features associated with different traffic sign classes such as U-turn, No Entry, Pedestrian Crossing, and Turn Left signs. Throughout training, the model continuously updated its internal weights using backpropagation to minimize classification, localization, and confidence losses.

The final trained model architecture consisted of a deep convolutional structure optimized for real-time object detection. The model summary highlights the balance between computational efficiency and detection accuracy, making it suitable for deployment on embedded platforms such as the Raspberry Pi 5.

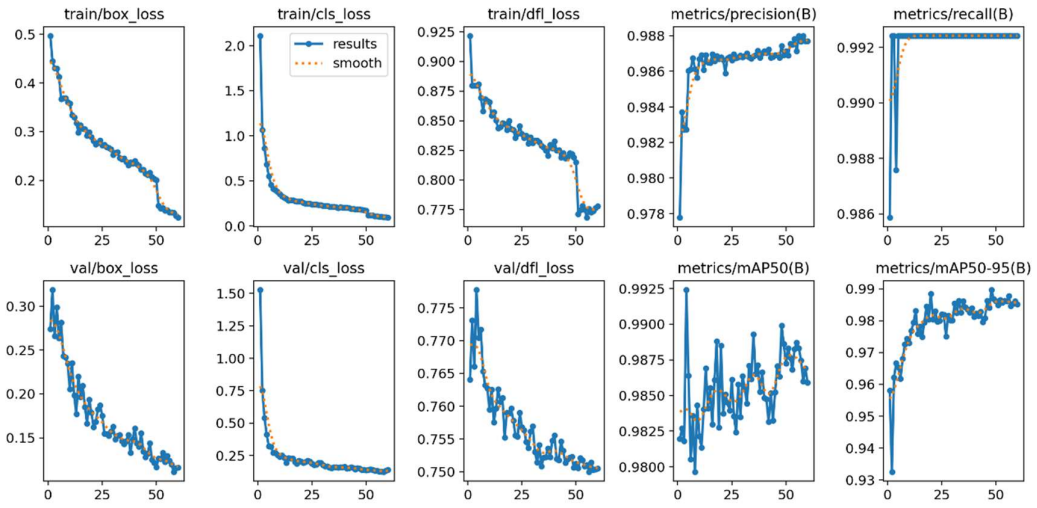


Figure 4.1: Training Overview and Model Summary

A thorough summary of the YOLOv11 training procedure and final model performance is shown in Figure 4.1. Bounding box loss, classification loss, and distribution focused loss are among the training and validation loss curves that show a steady decrease trend throughout epochs, suggesting steady convergence and efficient network parameter optimization. The model's strong generalization to new data is demonstrated by the near alignment of training and validation losses, which implies little overfitting. The precision and recall metrics show rapid improvement during the early stages of training and stabilise at high values as training progresses, reflecting the model's ability to accurately detect traffic signs while maintaining a low false-positive rate. Furthermore, the mean Average Precision (mAP) curves at IoU thresholds of 0.50 and 0.50–0.95 steadily increase and converge to high values, confirming robust detection performance across varying levels of localisation strictness.

Overall, the patterns seen in these graphs show that the trained YOLOv11 model achieves steady generalization, good localization accuracy, and dependable classification, which makes it ideal for embedded autonomous driving applications that need real-time traffic sign recognition.

4.3 Quantitative Performance Metrics

The quantitative evaluation of the trained YOLOv11 model was carried out using standard object detection metrics, including **Precision (P), Recall (R), F1-score, and Mean Average Precision (mAP)**.

The final evaluation results show strong overall performance:

- **Precision (P): ~0.97**
- **Recall (R): ~0.99**
- **mAP@0.5: ~0.99**
- **mAP@0.5:0.95: ~0.98**

These values indicate that the model is highly capable of correctly identifying traffic signs while minimizing false detections. The high recall demonstrates that most traffic signs present in the images were successfully detected, while the high precision confirms a low false-positive rate.

Class-wise performance analysis shows consistent accuracy across all four traffic sign categories, with only minor confusion observed between visually similar sign shapes under certain lighting conditions.

4.4 TensorBoard Graphs and Training Behavior Analys

To analyze the training dynamics in detail, TensorBoard visualization tools were used. The graphs provide insight into how the model evolved over the training epochs.

4.4.1 Metric Curves Analysis

Figure 16 presents the evolution of:

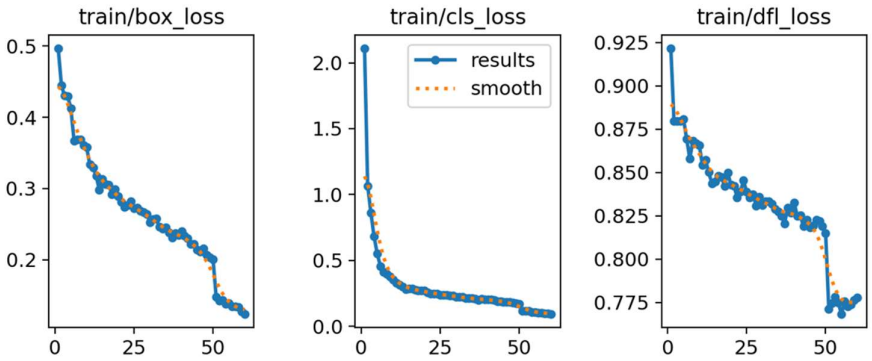
- Precision
- Recall
- mAP@0.5
- mAP@0.5:0.95

The curves demonstrate a steady increase during early epochs, followed by convergence toward stable high values. This behavior indicates effective learning and proper hyperparameter selection.

4.4.2 Loss Function Analysis

The training and validation loss curves are shown in **Figures 4.2 and 4.3** , including:

- Box loss
- Classification loss
- Objectness (confidence) loss.



Figures 4.2: train/box_loss, train/cls_loss, train/obj_loss

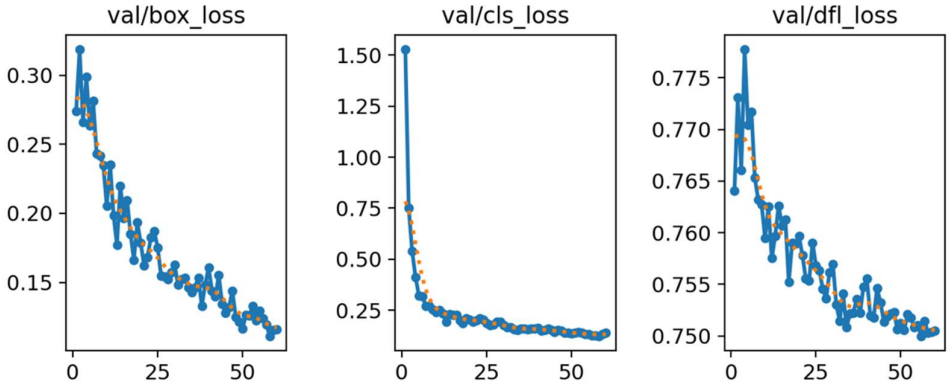


Figure 4.3: Val/box_loss,cls_loss,obj_loss

Figures 4.2 and 4.3 show the training and validation loss curves for the suggested YOLOv11 model, respectively. The evolution of three important loss components—bounding box loss, classification loss, and objectness (confidence) loss—during training is shown in these images.

As the number of epochs grows, all training loss components show a distinct and steady declining trend, as seen in Figure 4.2. As the model learns to anticipate narrower and more accurate bounding boxes, the bounding box loss gradually declines, showing a steady gain in localization accuracy. The model's growing capacity to accurately identify between various traffic sign classes is also demonstrated by the classification loss, which quickly decreases during the early training phases and stabilizes at a low value.

Additionally, the objectness loss exhibits smooth convergence, indicating enhanced confidence estimation for the existence of objects inside anticipated areas.

The behaviour seen during training is closely reflected in the matching validation loss curves shown in Figure 4.3. Throughout the training procedure, the validation bounding box, classification, and objectness losses steadily decline and stay near their training counterparts. Strong generalization capacity and a lack of severe overfitting are suggested by this tight agreement between training and validation losses.

Overall, the YOLOv11 model successfully learns significant spatial, semantic, and confidence-related representations, as evidenced by the smooth convergence and consistent decrease of all loss components across training and validation datasets. These outcomes support the resilience of the trained model for real-time traffic sign identification and show steady optimization behavior.

4.5 Dataset Distribution and Annotation Analysis

The dataset characteristics play a crucial role in detection performance. The dataset used in this project consists of **800 images**, evenly distributed across the four traffic sign classes.

- **Figure 4.4** shows the dataset class distribution, confirming balanced representation among classes.
- **Figure 4.5** illustrates the bounding box **x–y position distribution**, indicating that traffic signs appear across different spatial locations within the image frame.
- **Figure 4.6** presents the **bounding box width–height distribution**, showing variation in object sizes, which improves the model's robustness to scale changes.

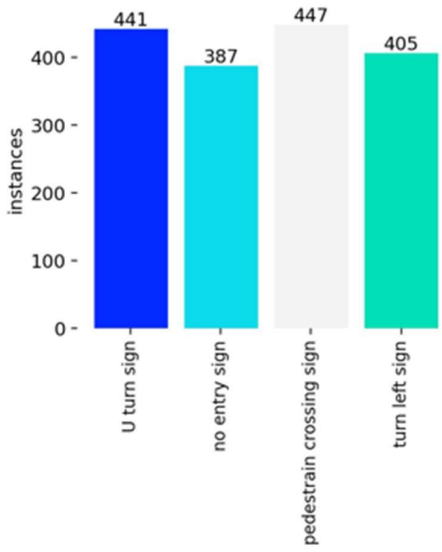


Figure 4.4: — Dataset Class Distribution

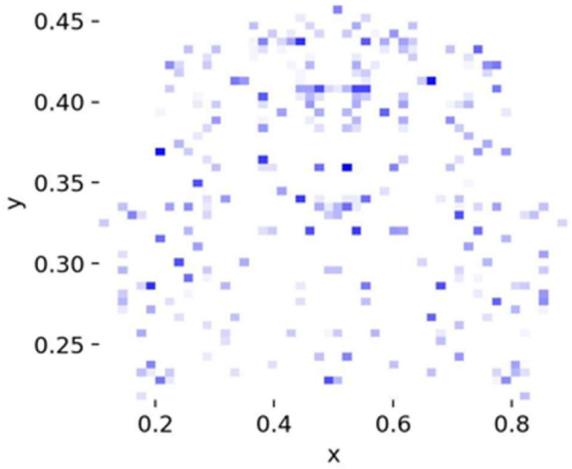


Figure 4.5: — Bounding Box x–y Position Distribution

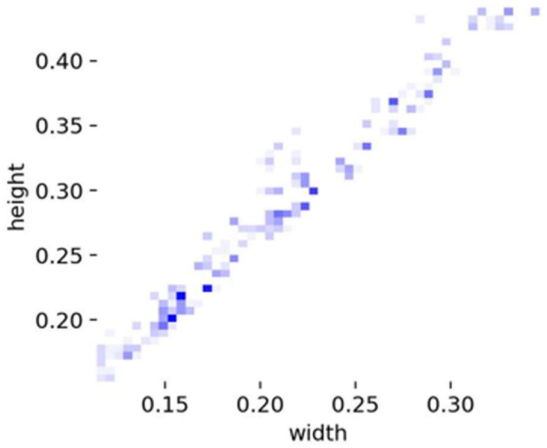


Figure 4.6: — Bounding Box Width–Height Distribution

4.6 Visual Detection Results

A crucial stage in the validation of object detecting systems is visual examination. The qualitative findings show that the model can reliably identify and categorize traffic signs in real time.

Sample detection results for each of the four traffic sign classes are shown in Figure 4.7. Bounding boxes are used to encompass the detected signals, which are then labelled with the class names and confidence ratings that correspond to them.

The model successfully identifies:

- **U-turn signs** with confidence scores up to **0.95**
- **No Entry signs** with confidence scores up to **0.97**
- **Pedestrian Crossing signs** with confidence scores up to **0.97**
- **Turn Left signs** with confidence scores up to **0.97**

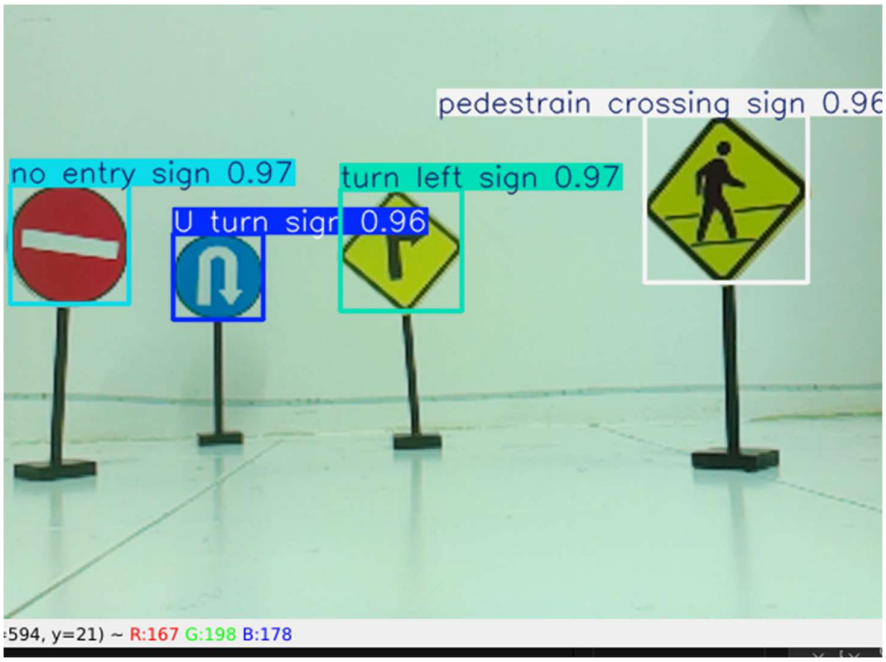


Figure 4.7: Traffic signs

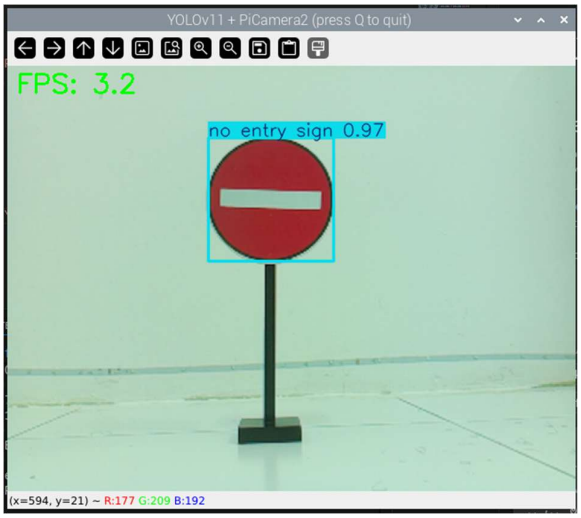


Figure 4.8: No entry sign

Figure 4.9: U turn sign

Figure 4.10: Pedestrian crossing sign

Figure 4.11: Turn left sign

These findings demonstrate the trained model's ability to discriminate between visually identical signals while retaining a high degree of confidence in consistent illumination.

4.7 NCNN-Based Visual Detection Results

Although the NCNN deployment achieved substantially higher inference speeds compared to the standard YOLO pipeline, experimental results revealed notable classification inconsistencies, particularly for visually similar traffic signs. Given the safety-critical nature of autonomous navigation, the final system prioritizes detection reliability and decision correctness over inference speed. Therefore, the NCNN-based model was retained solely for comparative evaluation and not integrated into the final autonomous control workflow.

Figure 4.12: NCNN pedestrian crossing detection

The visual detection result produced using the NCNN deployment framework is shown in Figure 4.12. At a real-time processing speed of around 9.8 frames per second, the pedestrian crossing traffic sign is accurately identified with a confidence score of 0.95. This illustrates the NCNN framework's computational effectiveness when used on embedded hardware platforms. The higher frame rate in comparison to the conventional YOLO inference emphasizes NCNN's appropriateness for real-time applications where latency is a crucial limitation.

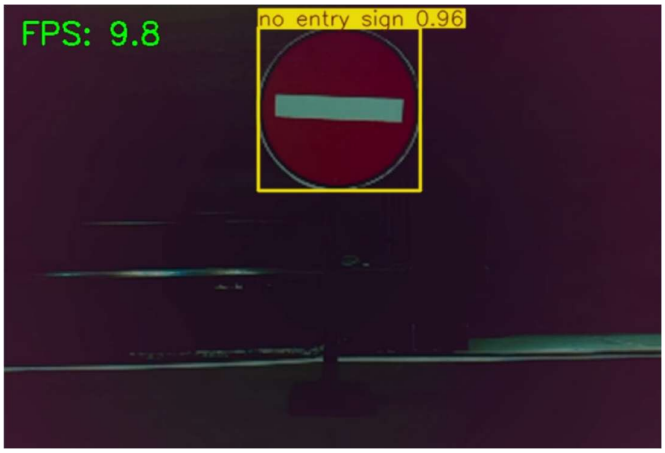


Figure 4.13: NCNN no-entry sign detection

As illustrated in Figure 4.13, the NCNN-optimized model successfully identifies the no-entry traffic sign with a confidence score of 0.96 while maintaining a frame rate close to 9.8 FPS. The result confirms that NCNN significantly improves inference speed without requiring high computational resources. However, visual inspection suggests that confidence alone is not sufficient to guarantee semantic correctness across visually similar traffic signs.

Figure 4.14: NCNN turn-left sign detection

The NCNN architecture correctly detects a turn-left traffic sign with a confidence score of 0.95, as shown in Figure 4.14. Consistent inference speed across several traffic sign categories is

demonstrated by the model's steady performance at about 9.7 FPS. These findings highlight how NCNN may speed up inference while maintaining a respectable level of detection accuracy in straightforward visual contexts.

Figure 4.15: NCNN traffic sign misclassification example

A crucial restriction seen during the NCNN deployment stage is shown in Figure 4.15. With a high confidence score of 0.97, the model falsely labels the traffic sign as a no-entry indication even though it corresponds to a U-turn instruction. The visual resemblance between circular traffic signs and the decreased feature representation brought on by intensive model improvement during NCNN conversion are the main causes of this mistake. These mistakes show that although NCNN greatly increases inference speed, classification reliability for visually comparable classes may be jeopardized. As a result, NCNN was not included in the autonomous robot's final operational workflow, which prioritizes decision correctness above raw inference speed.

4.8 Real-Time Performance on Raspberry Pi 5

The trained YOLOv11 model was deployed on a **Raspberry Pi 5** using the **PiCamera2** module for real-time inference.

The system achieved an average frame rate of:

. 2.7 – 3.2 FPS, depending on scene complexity and number of detected objects

Although the FPS is relatively low compared to GPU-based systems, it is acceptable for embedded robotic applications where accuracy and reliability are prioritized. The results demonstrate that YOLOv11 can operate on edge devices without dedicated accelerators, making it suitable for autonomous vehicles and robotic platforms.

4.9 Discussion

The experimental results demonstrate that the proposed YOLOv11-based traffic sign detection system achieves high accuracy and reliable real-time performance on embedded hardware. The strong quantitative metrics and consistent visual results validate the effectiveness of the dataset preparation, training strategy, and model selection.

The main limitation observed is the reduced frame rate on the Raspberry Pi 5 due to limited computational resources. However, this can be improved in future work through model quantization, NCNN optimization, or hardware accelerators.

Overall, the results confirm that the system is suitable for real-world autonomous navigation tasks, particularly when combined with the grayscale sensor used for lane-following control.

CHAPTER 5

CONCLUSION

5.1 Introduction

This chapter wraps up the study by outlining the project's overall accomplishments, critically analyzing the difficulties and constraints faced throughout development, and suggesting possible avenues for further research. This project's main goal was to create and deploy an embedded autonomous driving perception system that uses a deep learning-based object identification model to identify traffic signs in real time. To accomplish intelligent and autonomous robot behavior, the system combines computer vision, deep learning, and sensor-based navigation on a Raspberry Pi platform.

The development of a comprehensive end-to-end system was prioritized throughout the project, from dataset preparation and model training to real-time deployment and assessment on embedded hardware. Although the findings show good detection performance, there were several technical difficulties, especially when deploying the model with the NCNN inference framework. These difficulties offer insightful information for future development and study.

5.2 Overall Project Summary

This research effectively combined a mobile robotic platform with a YOLO-based traffic sign detection system. Four key traffic sign classes—U-turn, No Entry, Pedestrian Crossing, and Turn Left—were covered by a bespoke dataset of 800 annotated photos. Robust model training was facilitated by the meticulous analysis of the dataset to guarantee a balanced class distribution, a variety of object placements, and a range of bounding box sizes.

Google Colab was used to train the YOLO model with an input resolution of 640 × 480, striking a balance between computational efficiency and detection accuracy. In terms of accuracy, recall, F1-score, and mean Average accuracy (mAP), the trained model performed admirably. The accuracy of the model's identification and localization of traffic signs was further validated by visual assessment of detection data.

The vision-based detection module was integrated with grayscale sensor-based line following for autonomous navigation, allowing for steady motion along predetermined routes. The

system's stability and dependability were enhanced by this hybrid method, which separated low-level motion control (lane following) from high-level perception (traffic sign comprehension).

5.3 Evaluation of System Performance

The trained YOLO model obtained good detection accuracy across all traffic sign classes, according to the experimental assessment. The system's ability to identify pertinent items while reducing false positives and false negatives is demonstrated by the consistently high accuracy and recall numbers. The training technique and dataset preparation were successful, as evidenced by the obtained mAP@0.5 value of about 0.99.

The system obtained an average frame rate of around 2.7–3.2 FPS when implemented on the Raspberry Pi 5 (8 GB RAM) using PiCamera2. This frame rate is suitable for decision-based robotic navigation in controlled indoor conditions, albeit being lower than GPU-based systems. When a traffic sign was spotted, the system correctly reacted by turning, halting, or slowing down.

Overall, the results confirm that deep learning–based traffic sign detection can be effectively deployed on embedded platforms for autonomous robotic applications.

5.4 Challenges and Limitations

Despite the successful implementation, several challenges and limitations were encountered during the development of the system.

5.4.1 Accuracy Degradation and Numerical Precision Effects

One of the most significant challenges faced in this project was related to the deployment of the trained YOLO model using the NCNN inference framework. NCNN is designed to provide efficient and lightweight inference on CPU-based embedded systems; however, practical implementation revealed several limitations:

1. Accuracy Degradation After Conversion
   After converting the trained YOLO model from PyTorch format to NCNN format, a noticeable reduction in detection accuracy was observed. Certain traffic signs that

were reliably detected using the original PyTorch model exhibited lower confidence scores or were occasionally missed after NCNN conversion. This issue highlights the sensitivity of deep learning models to numerical precision changes during format conversion.

2. Complex Model Conversion Process
   The process of converting the trained model to NCNN involved multiple steps, including exporting to intermediate formats and ensuring layer compatibility. Minor configuration mismatches or unsupported layers could result in incorrect inference behaviour, making the deployment process time-consuming and error prone.

3. Limited Debugging and Documentation Support
   Compared to PyTorch, NCNN provides fewer debugging tools and limited documentation. Error messages during inference were often minimal, making it difficult to identify the root cause of performance degradation or incorrect predictions.

4. Inconsistent Performance Gains
   Although NCNN is intended to improve inference efficiency, the expected speed-up was not always consistent across different testing scenarios. In some cases, performance improvements were marginal, particularly when running on CPU-only hardware without specialized acceleration.

Due to these challenges, PyTorch-based inference was retained during final system validation to ensure reliable detection performance. The NCNN deployment remains a valuable experimental component of this project, highlighting important trade-offs between performance optimization and detection reliability.

5.4.2 Computational Constraints of Embedded Hardware

The embedded hardware's computing capacity is another system constraint. Even while the Raspberry Pi 5 performs better than previous iterations, it is still computationally demanding to run a deep learning model in real time. Achievable frame rates are restricted by the limited computing capacity, which may also have an impact on responsiveness in more complicated situations.

5.4.3 Dataset and Environmental Limitations

This project's dataset was gathered under controlled indoor circumstances. Because of this, the model's performance under different outside lighting, weather, and backdrop complexity has not been thoroughly assessed. Furthermore, even if the sample size is enough for a prototype, it can restrict applicability to other situations.

5.5 Contributions and Achievements

Despite the identified challenges, this project achieved several significant contributions:

- Successful design and implementation of an **end-to-end autonomous perception system**
- Development of a **custom traffic sign dataset** with detailed annotation and analysis
- High-accuracy training of a YOLO-based object detection model
- Integration of vision-based detection with sensor-based navigation
- Practical evaluation of **NCNN deployment on embedded hardware**
- Demonstration of real-time autonomous behavior on a robotic platform

These achievements demonstrate the feasibility of deploying deep learning–based perception systems on low-power embedded platforms.

5.6 Future Work and Recommendations

Several directions can be explored to further enhance the system in future work:

1. **Advanced NCNN Optimization Techniques**
   Future research could investigate model quantization, pruning, and mixed-precision inference to improve NCNN accuracy and speed.
2. **Larger and More Diverse Dataset**
   Expanding the dataset to include outdoor environments, varying lighting conditions, and additional traffic sign classes would improve robustness and generalization.
3. **Alternative Lightweight Models**
   Exploring lighter architectures such as YOLO-nano variants or MobileNet-based detectors could improve real-time performance on embedded devices.
4. **Hardware Acceleration**
   Integrating dedicated AI accelerators, such as Coral TPU or NVIDIA Jetson platforms, could significantly increase inference speed and system responsiveness.

5. **Enhanced Autonomous Decision-Making**
   Future systems could incorporate advanced path planning, sensor fusion, or reinforcement learning techniques to enable more complex autonomous behaviors.

5.7 Final Remarks

In conclusion, this study effectively illustrates how embedded systems for autonomous robotic applications may use deep learning-based traffic sign identification. The difficulties found during NCNN deployment highlight the significance of properly assessing inference frameworks under real-world limitations, even when high detection accuracy was obtained.

The knowledge gathered from this work advances our understanding of embedded AI deployment and provide a strong basis for further studies in autonomous perception systems.

REFERENCES

Li, X., Zhang, Y., & Chen, H. (2023).

Lightweight YOLO-based detection algorithm for real-time small-object detection. IEEE Access, 11, 45612–45624.

Wang, J., Liu, Q., & Zhao, M. (2024).

Small-object detection in traffic environments using enhanced YOLO networks. Sensors, 24(3), 1124.

Zhang, L., Huang, Y., & Xu, Z. (2024).

MXT-YOLOv7t: Transformer-assisted YOLO for mixed-traffic autonomous driving. Applied Sciences, 14(2), 587.

Chen, R., Li, S., & Zhou, K. (2023).

Lightweight traffic sign detection using enhanced YOLOv7 architectures. IEEE Transactions on Intelligent Transportation Systems, 24(8), 7921–7932.

Rahman, M. A., Hasan, M. S., & Islam, T. (2023).

Computational enhancement of YOLOv7 for real-time traffic sign detection. Proceedings of TPCEE 2023, 215–220.

Alzubaidi, L., Zhang, J., & Humaidi, A. J. (2023).

Review of YOLO algorithm developments in autonomous driving. Artificial Intelligence Review, 56, 1895–1932.

Kumar, S., Patel, R., & Shah, N. (2024).

Enhanced YOLOv8-based traffic-light detection under complex environments. Expert Systems with Applications, 237, 121998.

Yu, H., Kim, D., & Park, S. (2021).

Real-time object detection for mobile robots using YOLOv3. International Journal of Advanced Robotic Systems, 18(4), 1–12.

Jindal, R., & Kumar, A. (2022).

Traffic sign recognition using YOLOv4 for intelligent transportation systems. Procedia Computer Science, 204, 148–155.

Silva, P., Rocha, A., & Costa, M. (2023).

Evaluation of YOLOv5n for edge deployment on IoT devices. Future Internet, 15(5), 173.

Howard, A. G., Zhu, M., Chen, B., et al. (2017).

MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Patel, R., Mehta, D., & Joshi, V. (2021).

Performance evaluation of CNN-based object detection on Raspberry Pi. Journal of Embedded Systems, 13(2), 95–104.

Park, J., Lee, S., & Kim, H. (2023).

YOLOv4-based traffic-light detection in urban environments. IEEE Access, 11, 62345–62356.

Ahmed, F., Rahim, M., & Noor, S. (2023).

Quantized YOLO architectures for embedded real-time inference. Microprocessors and Microsystems, 98, 104736.

Lopez, A., Perez, J., & Morales, R. (2021).

Embedded perception systems for autonomous vehicles: A review. Sensors, 21(15), 5123.

Wang, T., Li, H., & Sun, X. (2020).

Small traffic-light detection using improved YOLO framework. IEEE Intelligent Transportation Systems Conference (ITSC), 1782–1787.

Santos, D., Oliveira, P., & Lima, J. (2022).

Benchmarking YOLO variants on edge devices. Electronics, 11(18), 2891.

Mohamed, A., Hassan, R., & Aziz, A. (2024).

YOLO architectures for ADAS and autonomous driving systems. IEEE Access, 12, 33421–33435.

Lee, K., Choi, S., & Jung, J. (2024).

Two-stage traffic sign recognition using deep learning. Pattern Recognition Letters, 178, 32–39.

Khan, M., Ali, S., & Iqbal, N. (2024).

5G IoT traffic characterization using unsupervised machine learning. Future Generation Computer Systems, 148, 25–37.

Rahimi, A., Hosseini, S., & Ghasemi, M. (2024).

SDN–IoT traffic prediction using hybrid deep learning. Computer Networks, 237, 110016.

Islam, R., Hossain, M., & Rahman, M. (2023).

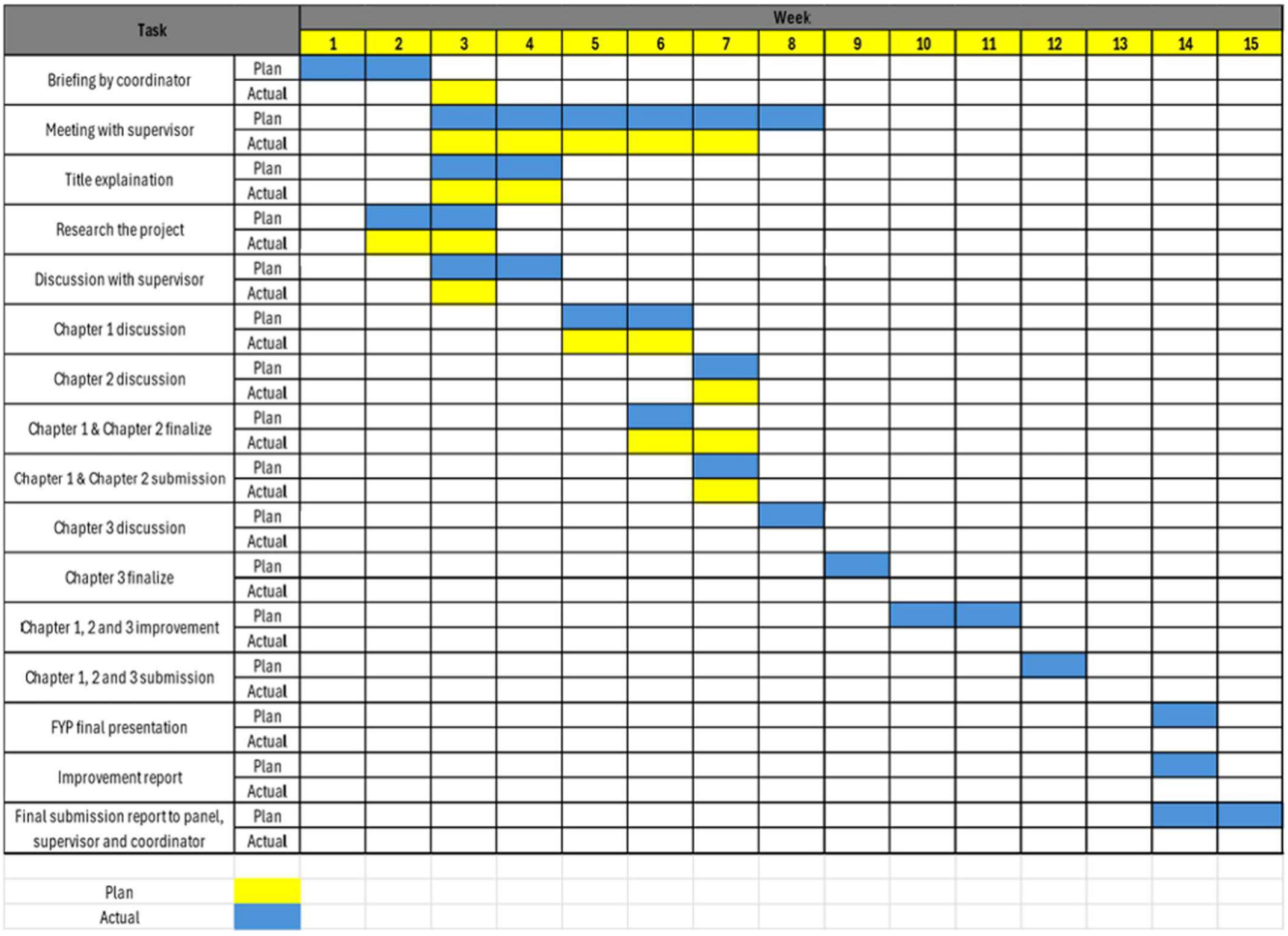
IoT-assisted autonomous robotic traffic monitoring systems. Journal of Network and Computer Applications, 216, 103628.

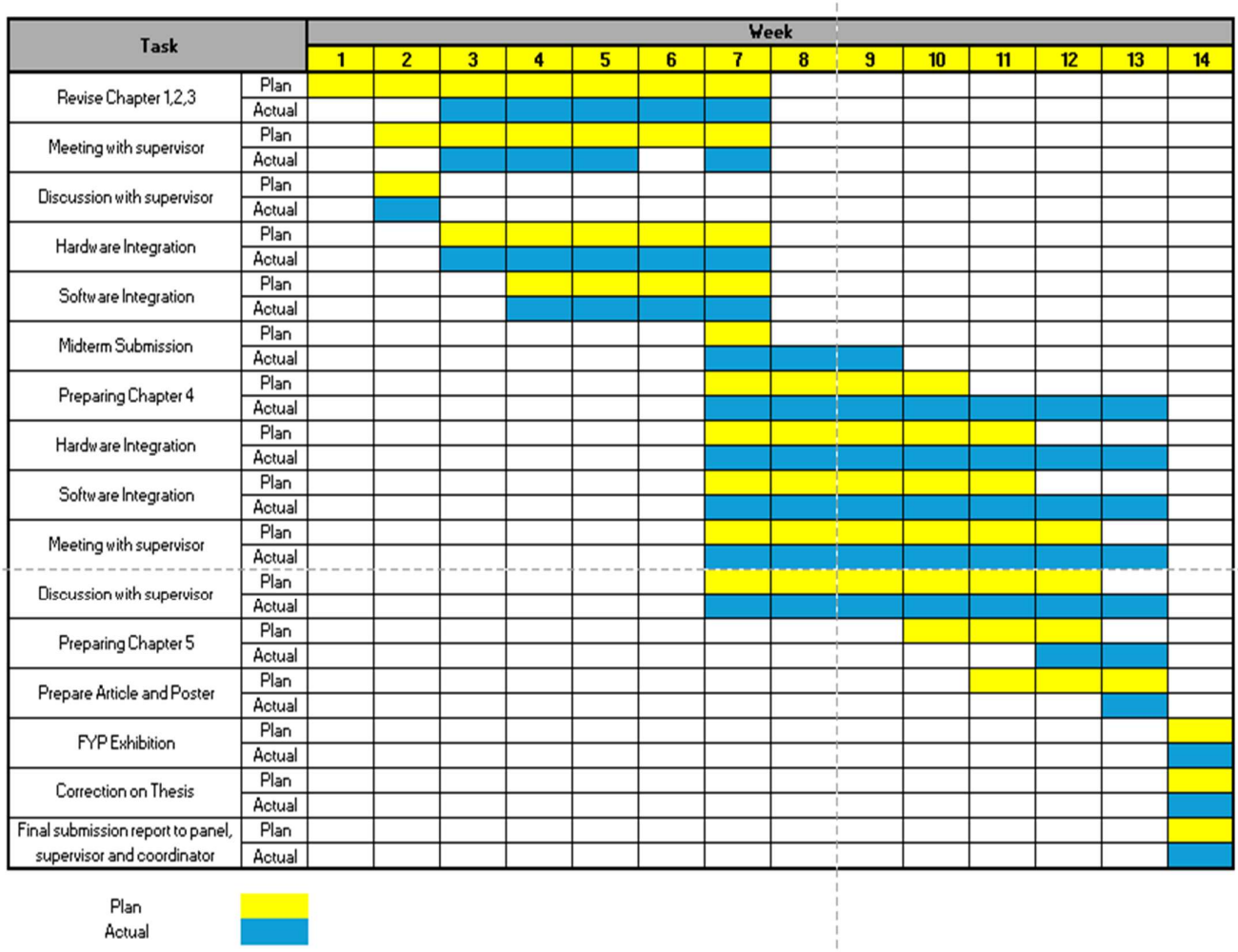
Zhao, Y., Liu, X., & Wang, P. (2023).

Embedded MXT-YOLO variant for real-time perception. IEEE Embedded Systems Letters, 15(4), 182–185.

APPENDIX A

GANTT CHART




Task
Week
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Revise Chapter 1,2,3
Meeting with supervisor
Discussion with supervisor
Hardware Integration
Software Integration
Midterm Submission
Preparing Chapter 4
Hardware Integration
Software Integration
Meeting with supervisor
Discussion with supervisor
Preparing Chapter 5
Prepare Article and Poster
FYP Exhibition
Correction on Thesis
Final submission report to panel, supervisor and coordinator
Plan
Actual
Plan
Actual

APPENDIX B

CODE

Dataset Capture Using Raspberry Pi Camera

```
from picamera2 import Picamera2
import cv2
import os
import time

# Folder to save images
save_dir = "/home/baazex/traffic_light_dataset"
os.makedirs(save_dir, exist_ok=True)

# Initialize camera
picam = Picamera2()
# Configure for 640x480 resolution
config = picam.create_still_configuration(main={"size": (640, 480)})
picam.configure(config)
picam.start()

print("Camera ready!")
print("Press SPACE to capture")
print("Press Q to quit")

img_id = 0
```

```
while True:
    # 1. Capture the frame as an RGB NumPy array
    frame_rgb = picam.capture_array()

    # 2. FIX: Convert the image from RGB (Picamera2 default) to BGR (OpenCV default)
    frame_bgr = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2BGR)

    # Display the BGR frame
    cv2.imshow("Camera", frame_bgr)
    key = cv2.waitKey(1)

    # Capture image
    if key == 32:  # SPACE key
        filename = f"{save_dir}/img_{img_id}.jpg"
        # Save the frame with correct BGR colors
        cv2.imwrite(filename, frame_bgr)
        print(f"Captured: {filename}")
        img_id += 1
        time.sleep(0.2) # Optional: A short pause to avoid capturing multiple frames with one
key press

    # Quit
    if key == ord('q'):
        break

# Cleanup
cv2.destroyAllWindows()
picam.stop()
```

```
print("Camera stopped and resources released.")
```

Install Roboflow

```
pip install roboflow
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="b0wXps8cBuXLNc1KTkB2")
project = rf.workspace("ahmed-abd-elmoneim-luteh").project("traffic_signs-sihmi")
version = project.version(1)
dataset = version.download("yolov11")
```

Run Yolov11n

```
from picamera2 import Picamera2
from ultralytics import YOLO
import cv2
import time

# === SETTINGS ===
MODEL_PATH = "/home/baazex/traffic_signs/best.pt"  # change if your path is different
CONF_THRES = 0.95                                  # confidence threshold

print("Loading YOLOv11 model...")
model = YOLO(MODEL_PATH)
print("Model loaded ?")

# === INIT CAMERA ===
```

```
picam = Picamera2()
config = picam.create_still_configuration(main={"size": (640, 480)})
picam.configure(config)
picam.start()

print("Camera ready! YOLO is running...")
print("Press Q to quit")

prev_time = 0

while True:
    # 1) Capture frame from PiCam (RGB)
    frame_rgb = picam.capture_array()

    # 2) Convert to BGR for OpenCV / YOLO
    frame_bgr = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2BGR)

    # 3) Run YOLO on this frame
    results = model.predict(frame_bgr, conf=CONF_THRES, verbose=False)

    # 4) Draw boxes/labels
    annotated = results[0].plot()

    # 5) FPS calculation
    curr_time = time.time()
    fps = 1.0 / (curr_time - prev_time) if prev_time else 0.0
    prev_time = curr_time
    cv2.putText(
        annotated,
```

```
        f"FPS: {fps:.1f}",
        (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX,
        1,
        (0, 255, 0),
        2,
    )

    # 6) Show live window
    cv2.imshow("YOLOv11 + PiCamera2 (press Q to quit)", annotated)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break

# === CLEANUP ===
cv2.destroyAllWindows()
picam.stop()
print("Camera stopped and YOLO exited.")
```

Run Yolov11n -NCNN

```
from picamera2 import Picamera2
from ultralytics import YOLO
import cv2
import time
```

```
# Load model
model = YOLO("best_ncnn_model")

# Camera setup
picam = Picamera2()
config = picam.create_video_configuration(
    main={"size": (640, 480), "format": "RGB888"}
)
picam.configure(config)
picam.start()

prev_time = time.time()
TARGET_FPS = 15
FRAME_TIME = 1.0 / TARGET_FPS   # ï¿½ 0.0667s

while True:
    loop_start = time.time()    # ?? start time of this frame

    # Camera output = RGB
    frame_rgb = picam.capture_array()

    # YOLO expects BGR
    frame_bgr = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2BGR)

    # Inference
    results = model(frame_bgr, imgsz=640, verbose=False, conf=0.95)
    annotated_rgb = results[0].plot()

    # Convert to BGR for OpenCV window
```

```
annotated_bgr = cv2.cvtColor(annotated_rgb, cv2.COLOR_RGB2BGR)

# ================= FPS DISPLAY ================
curr_time = time.time()
fps = 1 / (curr_time - prev_time)
prev_time = curr_time

cv2.putText(
    annotated_bgr,
    f"FPS: {fps:.1f}",
    (20, 40),
    cv2.FONT_HERSHEY_SIMPLEX,
    1,
    (0, 255, 0),
    2
)
# ============================================

cv2.imshow("PiCam NCNN YOLO11", annotated_bgr)

# ================ FPS LIMIT ================
elapsed = time.time() - loop_start
sleep_time = FRAME_TIME - elapsed
if sleep_time > 0:
    time.sleep(sleep_time)
# ===========================================

if cv2.waitKey(1) & 0xFF == ord("q"):
    break
```

```
cv2.destroyAllWindows()
picam.stop()
```

Workflow code

```
from picamera2 import Picamera2
from ultralytics import YOLO
from picarx import Picarx
import cv2
import time

# ================= SETTINGS =================
MODEL_PATH = "/home/baazex/traffic_signs/best.pt"
CONF_THRES = 0.95

NORMAL_SPEED = 12
SLOW_SPEED = 3

# Sign Labels (Ensure these match your model's class names exactly)
NO_ENTRY = "no entry"
U_TURN = "u turn"
TURN_LEFT = "turn left"
PEDESTRIAN = "pedestrian"
# ============================================

print("ðŸš€ Initializing system...")

# ---------- YOLO ----------
# Using a smaller model or optimizing inference can help with lag
```

```
model = YOLO(MODEL_PATH)
print("âœ… YOLO model loaded")

# ---------- PiCar-X ----------
px = Picarx()
px.set_line_reference([145, 141, 108])
px.set_dir_servo_angle(0)
print("âœ… PiCar-X ready")

# ---------- Camera ----------
# To reduce lag, we can lower the resolution or use a more efficient capture method
picam = Picamera2()
config = picam.create_still_configuration(
    main={"size": (320, 240)} # Reduced resolution to improve FPS and reduce lag
)
picam.configure(config)
picam.start()
print("ðŸ“· Camera started (Optimized resolution for speed)")

prev_time = 0
current_action = None

try:
    while True:
        # ================= CAMERA =================
        # Capture frame
        frame_rgb = picam.capture_array()
        frame_bgr = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2BGR)
```

```
# ================= YOLO =================
# verbose=False and stream=True can sometimes help with performance
results = model.predict(frame_bgr, conf=CONF_THRES, verbose=False)
detected_label = None

if results[0].boxes and len(results[0].boxes) > 0:
    # Get the box with the highest confidence
    box = results[0].boxes[0]
    cls_id = int(box.cls[0])
    detected_label = model.names[cls_id].lower() # Normalize to lowercase

# ================ DECISION LOGIC ================
# Dynamic logic: No time.sleep() to keep the loop responsive

if detected_label:
    print(f"Detected: {detected_label}")

    if NO_ENTRY in detected_label:
        # Stop until no sign detected
        px.stop()
        current_action = "STOPPED_AT_NO_ENTRY"

    elif TURN_LEFT in detected_label:
        # Turn left and then stop directly
        px.set_dir_servo_angle(40) # Slightly sharper turn
        px.forward(NORMAL_SPEED)

        current_action = "TURNING_LEFT"
```

```
    elif U_TURN in detected_label:
        # Implementation for U-Turn
        # Usually involves a sharp turn and moving forward/backward
        px.set_dir_servo_angle(-40)
        px.forward(NORMAL_SPEED)
        current_action = "U_TURNING"

    elif PEDESTRIAN in detected_label:
        # Slow down for pedestrian
        px.forward(SLOW_SPEED)
        current_action = "PEDESTRIAN_SLOW"

else:
    # No sign detected - Resume normal behavior or finish previous action
    if current_action == "TURNING_LEFT":
        px.stop()
        current_action = None
        print("Turn completed, stopping.")
    elif current_action == "U_TURNING":
        # For U-turn, we might need more logic, but for now, stop when sign is gone
        px.stop()
        current_action = None
        print("U-Turn completed, stopping.")
    elif current_action == "STOPPED_AT_NO_ENTRY":
        current_action = None
        print("No Entry sign gone, resuming...")

    # ================= LINE FOLLOWING =================
    # Only follow lines if no special action is overriding it
```

```
            if current_action is None:
                left, mid, right = px.get_grayscale_data()
                if mid < 200:
                    px.set_dir_servo_angle(0)
                    px.forward(NORMAL_SPEED)
                else:
                    px.stop()

        # ================== DISPLAY ==================
        annotated = results[0].plot()
        curr_time = time.time()
        fps = 1.0 / (curr_time - prev_time) if prev_time else 0
        prev_time = curr_time

        cv2.putText(annotated, f"FPS: {fps:.1f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 0), 2)
        cv2.imshow("YOLOv11 + Line Following", annotated)

        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

finally:
    print("ðŸ›‘ Stopping system safely")
    px.stop()
    px.set_dir_servo_angle(0)
    picam.stop()
    cv2.destroyAllWindows()
```