

PALESTINE POLYTECHNIC UNIVERSITY



College of Information Technology and Computer Engineering

Department of Computer Engineering

Smart Traffic System

Project Team :

Samah Mebed 201012
Mohammad Taha 217065
Momen Qarajeh 201006

Supervisor :

Dr. Alaa Halawani

This project is submitted to fulfill the requirements for a Bachelor's degree in Computer Systems Engineering

June 2025

Certification and Anti-Plagiarism Declaration

This is to declare that the graduation project produced under the supervision of Dr. Alaa Halawani having the title “Smart Traffic System ” was prepared by students below in partial fulfilment of the requirements for the degree of Bachelor in Engineering of Computer System Engineering and no part hereof has been reproduced illegally (in particular: cut and paste) which can be considered as Plagiarism.

All referenced parts have been used to support and argue the idea and have been cited properly. We certify that we will not commit any plagiarism, cheating, or any other academic integrity violation. We will be responsible and liable for any consequence if violation of this declaration is proven.

Any use of Artificial Intelligence (AI) tools in the preparation of this project has been clearly declared and documented in the related appendix at the end of the report, in accordance with the PPU guidelines.

Date:

Graduation Project Group:

Samah Mebed Signature:

Mohammad Taha Signature:

Momen Qarajeh Signature:

Abstract

The rise in traffic congestion in cities is a big problem for managing traffic and keeping roads safe. The Smart Traffic System is an advanced traffic light solution that helps reduce traffic jams, cut down waiting times, and improve how traffic moves, while also making roads safer for both vehicles and pedestrians. The system integrates technologies such as computer vision, IoT, and machine learning. Employing a Raspberry Pi microcontroller with cameras and motion sensors, the system monitors traffic in real time. Cameras capture images to identify vehicles, and inbuilt camera microphones are employed to sense emergency vehicle sirens. Upon detecting an emergency vehicle, the system prioritizes it by turning traffic lights green instantly. Likewise, it varies signal timing according to pedestrian presence to provide safety during crossings. By processing data from input sources at once, the Smart Traffic System enhances traffic management at heavy-duty intersections, offering a flexible solution to congestion, minimizing accidents, and promoting road safety.

يُمثل ارتفاع الازدحام المروري في المدن مشكلة كبيرة لإدارة حركة المرور والحفاظ على سلامة الطرق. يُعد نظام المرور الذكي حلاً متطوراً لإشارات المرور، يُساعد في تقليل الاختناقات المرورية، وتقليص أوقات الانتظار، وتحسين حركة المرور، مع جعل الطرق أكثر أماناً للمركبات والمشاة على حدٍ سواء. يدمج النظام تقنيات مثل الرؤية الحاسوبية، وإنترنت الأشياء، والتعلم الآلي. يستخدم النظام متحكم Raspberry Pi مع كاميرات وأجهزة استشعار الحركة، ويراقب حركة المرور آنياً. تلتقط الكاميرات صوراً لتحديد المركبات، وتستخدم ميكروفونات الكاميرا المدمجة لاستشعار صفارات سيارات الطوارئ. عند اكتشاف مركبة طوارئ، يُعطي النظام الأولوية لها بتحويل إشارة المرور إلى اللون الأخضر فوراً. وبالمثل، يُغير النظام توقيت الإشارة وفقاً لوجود المشاة لضمان السلامة أثناء العبور. من خلال معالجة البيانات من مصادر الإدخال فوراً، يُعزز نظام المرور الذكي إدارة حركة المرور عند التقاطعات المرورية، مُقدماً حلاً مرناً للازدحام، ويقلل الحوادث، ويُعزز السلامة على الطرق.

Keywords: Smart Traffic System, Real-Time Traffic Management, Emergency Vehicle Detection, Pedestrian Safety, Computer Vision, Machine Learning, YOLOv8, YAMNet, Raspberry Pi, Traffic Congestion .

Acknowledgment

In the name of Allah, the Most Gracious and Most Merciful, who granted us the strength, knowledge, and perseverance to complete this project. We would like to express our sincere gratitude to all those who have supported and inspired us throughout this journey, our families, friends, and our supervisor.

We are especially thankful to our supervisor, Dr. Alaa Halawani, for his valuable guidance, insightful advice, and continuous support, which have been instrumental to the progress and success of this project.

We also extend our heartfelt appreciation to our families, whose constant encouragement, sacrifices, and belief in our potential have been our greatest source of motivation. We are equally grateful to our friends and colleagues for their collaboration, assistance, and moral support.

Finally, we would like to thank everyone who contributed to this project in any capacity, whether directly or indirectly. Every form of support be it technical help, thoughtful advice, or kind words played a meaningful role in helping us reach this achievement.

Contents:

Chapter 1

Introduction.....	8
1.1. Preface.....	8
1.2. Problem Statement.....	8
1.3. Aims & Objectives.....	9
1.4. System Requirements.....	9
1.4.1. Functional Requirements.....	9
1.4.2. Non-Functional Requirements.....	9
1.5. System Description.....	10
1.6. Limitations and Constraints.....	10
1.7. Schedule.....	11
1.8. Report Outline.....	11

Chapter 2

Background.....	12
2.1. Preface.....	12
2.2. Theoretical Background.....	12
2.2.1. Computer Vision.....	12
2.2.2. Machine Learning.....	12
2.2.3. Convolutional Neural Networks (CNNs).....	12
2.2.4. Internet of Things (IoT).....	13
2.2.5. Cloud Computing.....	13
2.2.6. YOLO Architecture.....	13
2.2.7. YAMNet Architecture.....	14
2.3. Literature Review.....	15
2.3.1. Intelligent Traffic Light Guidance System.....	15
2.3.2. Smart Traffic Lights Signals.....	15
2.3.3. Smart Traffic Management System.....	15
2.4. Summary.....	16

Chapter 3

System Design.....	17
3.1. Preface.....	17
3.2. System components and Design alternatives.....	17
3.2.1. Hardware Components.....	17
3.2.1.1. Controllers.....	17
3.2.1.2. Cameras.....	18
3.2.1.3. USB Camera Microphones.....	19
3.2.1.4. Momentary Push Button.....	19
3.2.1.5. Motion Sensor.....	20
3.2.2. Software Component.....	20
3.2.2.1. Object Detection and Image Processing.....	20
3.2.2.2. Emergency Vehicle Sound Detection.....	21
3.2.2.3. Cloud Storage.....	22
3.2.2.4. Image Acquisition and AI Model Integration.....	22
3.3. Conceptual System Description.....	23
3.4. Algorithms and Methodologies.....	25

3.5. Schematic Diagrams.....	27
3.6. Summary.....	27
Chapter 4.....	28
Testing and Evaluation.....	28
4.1. Preface.....	28
4.2. Hardware Setup and Evaluation.....	28
4.2.1. System Prototype.....	28
4.2.2. Raspberry Pi 5.....	29
4.2.3. USB Cameras.....	29
4.2.4. Pedestrian Request Unit.....	30
4.2.5. LED Traffic Lights.....	30
4.2.6. Power Supply and Wiring.....	31
4.3. Software Modules and Performance.....	31
4.3.1. Vehicle Detection Module.....	31
4.3.2. Emergency Vehicle Detection Module:.....	34
4.4. Implementation Challenges and Issues.....	36
4.4.1. Hardware Implementation Challenge.....	36
4.4.2. Software Implementation Challenges.....	36
4.5. Module Integration and Priority Logic Testing.....	37
4.6. Full System Evaluation Under Real Scenarios.....	38
4.7. Summary.....	41
Chapter 5.....	42
Conclusion.....	42
5.1. Conclusion.....	42
5.2. Future Work.....	43
References.....	44
Declaration and Documentation of AI Tools Usage in the Graduation Project.....	46

List of Figures

1.1. System Description.....	10
2.1. Convolutional Neural Network Architecture.....	13
2.2. YOLO architecture.....	14
3.1. Logitech camera.....	18
3.2. USB Camera Microphone	19
3.3. Momentary Push Button.....	20
3.4. PIR Sensor.....	21
3.5. YOLO object detection and recognition.....	22
3.6. System Block Diagram.....	25
3.7. System schematic diagram.....	28
4.1. Prototype setup.....	28
4.2. Raspberry Pi 5.....	29
4.3. USB camera.....	29
4.4. Camera-detected scene.....	29
4.5. Pedestrian push button.....	30
4.6. PIR motion sensor.....	30
4.7. Vehicle traffic light.....	30
4.8. Pedestrian traffic light.....	30
4.9. Power adapter.....	31

4.10. Wiring layout.....	31
4.11. YOLOv8n detection outputs.....	32
4.12. YOLOv8n detection result.....	32
4.13. Average confidence in different environmental conditions.....	34
4.14. YAMNet detection output.....	35
4.15. Terminal output during normal traffic mode.....	38
4.16. Pedestrian Mode Terminal Output.....	39
4.17. YAMNet terminal output during emergency detection.....	39
4.18. Terminal output showing system behavior during emergency mode.....	40
4.19. Terminal output during emergency override in pedestrian mode.....	40
4.20. Terminal output during system idle state.....	41

List of Tables

1.1. Project schedule in the first and the second semester.....	11
2.1. Comparison with other projects.....	16
3.1. Comparison between Raspberry Pi 5 and NVIDIA Jetson Nano.....	17
3.2. comparison between Logitech Camera and IP Camera (Reolink RLC-511W).....	18
3.3. External Sound vs USB Camera Microphone.....	19
3.4. Comparison between PIR and Ultrasonic sensors.....	20
3.5. Comparison between YOLO and Haar Cascade.....	21
3.6. Comparison between YAMNet and VGGish algorithm.....	22
3.7. Comparison between Firebase and Amazon Web Services.....	23
3.8. Comparison between OpenCv and PIL.....	23
4.1. Different conditions, actual number of cars detected numbers and average confidence.....	33
4.2. Comparison of Real-Time Image Processing Performance Between YOLOv5 and YOLOv8-nano.....	36

Abbreviations:

IoT Internet of Things. 13,23,24,43,44.

CNN Convolutional Neural Network. 12,13,44.

PIR Passive Infrared Sensor. 20,23,30,37,39,40,42.

YOLO You Only Look Once. 13,14,16,20,21,23,25,29,32,32,33,37,38,42,43,44,46.

VGGish Visual Geometry Group-inspired architecture for audio embedding. 21,22.

YAMNet Yet Another Mobile Network. 14,16,19,21,22,24,25,29,34,35,37,38,39,40,42,45,46.

ADC Analog-to-digital converter. 36.

USB Universal Serial Bus. 9,10,15,17,18,19,23,24,29,31,34,36,37,39.

LED Light-emitting diode. 30.

GPIO general-purpose input/output. 17,18,29,36.

CPU Central Processing Unit. 17,29,37.

ID Identification. 36,37.

Chapter 1

Introduction

1.1. Preface

The problem of traffic congestion in metropolitan areas has become one of the most dominant issues of the modern world. Clogged roads result in longer commuting hours, worsen air quality, increase the number of vehicular accidents, and endanger both the drivers and pedestrians.

The Smart Traffic System Project seeks to Improve pedestrian safety and the flow of traffic with the assistance of new technologies such as computer vision and motion sensors. The project's aim is to implement an automated control system of traffic signals which will use live feeds from surveillance cameras and motion detector sensors. This system is meant to give preference for emergency vehicles, balance the length of signals, and minimize wait times.

The project intends to develop an effective solution for dynamic traffic management which is designed to lessen traffic congestion, minimize accidents, and enhance road safety. The goal is also to improve the overall experience of maneuvering through roadways for motorists and pedestrians.

1.2. Problem Statement

New developments in traffic control technology have resulted in intelligent traffic lights, utilizing sensors and real-time processing to dynamically control traffic lights. These maximize emergency vehicle response by the traffic system and pedestrian safety.

Traditional traffic lights are inefficient, and can cause congestion, delay, and are a risk to safety. Traditional traffic lights respond too slowly to the changes in traffic, do not detect emergency services, and do not cater to the demands made by pedestrians, leading to longer waiting times.

Our aim is to develop a smarter traffic control system based on the latest technologies, including computer vision, to provide priority to emergency vehicles and enhance pedestrian safety. The system will maximize traffic flow, reduce congestion, and provide smoother conditions for all via the adaptive control of traffic signals based on real-time information.

1.3. Aims & Objectives

1. Enhance traffic management to reduce congestion.
 - a) Implement adaptive signal control to alter traffic flow in real time.
 - b) Coordinate signal times to achieve maximum traffic efficiency and minimum waiting time.
2. Enhance the response to emergency vehicles.
 - a) Use built-in microphones of USB cameras to detect emergency vehicle sirens and adjust signals accordingly.
 - b) Create smart algorithms that reduce waiting time for emergency vehicles without causing problems for other road users.
3. Enhance pedestrian safety at intersections.
 - a) Use motion sensors to spot pedestrians and adjust the traffic lights to help them cross safely.
 - b) Incorporate pedestrian push buttons to allow pedestrians to request green signals.

1.4. System Requirements

1.4.1. Functional Requirements

- The system will handle camera data to detect vehicles in real time and respond accordingly by adjusting the traffic lights.
- The system will detect emergency vehicles and will coordinate traffic signals to allow smooth movement.
- The system will detect pedestrians at intersections with the aid of motion sensors and push buttons and optimize signal timing for safe crossing.

1.4.2. Non-Functional Requirements

- Response Time: The system should detect emergency vehicles early enough to allow the traffic lights to adjust in time and give them a clear path before they reach the intersection.
- System Availability: The system should work all day without stopping to make sure traffic signals run smoothly.
- Data Accuracy: The system must collect and use traffic data correctly, with very few errors, to make the right decisions.
- Safety: The system should protect both people walking and drivers by changing the lights in a safe way to prevent accidents.

1.5. System Description

The main goal of this system is to make traffic move better, prioritize emergency vehicles, and protect pedestrians. The system works in three simple steps:

Stage 1: Cameras at intersections take pictures every few seconds. The system analyzes these pictures to count how many cars are waiting. Then, it changes the traffic lights based on how crowded the road is.

Stage 2: Microphones in USB cameras listen for sirens from emergency vehicles. When the system hears a siren, it changes the light to green to let the emergency vehicle go through quickly.

Stage 3: The system keeps pedestrians safe using a push button and motion sensors. When a person presses the button, the system turns the light green so they can cross. If it doesn't see any movement for 3 seconds, the light turns red again.

Figure 1.1 below shows the system description.

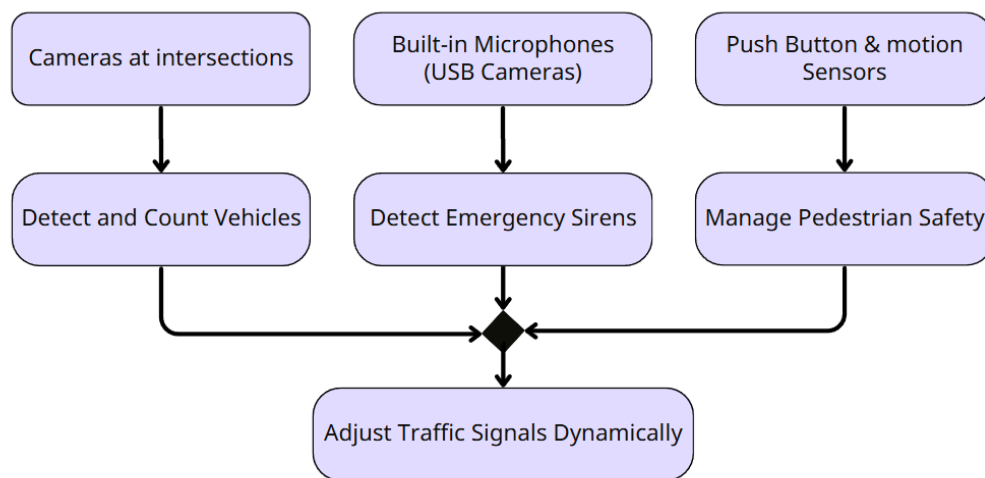


Figure 1.1: System Description.

1.6. Limitations and Constraints

- **Environmental Constraints:** Weather conditions like rain or fog may make it hard for the cameras to capture clear traffic data. This can affect how well the system works.
- **Technological Constraints:** The microphones in USB cameras might not work accurately in noisy areas, especially during busy hours when there's a lot of sound.

- **Scope Limitations:** This project is only a small simulation model. It will not be used in a real city. Because of time and limited resources, the system will be built by hand and tested in a small setup.
- **Operational Constraint :** The system can only respond to one emergency vehicle every 60 seconds. This helps the system stay stable and avoid too many changes to the normal traffic flow.

1.7. Schedule

The system implementation and operation tasks are distributed along the first and second semesters, summarised in Table 1.1.

Table 1.1: Project schedule in the first and the second semester.

	The first semester			The second semester			
Week	1- 4	5 - 10	10 - 15	1 - 5	6 - 9	10 - 14	15
Selection of project Idea							
Collecting the Data							
System Design							
System Implementation							
System testing							
system operation							
Documentation							

1.8. Report Outline

This report is organized as follows: Chapter 2 gives a short introduction to the major technologies and algorithms to be used in the project. It also includes a review that compares our project with other similar ones. Chapter 3 presents the project's design, covering both hardware and software parts. It discusses design choices, the conceptual background of the software, and presents a schematic diagram. Chapter 4 covers both the implementation and testing of the system. It explains how each component was assembled, how the system was tested, and discusses challenges we faced, chapter 5 concludes the results of the work and recommendations of improvements.

Chapter 2

Background

2.1. Preface

This chapter explains the basic theory needed for our project. After that, we review past studies and compare them with our project. This helps us show what is new and different in our work. In short, this chapter gives the background needed to understand where our project comes from and how it relates to other projects in the same field.

2.2. Theoretical Background

In this part, we describe the main parts of our project. Computer vision is used to understand inputs like images and videos. These inputs go into an algorithm that changes the timing of traffic lights based on what it sees. We also explain the software and hardware used in our system.

2.2.1. Computer Vision

Computer vision is a field of artificial intelligence that sometimes uses deep learning models to train computers and systems to understand images and visual signals [1]. Computer vision is widely used in applications that require classification of objects or situations.

2.2.2. Machine Learning

Machine learning is a branch of artificial Intelligence focused on building computer systems that learn from data. The breadth of machine learning techniques enables software applications to improve their performance over time [2].

2.2.3. Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN or ConvNet) is a deep learning algorithm specifically designed for any task where object recognition is crucial such as image classification, detection, and segmentation. Many real-life applications, such as self-driving cars, surveillance cameras, and more, use CNNs [3]. Their secret lies in a specialized set of layers. Convolutional layers act like filters that scan the image, extracting significant features like edges, lines, and shapes. Pooling layers then down sample the data, keeping the most important information while reducing complexity.

Finally, fully connected layers, similar to those in traditional neural networks, combine these extracted features to make a final prediction, such as classifying the image or recognizing objects within it as shown in Figure 2.1 [4].

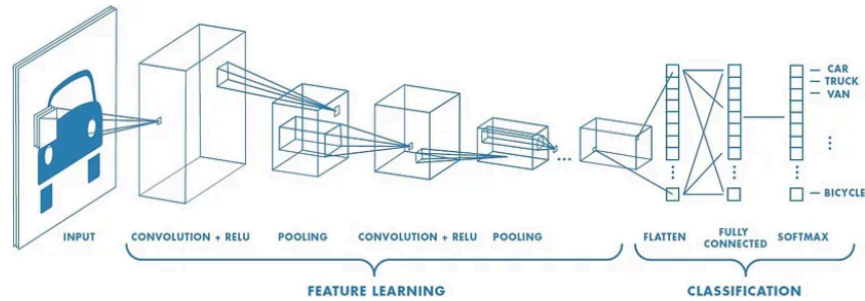


Figure 2.1: Convolutional Neural Network Architecture.

2.2.4. Internet of Things (IoT)

The Internet of Things refers to the physical items such as sensors, processing power, software, and other technologies that are linked to other systems and devices over the Internet or other communications networks and exchange data with them [5].

2.2.5. Cloud Computing

Cloud computing is the delivery of computing services including servers, storage, databases, networking, software, analytics, and intelligence over the internet (the cloud) to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping you lower your operating costs, run your infrastructure more efficiently, and scale as your business needs change [6].

2.2.6. YOLO Architecture

YOLO (You Only Look Once) is a real-time object detection algorithm that can detect and classify multiple objects in an image with high speed and accuracy. Unlike traditional methods that process regions of interest separately, YOLO treats the entire image as a single input and performs detection in one forward pass through the network [7].

The YOLO architecture is based on a convolutional neural network (CNN). The input image is divided into a grid, and for each grid cell, the model predicts bounding boxes and confidence scores for different object classes. YOLO uses convolutional layers for feature extraction, followed by fully connected layers that output the final object classifications and locations. Because it performs detection in a single step, YOLO is much faster than earlier methods like R-CNN, making it ideal for real-time applications such as smart traffic systems.

The architecture of YOLO is shown in Figure 2.2, which demonstrates how the model processes an input image through multiple convolutional and fully connected layers to produce object detections.

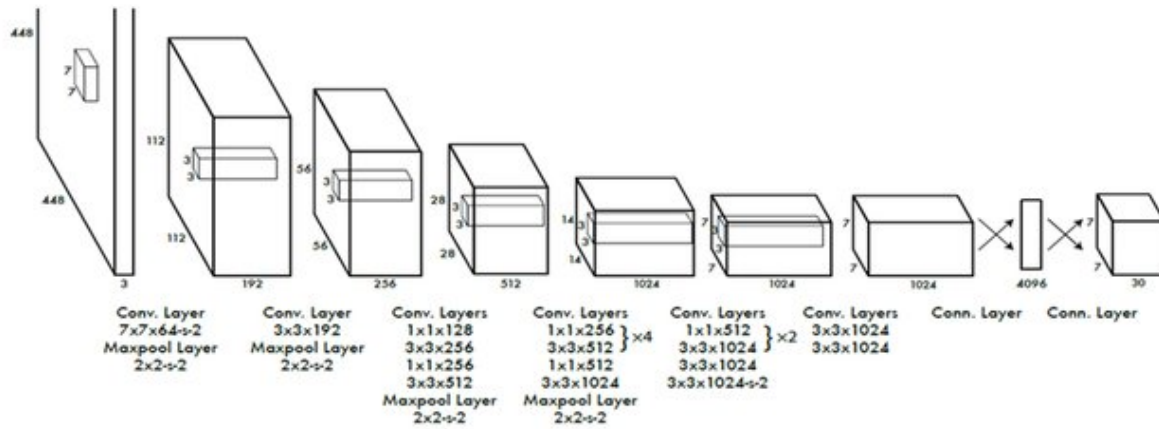


Figure 2.2: YOLO architecture.

2.2.7. YAMNet Architecture

YAMNet is a lightweight sound classification model developed by Google. It is trained on the AudioSet dataset and can recognize over 500 types of everyday sounds, including sirens, alarms, speech, and environmental noise. YAMNet is particularly useful for real-time sound detection on edge devices like the Raspberry Pi [8].

The architecture of YAMNet is based on MobileNet v1, which is a compact and efficient convolutional neural network designed for mobile and low-power devices. YAMNet takes short audio segments as input, converts them into mel-spectrograms (visual representations of sound), and then processes them through convolutional layers to extract relevant features. At the output layer, it provides class probabilities for all possible sound labels, along with confidence scores. Thanks to its lightweight design, YAMNet can run locally without an internet connection, making it ideal for smart traffic systems that need to detect emergency sirens and respond quickly.

2.3. Literature Review

Smart traffic signal systems have become a critical technology for improving urban traffic flow and ensuring road safety. Many studies and projects have proposed various designs to optimize traffic management and respond to changing conditions in real time. The following are discussions of such works.

2.3.1. Intelligent Traffic Light Guidance System

This system aims to improve traffic flow by directing drivers towards less congested roads. This system uses cameras to assess road conditions and provide real-time alerts to drivers via a display screen and a red light. The light turns on in case of congestion and the screen displays the congestion rate. The Smart Traffic System handles congestion by controlling the traffic light. In addition, it handles emergency situations and pedestrians in case of congestion [9].

2.3.2. Smart Traffic Lights Signals

In this project, algorithms were used to prioritize traffic lights based on the number of vehicles. They counted the number of vehicles based on the presence of two sensors, one at the beginning of the road and the other at its end (Proximity Sensor). There was communication between them via Bluetooth, and based on that, the traffic light was controlled [10]. But our system, the Smart Traffic System, uses real-time cameras and computer vision to monitor and analyze traffic, control traffic lights, pedestrian signals, and handle emergency situations.

2.3.3. Smart Traffic Management System

In this project, algorithms were used to dynamically control traffic lights based on vehicle detection at intersections. Infrared sensors were added at the intersections to detect the presence of vehicles. and the Arduino Uno used this information to change how long each light stays green or red. The setup allowed the traffic lights to prioritize directions with higher vehicle presence, improving flow and reducing congestion [11]. However, our system, the Smart Traffic System, uses real-time cameras and computer vision to count vehicles, and the built-in microphones of USB cameras to detect approaching emergency vehicles. This allows our system to dynamically control traffic lights, assisting emergency vehicles by adjusting signals in their favor, and prioritizing pedestrian crossings.

This comparison Table 2.1 highlights the superiority of the Smart Traffic System in utilizing cameras, microphones, and motion sensors, combined with real-time data processing through computer vision and artificial intelligence, enabling it to effectively handle vehicle detection, emergency situations, and pedestrians.

Table 2.1: Comparison with other projects.

	Intelligent Traffic Light Guidance System[9]	Smart Traffic Lights Signals[10]	Smart Traffic Management System[11]	Smart Traffic System (Our project)
Sensing Technologies	Cameras to monitor road conditions	Proximity sensors at road start and end	IR sensors for vehicle detection	Cameras, microphone, motion sensors
Data Processing	Real-time display for drivers	Processed by Arduino and Bluetooth communication	Local processing via Arduino	Real-time processing using computer vision
Software Components	Matlab, Python	Arduino C	Arduino C	Python, Octave, YOLO, YAMNet,
Relying on artificial intelligence (AI)	Yes(limited)	No	No	Yes
Vehicle Detection	Detects vehicles via road cameras	Detects vehicles with proximity sensors at road entry and exit	Detects vehicles at intersections with IR sensors	Counts and analysis vehicles using cameras and Computer Vision
Emergency and pedestrians Handling	No	No	No	Yes

2.4. Summary

In short, combining computer vision and deep learning gives a strong base for building smart traffic systems. These tools help the system understand images and sensor data in real time, which allows it to make quick decisions on busy city roads. After studying the theory and looking at other systems, we found the main parts and problems that smart traffic systems need to deal with like improving traffic flow, keeping pedestrians safe, and giving priority to emergency vehicles.

Chapter 3

System Design

3.1. Preface

This chapter provides an examination of the key hardware and software elements necessary for our project. We explore various options for each component and offer a conceptual framework of the system's design. Additionally, we introduce a basic block diagram to illustrate the system's structure. Furthermore, we detail the system's algorithms and methodologies, while schematic diagrams depict the connections and interactions between components.

3.2. System components and Design alternatives

3.2.1. Hardware Components

3.2.1.1. Controllers

A microcontroller is a compact computer with a processor, memory, and I/O peripherals. There are several options available depending on the project requirements. For our project, we considered Raspberry Pi 5 and NVIDIA Jetson Nano, as shown in the comparison Table 3.1 which details their features and performance.

Table 3.1: Comparison between Raspberry Pi 5 and NVIDIA Jetson Nano.

	Raspberry Pi 5	NVIDIA Jetson Nano
Image		
CPU	Quad-core ARM Cortex-A76, 2.4 GHz	Quad-core ARM Cortex-A57, 1.43 GHz
RAM	4GB or 8GB LPDDR4X-4267	4GB LPDDR4
GPU	Broadcom VideoCore VII	128-core NVIDIA Maxwell GPU
Storage	MicroSD, NVMe SSD, USB 3.0	MicroSD, USB, NVMe SSD
Number of Pins	40 GPIO	40 GPIO
Camera Interface	2*4-lane MIPI CSI connectors	MIPI CSI-2 connector
I/O Ports	mHDMI, USB, Ethernet, GPIO	HDMI, USB, Ethernet, GPIO
Price	₹450 - ₹550	₹800 - ₹1000

The Raspberry Pi 5 offers all the necessary features for our project, including GPIO, pins, and I/O ports, and is preferred over the NVIDIA Jetson Nano due to its lower price.

3.2.1.2. Cameras

In our smart traffic light system, computer vision is essential. The camera captures real-time data to detect vehicles and analyze traffic flow, optimizing traffic light operation based on vehicle presence and density. This helps reduce congestion. Table 3.2 compares Logitech Camera and IP Camera (Reolink RLC-511W) to justify this choice.

Table 3.2: Comparison between Logitech Camera and IP Camera (Reolink RLC-511W).

Features	Logitech Camera	IP Camera (Reolink RLC-511W)
Primary Use	Video calls, streaming, and basic monitoring.	Security surveillance and remote monitoring.
Connectivity	Connects via USB to a computer or device.	Connects via Ethernet or Wi-Fi to a network.
Resolution	Typically 720p to 1080p (consumer-grade).	Ranges from 1080p to 4K, with higher-end models available
Power Source	Powered through USB	Requires a dedicated power source or PoE (Power over Ethernet).
Field of View (FOV)	Narrow (60°–90° typical).	Wide (up to 360° with pan/tilt/zoom features).
Installation	Plug-and-play requires a computer.	Requires network setup and configuration.
Cost	Affordable (~\$20–\$100).	Varies widely (~\$50–\$500+).

The Logitech Camera shown in Figure 3.1 is an ideal choice for our smart traffic system, offering an affordable and portable solution. Its real-time video capability meets the system's monitoring needs, making it sufficient for vehicle detection and traffic analysis without the complexity or higher cost of an IP camera (Reolink RLC-511W).



Figure 3.1: logitech camera.

3.2.1.3. USB Camera Microphones

We used built-in microphones of USB cameras to detect emergency vehicle sirens. They continuously capture ambient sound from the intersection, which is analyzed in real time using a deep learning model (YAMNet) to identify sirens based on their unique frequency patterns.

Table 3.3: External Sound vs USB Camera Microphone.

Feature	USB Camera Microphone	External Sound (e.g., MAX4466)
Microphone Type	Built-in digital microphone	Electret Microphone
Gain Control	Fixed (controlled by software)	Adjustable via potentiometer
Audio Quality	Moderate	Moderate to high
Detection Range	5–10 meters (depends on environment)	Up to 10–15 meters
Suitable Applications	General audio capture (used for sirens)	Dedicated siren detection
Sensitivity	Sufficient for typical urban use	Tunable
Cost	Included in USB camera	Around \$2.28
Performance in Noisy Environments	Acceptable (software filtering applied)	Good(after tuning)

We decided to use the built-in microphones of USB cameras to reduce system cost and simplify the hardware architecture so we don't have to add more hardware components to convert the sound from analog to digital to analyze it. Although these microphones do not provide manual adjustment options like specialized sound sensors, they were useful in emergency vehicle siren detection during testing.

3.2.1.4. Momentary Push Button

The Momentary Push Button shown in Figure 3.3 is a switch that activates only while pressed, allowing pedestrians to request a green light for crossing. This improves the system's efficiency and safety by enabling real-time manual input for smooth pedestrian flow and timely responses at intersections.



Figure 3.3: Momentary Push Button.

3.2.1.5. Motion Sensor

We used motion sensors to detect pedestrian presence around the traffic light, changing the signal to red when no movement is detected, prioritizing traffic flow. Table 3.4 compares Passive Infrared Sensor (PIR) and Ultrasonic sensors to determine the best option for this application.

Table 3.4: Compare between PIR and Ultrasonic sensors .

Features	PIR Sensor	Ultrasonic Sensor
Detection Principle	Detects infrared radiation (heat) emitted by humans.	Emits sound waves and detects reflections.
Sensitivity to Humans	High, optimized for detecting human motion.	High, optimized for detecting human motion.
Outdoor Suitability	Good, but may be affected by extreme heat or cold.	Moderate weather conditions like rain and wind can interfere.
Power Consumption	Low	Moderate to high.
Directionality	Wide detection angle (typically 100–180°).	Narrow beam (typically 15–30°).
Cost	Low (~\$2–\$10 per unit).	Moderate (~\$4–\$15 per unit)

PIR sensors shown in Figure 3.4 are cost-effective, reliable, energy-efficient, and the most important reason to choose PIR Sensor as we did its accuracy in high heat objects (such as humans) detection and reduce false alarms caused by the movement of inanimate objects making them ideal for detecting pedestrians near traffic lights, and wide detection angle ensure accurate and efficient operation, with fewer false positives compared to ultrasonic sensors.

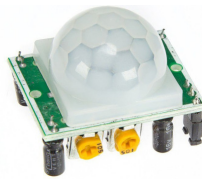


Figure 3.4: PIR Sensor.

3.2.2. Software Component

In this section, we will introduce the software components we used in our project.

3.2.2.1. Object Detection and Image Processing

The YOLO algorithm (You Only Look Once) is a method for object detection that identifies and locates objects in an image using a grid structure. It is well suited for real-time tasks like traffic monitoring because of its rapid processing speed [7].

Table 3.5 compares YOLO with Haar Cascade, highlighting the key differences between the two algorithms and evaluating their suitability for traffic monitoring applications.

Table 3.5: Compare between YOLO and Haar Cascade .

Feature	YOLO	Haar Cascade
Detection Method	Divides the image into a grid; each cell predicts bounding boxes and class probabilities.	Scans the image multiple times at different scales and orientations.
Speed	Very fast, suitable for real-time applications.	Slower due to repetitive scanning of the image.
Accuracy	Balances speed and accuracy effectively, especially in complex scenes.	Higher accuracy for detecting feature-based objects.
Efficiency	Processes entire images quickly and efficiently.	Less efficient, especially for multi-object detection.
Application	Ideal for real-time traffic monitoring and busy intersections.	Suitable for simpler applications with fewer objects.

Although the Haar Cascade algorithm is highly accurate in detecting some types of objects, we chose YOLO because of its high speed and efficiency in handling complex live scenes. These features make YOLO the ideal choice for our project that requires processing large amounts of visual data in real time. Figure 3.5 shows yolo car detection and recognition.

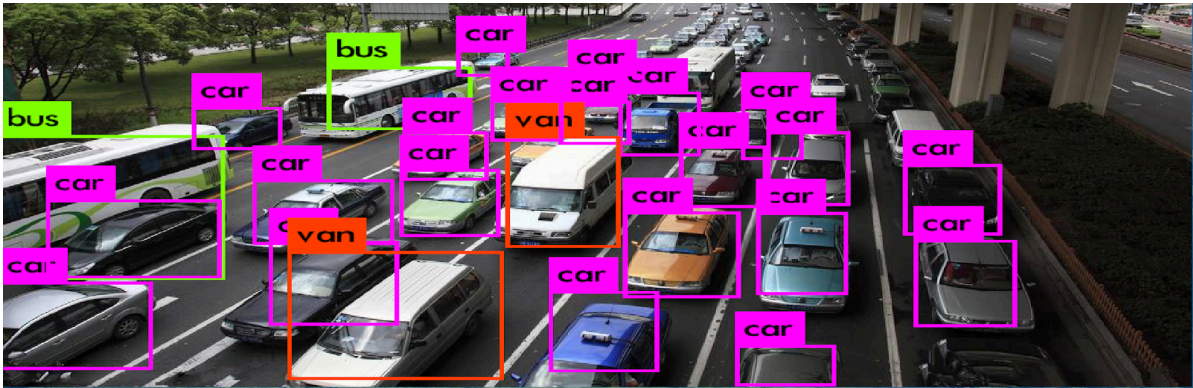


Figure 3.5: YOLO object detection and recognition [12].

3.2.2.2. Emergency Vehicle Sound Detection

YAMNet (Yet Another Mobile Network) is a deep learning-based sound recognition model developed by Google. It uses a MobileNetV2 neural network to convert audio into spectrograms for classification and is trained on 521 sound categories, achieving high accuracy in diverse environments [8].

Table 3.6 compares YAMNet and VGGish based on their feature extraction methods, ability to handle complex sounds, accuracy, and training data.

Table 3.6: Compare between YAMNet and VGGish algorithm table.

Feature	YAMNet	VGGish
Feature Extraction	Deep learning (spectrograms)	Deep learning (VGG-based)
Handling Complex Sounds	Excellent, handles overlapping sounds	Requires additional classifier; less effective alone
Accuracy	High	Depends on classifier accuracy
Training Data	Trained on AudioSet (large dataset)	Trained on AudioSet (for feature extraction)

Due to YAMNet's proficiency in recognizing complex sounds, such as ambulance sirens, without the need for manual feature extraction, it was selected for our project. YAMNet is trained on a wide array of sounds which optimizes and accelerates classification, improving system responsiveness during emergencies.

3.2.2.3. Cloud Storage

Cloud Storage for Firebase is a powerful object storage service tailored for Google's infrastructure, renowned for its low cost and ease of use. The Firebase SDKs for Cloud Storage provide top-tier security for file upload and download operations, even in challenging network conditions [13].

Table 3.7 summarizes the key differences between Firebase and AWS in terms of ease of use, cost, and real-time data synchronization.

Table 3.7: Compare between Firebase and Amazon Web Services table.

Feature	Firebase	AWS
Ease of Use	Simple setup, user-friendly interface	Requires advanced technical skills for setup and management
Cost	More affordable(almost free cost), especially for small projects	Generally more expensive, suitable for large-scale applications
Real-Time Data Synchronization	Built-in real-time synchronization for easy integration	Requires additional configuration for real-time data

We chose Firebase over AWS due to its better cost (almost free) , ease to use, and real-time data synchronization, making it ideal for our smart traffic system.

3.2.2.4. Image Acquisition and AI Model Integration

The OpenCV or Open Source Computer Vision Library is one of the most utilized open source computer vision and image processing libraries in both academia and industries. The library has efficient tools for capturing, processing, and analyzing image and video data, enabling the execution of computer vision tasks in real time on microcontrollers like the Raspberry Pi [14].

Table 3.7 summarizes the key differences between OpenCV and Pillow in terms of USB camera support, real-time processing capability, AI model integration, and compatibility with Raspberry Pi.

Table 3.8: Compare between OpenCv and PIL table.

Feature	OpenCv	PIL
USB Camera Support	Supports live camera input via <code>cv2.VideoCapture()</code>	No support for capturing from USB cameras
RealTime Processing	Optimized for real-time image capture and analysis	Not suitable for real-time applications
AI Model Integration	Directly compatible with models like YOLO	Requires manual conversion to NumPy format
Raspberry Pi Compatibility	Efficient and stable on Raspberry Pi	Requires additional setup and slower performance.

OpenCV was chosen for our project since it supports live frame grabbing from USB cameras and real-time processing. The latter is a critical support in order to monitor traffic flow and detect vehicles accurately. Unlike Pillow, which doesn't support cameras and is limited to processing static images, OpenCV offered smooth integration with YOLOv8nano and operated smoothly on the Raspberry Pi. This made it the ideal candidate for our real-time smart traffic system.

3.3. Conceptual System Description

Conceptual system design is a functioning prototype of an intelligent traffic management system using computer vision, Internet of Things (IoT) technologies, and machine learning that simulates and responds to real-world urban traffic flows. At the core of the system is a Raspberry Pi microcontroller, which talks to a collection of cameras, sensor to monitor and regulate traffic flow in real-time, as in Figure 3.6.

USB cameras with high resolution are mounted in various intersection points and connected directly to the Raspberry Pi. They capture one photo every second, and they are processed locally using the YOLOv8 (nano version) algorithm. YOLO scans every frame to detect and classify vehicles, locate them, and count the number of cars at every traffic light. Using this information, the Raspberry Pi will come to instantaneous conclusions regarding whether or not to provide each street with the green light based on vehicle density and waiting time.

The pedestrian crossing facility enhances safety with the use of a momentary push button and PIR motion sensor combined. After being pressed by a pedestrian, the facility will wait for 5 seconds before verifying movement. When there is movement, the pedestrian signal turns green. Where no movement is detected for 3 seconds, or 40 seconds (Maximum time for pedestrian signals when green), the pedestrian signal turns red in advance to allow traffic movement with maximum efficiency.

Additionally, the system continuously monitors live audio streams from USB microphones. The system can detect emergency sirens such as ambulances or police cars employing the YAMNet deep learning model. On detecting such a sound with high confidence, the system enters emergency mode, giving an instant green signal to the affected intersection for 60 seconds, while halting all other traffic and pedestrian movements.

All sensor, image, and audio data are processed locally by the Raspberry Pi, which also acts as the core control unit. It dynamically manages vehicle and pedestrian signals based on real-time inputs. For better monitoring and traceability, the system also generates organized logs for all meaningful events such as signal status updates, emergency detection, and pedestrian activations and forwards them to Firebase Realtime Database. These logs enable remote inspection of the system's decision history and facilitate debugging analysis and future optimization.

In summary, this prototype demonstrates a smart and scalable urban traffic control system successfully integrating edge AI processing, IoT infrastructure, and cloud logging to address modern smart city challenges in an effective and flexible manner.

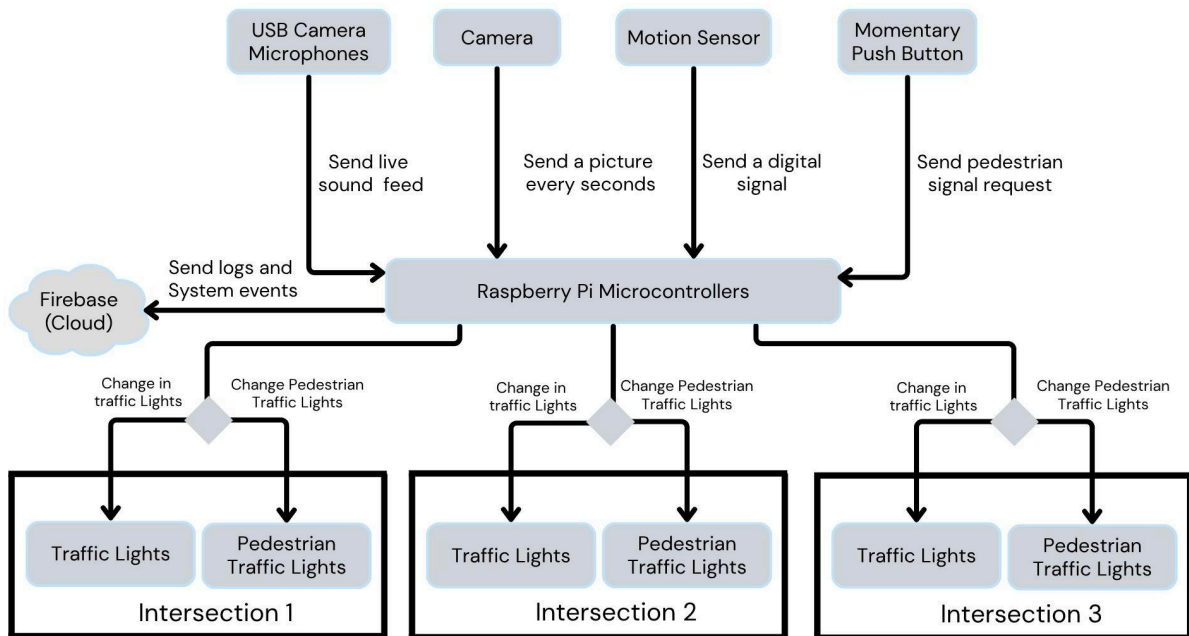


Figure 3.6: System Block Diagram.

3.4. Algorithms and Methodologies

Algorithm 1: Vehicle Detection and Counting

Every 1 second do:

For each street_index in [0, 1, 2]:

Capture frame from camera[street_index]

result $\leftarrow$ YOLOv8.detect(frame)

count $\leftarrow$ 0

For each detected_object in result:

If object.label == 'car':

count += 1 # Normal car takes 1 unit of space on street

Else if object.label == 'truck':

count += 2 # Truck is larger, takes 2 units of space on street

Else if object.label == 'bus':

count += 3 # Bus is even larger, takes 3 units of space on street

vehicle_counts[street_index] $\leftarrow$ count

Algorithm 2: Emergency Siren Detection (YAMNet)

Parallel thread for each microphone:

While True:

Record audio_sample from microphone

waveform $\leftarrow$ preprocess(audio_sample)

scores, labels $\leftarrow$ YAMNet.predict(waveform)

top_label, confidence $\leftarrow$ get_highest_score(scores)

If top_label in ['ambulance', 'fire truck', 'police car'] and confidence $\geq$ 0.65 :

If emergency_mode is OFF:

emergency_mode $\leftarrow$ ON

green_street $\leftarrow$ street of current mic

Force all vehicle lights to RED

Force all pedestrian lights to RED

Set vehicle light of green_street to GREEN

Wait 60 seconds

emergency_mode $\leftarrow$ OFF

Algorithm 3 Pedestrian Request and Motion Detection Algorithm

```
last_button_time ← 0
When pedestrian button[street] is pressed:
    current_time ← now()
    If current_time - last_button_time < 60 seconds:
        Ignore the button press
    Else:
        last_button_time ← current_time
        Wait 5 seconds
        If emergency_mode is OFF and pedestrian_mode is OFF:
            pedestrian_mode ← ON
            Force all vehicle lights to RED
            Set all pedestrian lights to GREEN
            start_time ← current_time
            While current_time - start_time < 40 seconds:
                If no motion detected for 3 seconds:
                    Break
            Set all pedestrian lights to RED
            pedestrian_mode ← OFF
            Wait 5 seconds before resuming traffic
```

Algorithm 4 Smart Traffic Management

```
Every 1 seconds do:
    If emergency_mode or pedestrian_mode is ON:
        Skip this cycle
    For each street_index in range :
        If vehicle_counts[street_index] > 0:
            waiting_times[street_index] += 1
    If any waiting_time ≥ 50:
        prioritized_street ← that street #This street has vehicles waiting for too long and needs
        priority
    Else:
        prioritized_street ← street with max(vehicle_counts)
    If prioritized_street ≠ current_green_street:
        Set current_green_street to RED
        Wait 5 seconds
        Set prioritized_street to GREEN
        current_green_street ← prioritized_street
        Reset its waiting_time
```

Chapter 4

Testing and Evaluation

4.1. Preface

In this chapter, we explain how we tested and evaluated the Smart Traffic System. We tested each hardware and software component to ensure proper functionality. We also evaluated the overall system to verify its performance in real world conditions and to confirm that all modules operate together smoothly while following the intended priority logic.

4.2. Hardware Setup and Evaluation

4.2.1. System Prototype

We built a physical prototype of a three-way intersection to test how the smart traffic system works in real life. We placed the cameras, traffic lights, and pedestrian controls on a wooden board to match how they would be used in the real world. The setup helped us test the system safely and clearly. As shown in Figure 4.1, the camera angle was good, the signals were easy to see, and the wires stayed in place during testing.

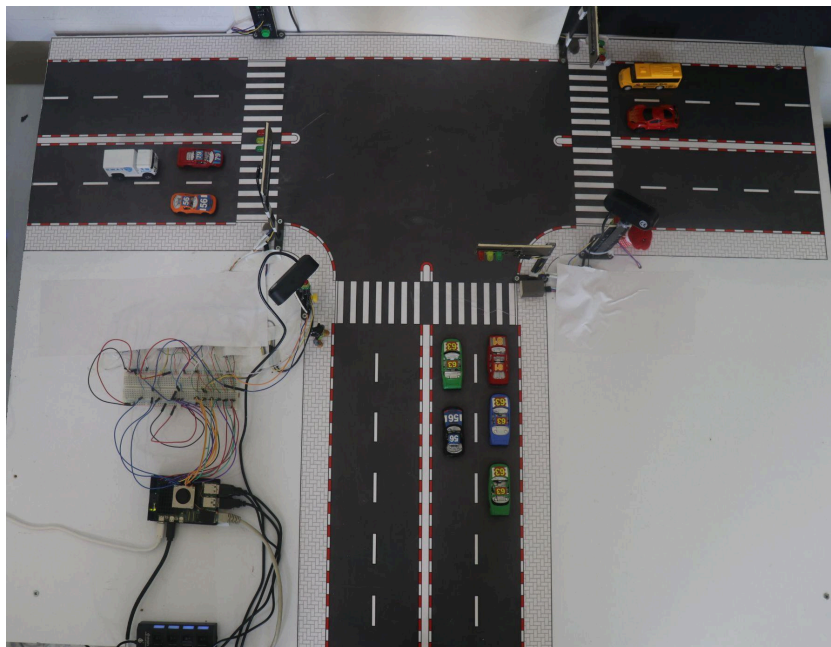


Figure 4.1: Prototype setup.

4.2.2. Raspberry Pi 5

We employed Raspberry Pi 5 as the main controller for all the processing jobs: sound detection, image analysis, and GPIO control. We selected it for having a more powerful CPU and RAM performance than its predecessors. The hardware performed all the jobs efficiently even under stressful usage. YOLOv8 and YAMNet were running alongside each other without crashing or overheating. The Raspberry Pi's performance was stable. Figure 4.2 depicts the Raspberry Pi 5 utilized in the system.

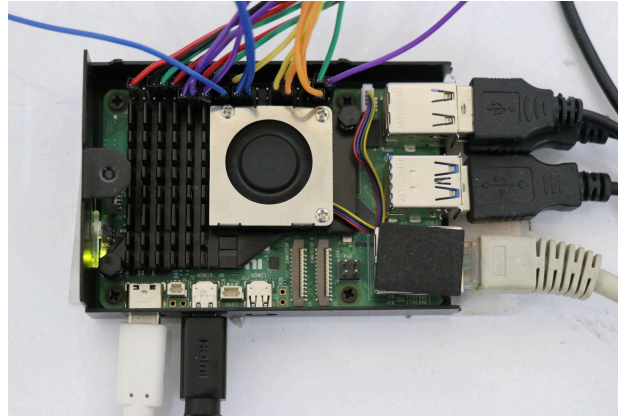


Figure 4.2: Raspberry Pi 5.

4.2.3. USB Cameras

We tested three USB cameras in the model, each placed to cover a different angle of the intersection. The cameras also had built-in microphones, allowing them to capture both image and ambient audio in real time. The cameras provided clear, high-quality images that worked well with the YOLOv8 model for vehicle detection. They also recorded clean audio used later by the YAMNet model for emergency sound recognition. Figure 4.3 shows the USB camera used in the system after installation. Figure 4.4 shows a group of vehicles as seen through the camera during the detection process.



Figure 4.3: USB Camera.

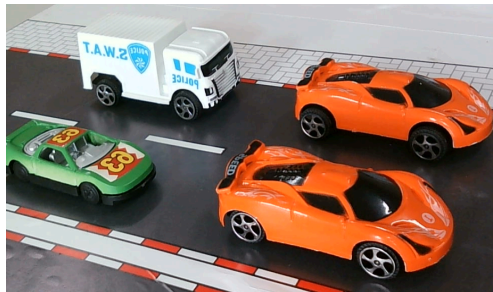


Figure 4.4: Camera-detected scene.

4.2.4. Pedestrian Request Unit

We tested the pedestrian system which includes a push button and a motion sensor. When we pressed the button, the system went into waiting mode and turned on the PIR sensor. The button, as shown in Figure 4.5, always worked right away and didn't miss any presses. The motion sensor, as shown in Figure 4.5, also worked well and only detected real movement when someone was crossing. Overall, both the button and the sensor worked together without any problems. The system was able to tell when someone wanted to cross, and it cancelled the request if no one was actually there.

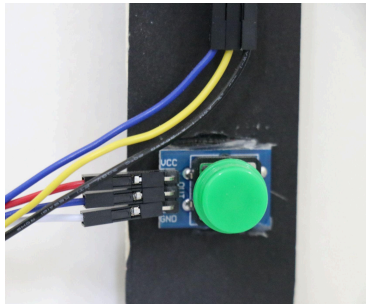


Figure 4.5: Pedestrian push button.

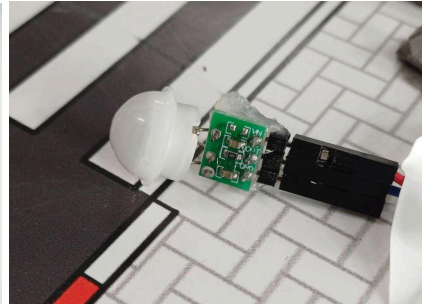


Figure 4.6: PIR motion sensor.

4.2.5. LED Traffic Lights

We employed the use of LED traffic lights in the model for the purpose of depicting road and pedestrian signals. There were red, yellow, and green LEDs within each unit. We placed the light sources onto the model and interconnected them with stable wiring. We checked the intensity and visibility of the light under typical indoor conditions. The lights were bright, clearly visible from a good distance, and turned on and off smoothly. No flickering or power issues were observed during the tests. Figure 4.7 shows the vehicle traffic light, and Figure 4.8 shows the pedestrian traffic light.

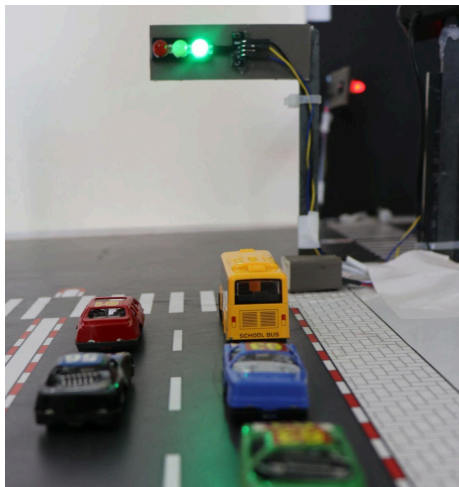


Figure 4.7: Vehicle traffic light.

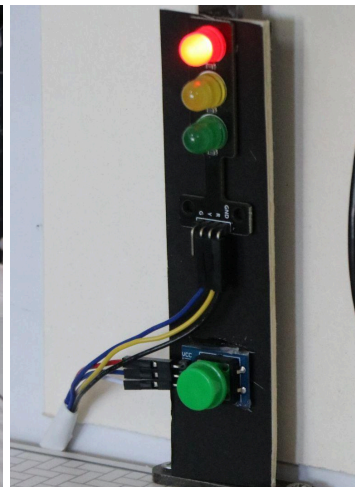


Figure 4.8: Pedestrian traffic light.

4.2.6. Power Supply and Wiring

The system was powered by a 5V/3A adapter as shown in Figure 4.9 connected to the Raspberry Pi. A powered USB hub was used to handle the additional peripherals. All wiring was carefully arranged for both safety and neatness. The power system remained stable during long tests, and no overheating or voltage issues were detected. As seen in Figure 4.10, the wiring was organized and well-secured, ensuring smooth operation.



Figure 4.9: Power adapter.



Figure 4.10: Wiring layout.

4.3. Software Modules and Performance

4.3.1. Vehicle Detection Module

This component of the system monitors vehicle traffic on each street with the help of three USB cameras, one looking at each road for the sake of complete coverage of the intersection. These cameras shoot frames at intervals, which are transmitted to the Raspberry Pi 5 for analysis. We employed a pre-trained YOLOv8n object detection model, which was chosen for its light weight and speedy performance for deployment to edge devices such as Raspberry Pi. The model scans every image to identify cars, buses, and trucks, and outputs bounding boxes, class labels, and detection confidence. Figure 4.11 presents an example street scene captured.

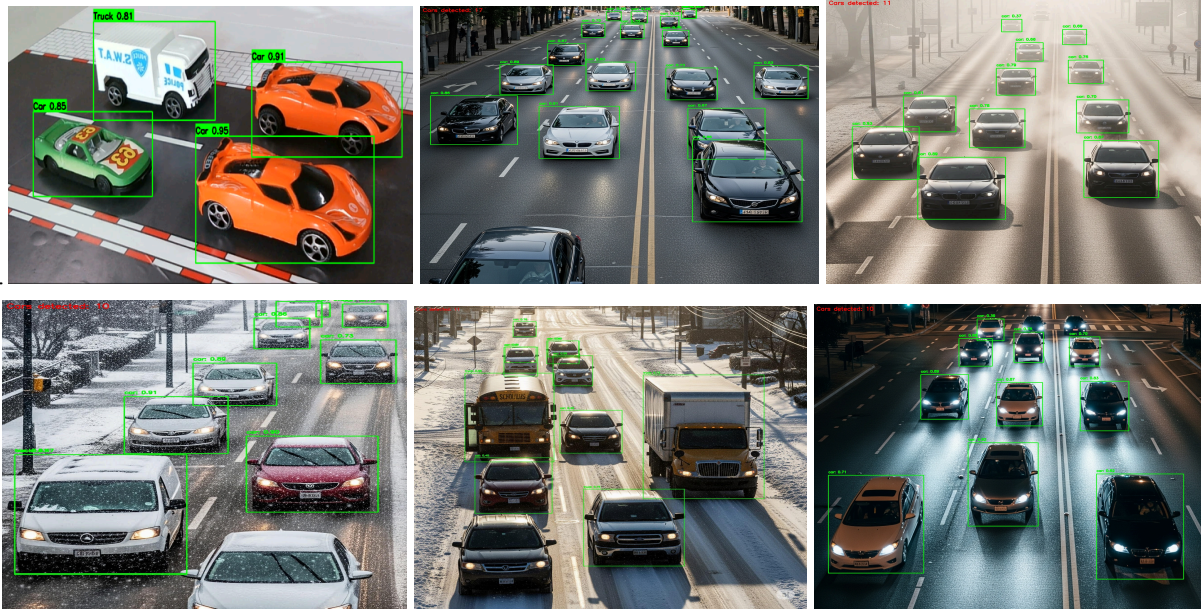


Figure 4.11: YOLOv8n detection outputs.

The number of vehicles detected per street is then sent to the control system, which selects which traffic light should turn green based on the congestion level. This real time feedback allows the system to manage traffic efficiently. The model's terminal output, which summarizes the number of vehicles, detection time, and decision logic, is shown in Figure 4.12 .

```

0: 480x640 1 truck, 390.7ms
Speed: 3.1ms preprocess, 390.7ms inference, 2.5ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 cars, 1 airplane, 409.3ms
Speed: 3.0ms preprocess, 409.3ms inference, 2.9ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 airplane, 1 bus, 358.7ms
Speed: 2.8ms preprocess, 358.7ms inference, 2.5ms postprocess per image at shape (1, 3, 480, 640)
▣ Street 1: 2 vehicles, wait: 4 seconds
▣ Street 2: 2 vehicles, wait: 4 seconds
▣ Street 3: 3 vehicles, wait: 0 seconds
  
```

Figure 4.12: YOLOv8n detection result.

We tested the detection module using over 60 images taken under various lighting conditions, including both day and night. Results were manually verified to evaluate accuracy. The system consistently identified vehicles accurately and quickly, without introducing delay to traffic control decisions.

Evaluation results:

- Average confidence (73.6%): It's an average of the confidence scores assigned by the model to each predicted bounding box.
- Precision (93.79%): Precision measures the accuracy of positive predictions - out of all the vehicles the model predicted, how many were actually correct?
- Recall (87.52%): Recall measures how many of the actual positive instances (vehicles) the model was able to detect.
- Processing time (≤ 450 ms per frame): This is the time (in milliseconds or seconds) it takes for the model to process one image frame from input to output.
- False positive rate (5.8%): Proportion of non-vehicle objects incorrectly identified as vehicles.
- False negative rate(12.48%): Proportion of actual vehicles that were not detected by the model.
- Type of car detected(99.8%): Out of all correctly detected vehicles.

Based on these results, the YOLOv8n model proved to be a reliable and effective solution for real-time vehicle detection in our smart traffic system. Its high accuracy, low error rates, and fast response make it suitable for dynamic environments where timely and accurate signal control is essential.

We sent a group of photos in different conditions to YOLO code in our project. The first photo is in normal condition, the second photo is in fog, the third photo is in night condition, and the fourth photo is also in night condition and in snow condition, Table 4.1 shows the confidence in all conditions.

Table 4.1: Different conditions, actual number of cars detected numbers and average confidence.

Environmental Condition	Actual Number of Vehicles	Detected Number	Average Confidence	Detection Precision (%)	Number of Images
Clear Day	110	105	0.83	98.10%	10
Heavy Fog	103	94	0.67	90.4%	10
Night	110	102	0.68	92.7%	10
Snowy Weather	113	109	0.82	96.47%	10
Heavy Dust	133	121	0.68	90.98%	10

Figure 4.13 shows the average confidence of the vehicle detection model across different environmental conditions. The highest confidence was recorded during clear daylight (0.83) and snowy weather (0.82) . Confidence dropped in Heavy Dust and Night (0.68) .Another weather the Confidence dropped in is heavy fog (0.67) . The overall average confidence is 0.73 , indicating generally good performance.

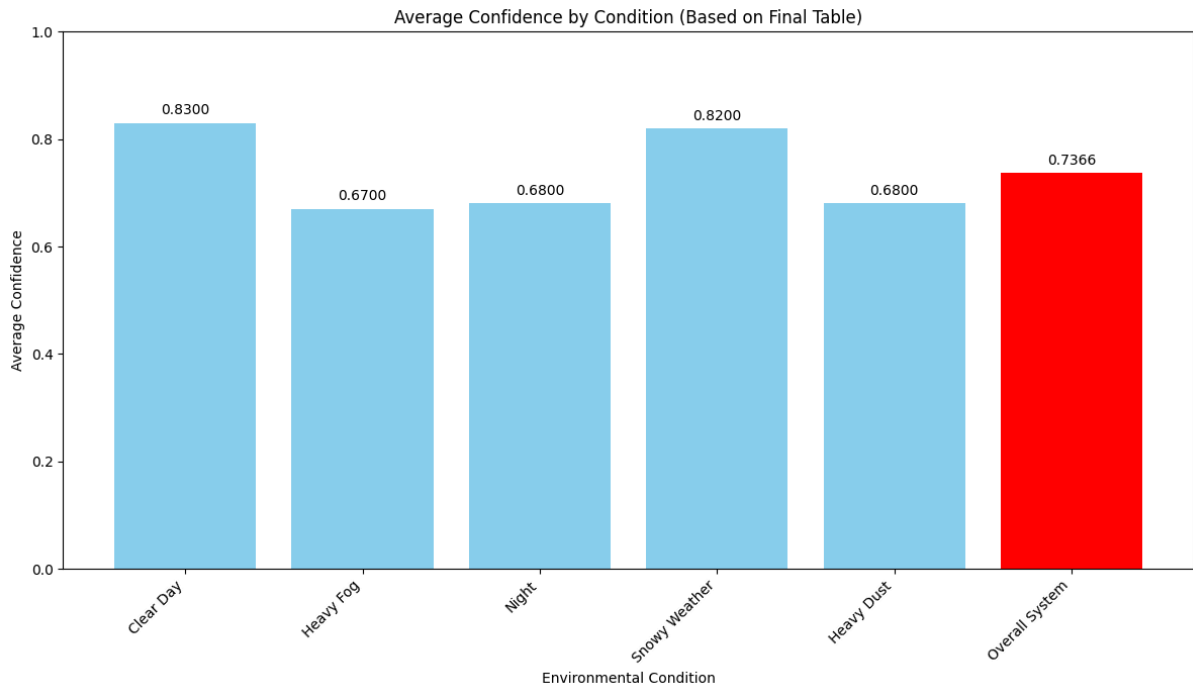


Figure 4.13: Average confidence in different environmental conditions.

4.3.2. Emergency Vehicle Detection Module

This part of the system detects emergency vehicles by analyzing ambient sound captured from the intersection. Instead of relying on a dedicated sound sensor, we used the built-in microphones of the USB cameras to continuously collect real-time audio.

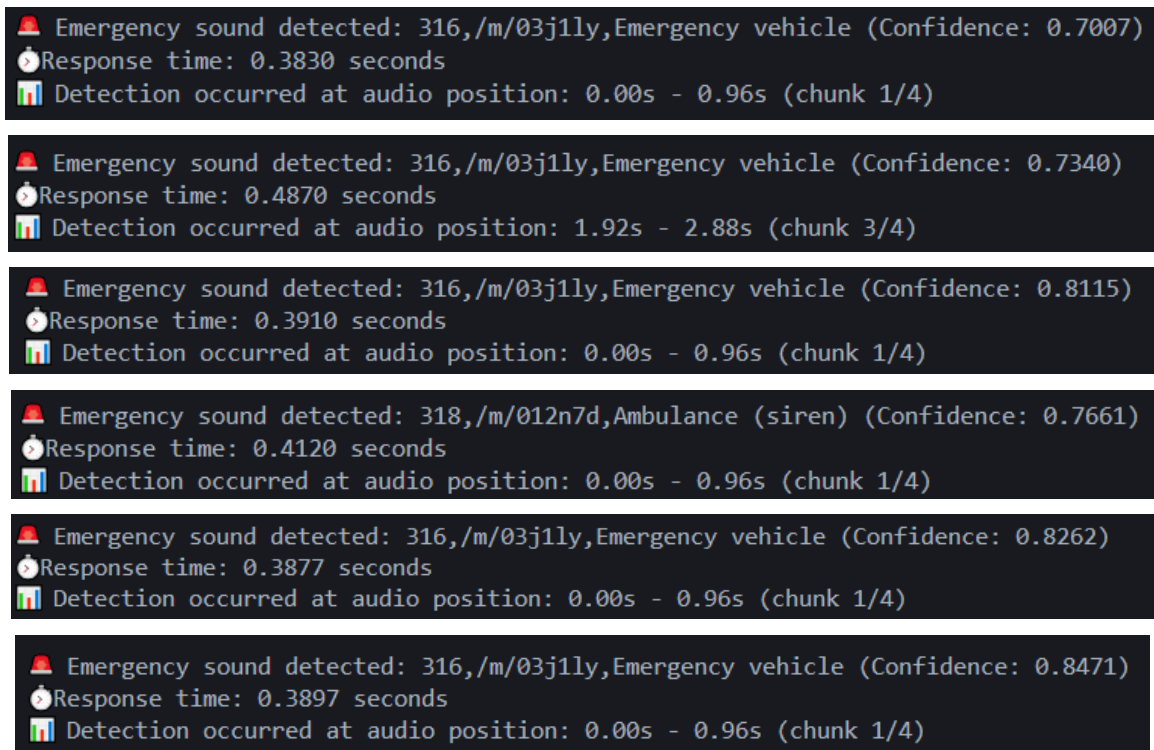
We employed YAMNet, a pre-trained deep learning model developed by Google for sound classification. It can recognize over 500 sound types and runs locally on the Raspberry Pi 5. The model listens to short audio clips, classifies the sound, and returns the most likely label along with a confidence score. If a siren is detected with high confidence, the system overrides the normal traffic logic and gives a green light to the corresponding lane, allowing emergency vehicles to pass without delay.

To evaluate the system, we tested it by playing emergency sirens using a speaker, combined with realistic background noise including traffic sounds and human conversations. YAMNet processed the sounds in real time and displayed the detected label, confidence percentage, and response time.

Evaluation Results:

- Average Detection Accuracy (99%): Most sirens were correctly identified.
- Average response Time (0.408 second): The system responded fast enough for real-time traffic adjustment.
- False Positive Rate(0%): Some loud non-siren sounds were mistakenly classified as sirens.
- Confidence Scores (78.04%)(70% min – 84.71% max): Most detections showed strong confidence values. However, in cases where background noise was very loud, the confidence sometimes dropped noticeably.

An example of the system's terminal output is shown in Figure 4.14, where the detected sound class, confidence score, and processing time are clearly reported.



```
🚒 Emergency sound detected: 316,/m/03j1ly,Emergency vehicle (Confidence: 0.7007)
🕒 Response time: 0.3830 seconds
📊 Detection occurred at audio position: 0.00s - 0.96s (chunk 1/4)

🚒 Emergency sound detected: 316,/m/03j1ly,Emergency vehicle (Confidence: 0.7340)
🕒 Response time: 0.4870 seconds
📊 Detection occurred at audio position: 1.92s - 2.88s (chunk 3/4)

🚒 Emergency sound detected: 316,/m/03j1ly,Emergency vehicle (Confidence: 0.8115)
🕒 Response time: 0.3910 seconds
📊 Detection occurred at audio position: 0.00s - 0.96s (chunk 1/4)

🚒 Emergency sound detected: 318,/m/012n7d,Ambulance (siren) (Confidence: 0.7661)
🕒 Response time: 0.4120 seconds
📊 Detection occurred at audio position: 0.00s - 0.96s (chunk 1/4)

🚒 Emergency sound detected: 316,/m/03j1ly,Emergency vehicle (Confidence: 0.8262)
🕒 Response time: 0.3877 seconds
📊 Detection occurred at audio position: 0.00s - 0.96s (chunk 1/4)

🚒 Emergency sound detected: 316,/m/03j1ly,Emergency vehicle (Confidence: 0.8471)
🕒 Response time: 0.3897 seconds
📊 Detection occurred at audio position: 0.00s - 0.96s (chunk 1/4)
```

Figure 4.14: YAMNet detection output.

These results confirm that YAMNet is an effective and reliable solution for emergency vehicle detection in smart traffic systems.

4.4. Implementation Challenges and Issues

While building the Smart Traffic System, we faced many real-world challenges that needed flexible thinking and practical solutions. These issues were grouped into two main types: hardware and software. The next parts explain these challenges and how we solved each one.

4.4.1. Hardware Implementation Challenge

- **Limited USB Ports on Raspberry Pi:** The Raspberry Pi 5 only gives you four USB ports. This was a challenge when we were putting our system together. We needed to plug in three USB cameras for seeing the vehicles and hearing audio, plus a mouse and keyboard for controlling the system. Since we had more things to plug in than available ports, we used a USB hub to add more connections. This helped make sure everything could run at the same time.

4.4.2. Software Implementation Challenges

- **Unstable Device IDs for USB Cameras and Microphones :** We kept having a problem with the USB cameras and microphones. Every time we restarted the Raspberry Pi, it would give these devices new ID numbers. This was a problem because our system couldn't recognize the devices correctly anymore. The device locations we had saved in our computer code didn't match the new ID numbers of the actual cameras and microphones. We tried really hard to give the devices unchanging, permanent ID numbers, but we just couldn't get those IDs to stay the same. So, because of this ongoing issue, we had to resort to manually changing the device IDs in our code each time before we ran the system. This manual step took a long time and often caused mistakes, especially when we were testing repeatedly.
- **YOLOv5 Processing Delay:** One of the challenges we faced was that YOLO analysis for a specific image captured from a street took no less than 1050 milliseconds. This delay was due to our use of YOLOv5. We addressed this issue by switching from YOLOv5 to YOLOv8-nano, which resulted in a significant improvement in image processing time—reducing it to no more than 450 milliseconds, table 4.2 shows the comparison between YOLOv5 and YOLOv8-nano.

Table 4.2: Comparison of Real-Time Image Processing Performance Between YOLOv5 and YOLOv8-nano.

Metric	YOLOv5 (Before)	YOLOv8-nano	Improvement
Inference Time	848.2 ms – 1050 ms	356.3 ms – 450 ms	~57% faster
Post Processing Time	2.6 ms – 5.5 ms	2.1 ms – 5.5 ms	Minor
Total Processing Time	≥ 1050 ms	≤ 450 ms	~44% faster

- **Thread Synchronization and Performance Bottlenecks:** When we were building the system, we gave important jobs like finding cars, recognizing sirens, and watching pedestrians their own separate work paths so they could run at the same time (this is called parallel processing). At first, each USB camera also had its own path to take pictures, analyze them using the YOLOv8 algorithm, and count vehicles on its own. But, trying to make all the cameras work together simultaneously really overloaded the Raspberry Pi's brain (its CPU). Because of this overload, the system would frequently crash or just freeze up. So, our solution was to redesign the way pictures were handled. Rather than processing all three cameras at the same time, we set it up so a single process would deal with them one by one. This change eased the burden on the processor, and the system still reacted fast enough for managing traffic as it happened.

4.5. Module Integration and Priority Logic Testing

To ensure seamless interaction between all subsystems, we conducted integration testing focusing on verifying correct coordination and adherence to the intended priority logic among the vehicle detection, emergency sound detection, pedestrian handling, and traffic light control units. The main goal was to ensure the system behaves properly when multiple events occur simultaneously and that the priority order is respected.

The system was observed while handling live inputs from three USB cameras, which provided images for the YOLOv8n vehicle detection model and audio for the YAMNet emergency siren detection model. Simultaneously, the pedestrian system monitored crossing requests using a push button and a PIR motion sensor.

The test verified the following priority order:

1. **Emergency Detection (Highest Priority):** When a siren sound was detected with high confidence by YAMNet, all other processes were immediately suspended. The system granted a green light to the lane of the approaching emergency vehicle to ensure uninterrupted passage.
2. **Pedestrian Crossing (Medium Priority):** When the button was pressed and motion confirmed, the system allowed pedestrian crossing but only if no emergency event was active. Vehicle traffic was stopped and pedestrian signals were activated for a limited time.

3. **Traffic Congestion (Lowest Priority):** In the absence of both emergency and pedestrian activity, the YOLOv8n model counted vehicles on each road. The system then granted the green light to the most congested road to help distribute traffic efficiently.

This testing confirmed that:

- The system consistently followed the correct priority: Emergency → Pedestrians → Vehicles
- All subsystems communicated effectively and responded without delays or conflicts.
- Transitions between modes (normal, pedestrian, emergency) were smooth and precise.

These results verify the successful integration of all modules and the system's ability to prioritize events accurately in real-time.

4.6. Full System Evaluation Under Real Scenarios

After completing unit and integration testing, we performed full system testing to evaluate the behavior of the entire system under realistic traffic intersection scenarios. All major subsystems were tested together: vehicle detection, emergency detection, pedestrian system, and traffic light control.

Scenario 1: Normal Traffic – No Emergency, No Pedestrians

The system operated in its default mode where no pedestrian requests or emergency vehicles were present. Cameras captured images from all streets and YOLOv8n analyzed the number of vehicles. The system selected which street to prioritize based on the following parameters: vehicle count, waiting time for each street, and a predefined maximum duration for green signals. The system automatically granted green lights to the most congested or longest-waiting street, then rotated to others in a fair and timely manner. Traffic signals responded correctly with no conflicts.

Figure 4.15 shows a sample output from the system terminal, where Street 3 was prioritized due to a long waiting time, despite having fewer vehicles than Street 2. This demonstrates how the system balances both vehicle count and fairness in wait time.

```

□ Street 1: 0 vehicles, wait: 0 seconds
□ Street 2: 4 vehicles, wait: 0 seconds
□ Street 3: 3 vehicles, wait: 53 seconds
□ Street 3 has waited for 53 seconds (threshold: 50)
□ Changed signal from Street 2 to Street 3 due to long wait
□ Reset wait timer for Street 3 (got green signal)
□ Started wait timer for Street 2

```

Figure 4.15: Terminal output during normal traffic mode.

Scenario 2: Pedestrian Request During Normal Traffic

During regular traffic conditions, a pedestrian pressed the push button to request crossing. The system entered pedestrian mode and activated the PIR motion sensor to detect actual presence. Once movement was confirmed, vehicle lights turned red and pedestrian lights were activated for a maximum of 40 seconds or until no motion was detected. After the crossing ended, the system returned to normal operation. The system responded quickly and accurately, with no false triggers. Figure 4.16 shows the terminal output during this pedestrian crossing scenario, including motion detection, pedestrian mode activation, and automatic return to normal traffic flow after no motion was detected.

```

✓ Pedestrian motion detection initialized for crossing 1
□ Motion stopped at crossing 1
□ Motion detected at crossing 1
□ Motion stopped at crossing 1
□ No pedestrian motion for 3 seconds at crossing 1. Stopping pedestrian mode.
□ Deactivating pedestrian mode.
□ Waiting 5 seconds before restoring traffic signals...
□ Restoring traffic flow: Street 2 gets green signal with 4 vehicles
□ Reset wait timer for Street 2 (got green signal)
□ Started wait timer for Street 3

```

Figure 4.16: Pedestrian Mode Terminal Output.

Scenario 3: Emergency Vehicle During Normal Traffic

A simulated ambulance siren was played near the USB camera microphone. The YAMNet model successfully identified the sound as an emergency vehicle with high confidence. All ongoing traffic and pedestrian activity was halted immediately. The system granted a green signal to the emergency direction and maintained it for 60 seconds to ensure uninterrupted passage. This confirmed the system's effectiveness in handling urgent situations.

Figure 4.17 shows the terminal output from this test, where YAMNet correctly detected the emergency sound, reported the confidence level, and recorded the response time in real time.

```

□ Street 1: 0 vehicles, wait: 0 seconds
□ Street 2: 4 vehicles, wait: 0 seconds
□ Street 3: 3 vehicles, wait: 36 seconds
□ EMERGENCY DETECTED on Street 1! Sound: 316,/m/03j1ly,Emergency vehicle (Score: 0.87)
□ Saved previous green signal: Street 2
□ Reset wait timer for Street 1 (emergency detected)

```

Figure 4.17: YAMNet terminal output during emergency detection.

Scenario 4: Pedestrian Request During Emergency Mode

While the system was in emergency mode, a pedestrian pressed the crossing button. Despite detecting the request and motion via the PIR sensor, the system ignored the pedestrian action until the 60-second emergency window ended. This ensured that the emergency vehicle retained full right-of-way. Once the emergency cleared, the system resumed normal mode and allowed pedestrian crossing. Figure 4.18 displays the system's behavior during this event, where the pedestrian request was blocked due to active emergency mode, confirming that emergency events have the highest priority.

```

[Pedestrian Signal 1] Button pressed, waiting 5 seconds...
[Pedestrian 1] Cannot activate: Emergency Mode is active.

```

Figure 4.18: Terminal output showing system behavior during emergency mode.

Scenario 5: Emergency Detected During Pedestrian Crossing

A pedestrian was already crossing, and an emergency siren was detected. The system immediately interrupted the pedestrian mode, halted pedestrian lights, and switched traffic control to prioritize the emergency direction. After the 60-second emergency passage was completed, the system reverted to regular operation. This scenario validated the system's strict enforcement of priority rules. As shown in Figure 4.19, the terminal output clearly demonstrates this behavior. The system detected motion at crossing 3 and activated pedestrian mode. However, once an emergency sound with high confidence (score: 0.90) was detected on Street 1, the system saved the previous green signal, deactivated pedestrian mode, and granted the green signal to the emergency direction. This ensured immediate response while maintaining proper state recovery after the emergency.

```

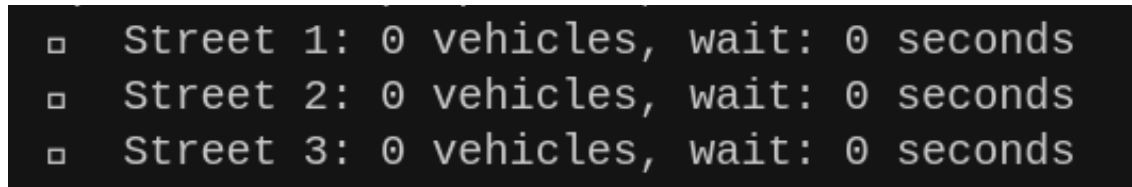
EMERGENCY DETECTED on Street 1! Sound: 316,/m/03j1ly,Emergency vehicle (Score: 0.90)
Saved previous green signal: Street 3
Reset wait timer for Street 1 (emergency detected)
Emergency detected while pedestrian mode is active. Forcing deactivation.
Deactivating pedestrian mode.
Restoring traffic flow: Street 1 gets green signal with 3 vehicles
Reset wait timer for Street 1 (got green signal)
Emergency mode detected while pedestrian mode is active. Deactivating pedestrian mode.

```

Figure 4.19: Terminal output during emergency override in pedestrian mode.

Scenario 6: All Roads Empty – No Vehicles, Pedestrians, or Emergencies

In the absence of any vehicles, pedestrian activity, or emergency sounds, the system entered an idle state. All traffic signals remained red, and no actions were taken until a new event occurred. This behavior prevented unnecessary signal changes and kept the system running efficiently, as shown in Figure 4.20, where all streets are empty with zero wait time. Across all scenarios, the system followed the designed priority order: Emergency then Pedestrians then Vehicles. Transitions between modes were smooth and accurate. The system consistently demonstrated reliable and precise responses under various real-world situations, confirming its readiness for actual deployment.



```
□ Street 1: 0 vehicles, wait: 0 seconds
□ Street 2: 0 vehicles, wait: 0 seconds
□ Street 3: 0 vehicles, wait: 0 seconds
```

Figure 4.20: Terminal output during system idle state.

4.7. Summary

In this chapter, we reviewed the testing and evaluation of the intelligent traffic system. Each module was tested individually and then as part of the integrated system to ensure proper performance and coordination. The results confirmed that the system operates accurately and reliably, adhering to the logic of priorities in real-world conditions.

Chapter 5

Conclusion

5.1. Conclusion

The smart traffic system demonstrably shows the potential of combining AI models and sensor technologies to improve urban mobility. Employing YOLOv8n for real-time vehicle detection and YAMNet for emergency vehicle siren detection, the system can enforce adaptive traffic signal control that reduces congestion and improves emergency and pedestrians' response time. The use of a Raspberry Pi 5, PIR sensors, and momentary push buttons results in seamless integration of software and hardware with the capability to make dynamic decisions based on real-time data.

One of the system's greatest features is the condition to provide priority to emergency vehicles. Using YAMNet, the system identifies ambulance and fire truck sirens and any emergency sound when they are approaching the system, overriding traffic lights to immediately grant them right-of-way. This functionality is able to reduce delays for emergency services dramatically, potentially saving lives during actual emergencies. Also, the system ensures pedestrian safety through motion sensors and push-button interfaces, halting car traffic for 40 seconds to allow a safe crossing if the pedestrian is detected by the motion sensor.

The traffic signal system has a priority order of strict sequencing: priority to emergency vehicles, pedestrians second, and congestion control third. When there are no constant emergencies or pedestrian requests, the system employs YOLOv8n to analyze vehicle density and adjust signal time accordingly. Testing measured 98.33% accuracy in vehicle detection, 4.84% false negative, and 1.61% false positive, indicating good performance in controlled systems.

Despite these achievements, there are significant limitations to the system. It currently operates in a classroom test setting with precision-controlled lighting and minimal noise interference. Under real conditions rush hour or inclement weather the precision of visual and acoustic detection can be compromised. Second, the Raspberry Pi 5 dependency on the

Raspberry Pi 5 constrains real-time processing speed compared to platforms like the NVIDIA Jetson Nano.

Yet, an intelligent traffic system is a crucial step towards safer and smarter city transportation. Its modular design allows it to support extension into the future, and integrating machine learning models and physical sensor fusion lays the groundwork for scalable applications of a smart city.

5.2. Future Work

To enhance the system's performance and adaptability, several improvements are recommended:

1. **Hardware and Software Optimization:** Upgrading from Raspberry Pi 5 to NVIDIA Jetson Nano would greatly increase real-time processing speed, facilitating more fluid signal control and faster object detection. Furthermore, the use of directional microphones technology helps to reduce background noise interference, thus improving the emergency vehicle detection in more noisy city streets.
2. **Enhanced machine learning Models:** Refining the YOLOv8n model with additional training data including diverse vehicle types like detecting the emergency vehicle from image using yolo, weather conditions, and lighting scenarios could improve detection accuracy. Similarly, fine-tuning YAMNet with specialized emergency siren datasets would increase its reliability in characteristics between different types of emergency vehicles.
3. **Real-World Deployment:** The system must be implemented in actual urban environments to verify its operation under dynamic conditions. Collaboration with local traffic authorities would facilitate pilot implementation at high-traffic intersections, obtaining valuable feedback for future refinements.
4. **Traffic Lights Communication:** The future can be dedicated to making intercommunication of traffic lights across different parts of the city so as to improve the efficiency and scalability of the system. Traffic lights can exchange real-time data about vehicle density, emergency alerts, and signal status with surrounding intersections using IoT-based protocols (e.g., MQTT, LoRaWAN) or 5G connectivity.
5. **Enhanced Emergency Vehicle Coordination:** In future improvements, the system could provide for simultaneous coordination of multiple emergency vehicles, rather than just one vehicle every 60 seconds currently. This will utilise a two criteria priority system based on vehicle type (for example, ambulances should always have priority to fire trucks, or police vehicles) and distance to the intersection (in the event of equal type, the closest vehicle will have priority).

References

- [1] S. Sharma, "Computer Vision Examples," 2021, Accessed: Nov. 7, 2024 . [Online]. Available: <https://blog.gramener.com/computer-vision-examples/>.
- [2] L. Craig, "What is machine learning? Guide, definition and examples," Accessed: Nov.7,2024.[Online].Available: <https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML>.
- [3] Z. Kelta, " Convolutional Neural Networks (CNN) with TensorFlow Tutorial," 2023, Accessed: Nov. 7, 2024. [Online]. Available:<https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns>.
- [4] M. Mishra, "Convolution online photo," 2018, Accessed: Nov. 7, 2024. [Online]. Available:<https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939/> A Guide to Convolutional Neural Networks — the ELI5 way | Saturn Cloud Blog.
- [5] J. Schulze, "What Is the Internet of Things (IoT)? With Examples," 2024. Accessed: Nov. 7, 2024 [Online].Available: <https://www.coursera.org/articles/internet-of-things>.
- [6] Azure, "What is cloud computing?," Accessed: Nov. 7, 2024. [Online]. Available: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing>.
- [7] R. Kundu, "YOLO Object Detection: What It Is and How It Works," 2024, Accessed: Nov. 10, 2024. [Online]. Available: <https://www.v7labs.com/blog/yolo-object-detection>.
- [8] Google, "YAMNet: Pretrained Model for Audio Classification," Accessed: Nov. 13, 2024. [Online]. Available: <https://www.kaggle.com/models/google/yamnet> .
- [9] A.Dabbas and S.Froukh, "Intelligent Traffic Light Guidance System (ITLGS)," B.Sc. thesis, College of IT and Computer Engineering, Palestine Polytechnic University, 2024.[Online].Available:<https://scholar.ppu.edu/handle/123456789/6684>.

- [10] H. Dwayat, K. Awwad and S. Jawabreh, "Smart Traffic Lights Signals," B.Sc. thesis, College of IT and Computer Engineering, Palestine Polytechnic University, 2020. [Online]. Available: <https://scholar.ppu.edu/handle/123456789/7147>.
- [11] N. H. Nagapragna and S. S. Joshi, "SMART TRAFFIC MANAGEMENT SYSTEM," Department of Electrical and Electronics Engineering, Sapthagiri College of Engineering, Bangalore, 2024. Accessed: Nov. 7, 2024 [Online]. Available: https://www.researchgate.net/publication/384054535_SMART_TRAFFIC_MANAGEMENT_SYSTEM.
- [12] E. Saul, "YOLO Object Detection and Recognition," Accessed: Nov. 11, 2024. [Online]. Available: <https://github.com/EternalSaul/Vehicle-Detection>.
- [13] Google, "Cloud Storage Documentation," Accessed: Dec. 3, 2024. [Online]. Available: <https://firebase.google.com/docs/storage>.
- [14] A. Smorkalov, "Open Source Computer Vision Library," Accessed: Nov. 13, 2024. [Online]. Available: <https://opencv.org/>

Declaration and Documentation of AI Tools Usage in the Graduation Project

Purpose of Use	AI Tool Used	Report Sections	Prompt Instruction Used	Was Output Modified?	Estimated Usage %	Notes
Diagram Generation	ChatGPT	Chapter 1 – Figure 1.1	Help me draw a system diagram for Smart Traffic System	Yes	15%	Used to generate system architecture sketch for Figure 1.1
Text Structuring	ChatGPT	Chapter 2 – Literature Review	Help organize and rephrase literature review paragraphs	Yes	10%	Used to organize and improve clarity of previous studies
Text Explanation	ChatGPT	Chapter 3 – Section 3.2	Explain system components in simple way	Yes	15%	Used to help with writing
Writing Assistance	ChatGPT Gemini	All chapters	Help in writing	Yes	20%	Helped draft introduction to implementation
Analysis Guidance	ChatGPT	Chapter 4 – Sections 4.3.1 & 4.3.2	What performance metrics should I evaluate for YOLO and YAMNet	Yes	20%	Guided which metrics to include
Clarification	ChatGPT	Chapter 4 – Sections 4.5 & 4.6	Difference between integration testing and system testing	Yes	15%	Clarified concepts and helped write summary
Text Structuring	ChatGPT	Chapter 5 – Testing and Results	Help rephrase and organize the test result paragraphs	Yes	10%	Used to polish the paragraph structure
Code Generation	Manus AI, Qwen AI	Chapter 3 – Implementation Code	Generate Python code for YOLOv8 and YAMNet on Raspberry Pi	Yes	15%	Helped draft and troubleshoot initial code
Image Generation	Gemini AI	Chapter 5 – YOLO Testing Image	Create realistic images for YOLO testing with cars	Yes	15%	Used in test presentation and report figures

Graduation Project Team Declaration:

We declare that the use of AI tools was carried out responsibly and in accordance with university policies. All outputs have been reviewed and edited to reflect our personal understanding.

Group team: Samah Mebed

Signature:

Mohammad Taha

Signature:

Momen Qarajeh

Signature:

Date: 1/6/2025