Vision of the Institution

To become a leading technical institute of academic excellence by imparting high patterns of discipline through innovative programs of global standards making our students technically superior and ethically strong to serve the Nation.

Mission of the Institution

To create an environment that shall foster the growth of intellectually capable, innovative professionals who can contribute to the growth of Technology in partnership with industry and develop and harness it for the welfare of the Nation and mankind.

Vision of the Department

To evolve as a centre of Excellence, to train students in contemporary technologies to meet the needs of global industry and to develop them into skillful engineers instilled with human values and Professional ethics.

Mission of the Department

Impart quality education to produce successful entrepreneurs and globally competent engineers for meeting the current and future needs of industry.

Program Educational Objectives (PEOs):

PEO:1	Have fundamental and advanced knowledge in Electronics and communication Engineering and excel in their professional carrier or in higher studies/research.
PEO:2	Have Knowledge in the design and development of innovative electronics and communication systems or sub-systems, to meet the needs of the society.
PEO:3	Have demonstrated practices and skills in communication, professional attitude, teamwork, leadership, values and ethics.

Program Outcomes (POs):

Engineering Graduates will be able to:

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern Tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and of the engineering practice.

9. Individual and Team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Objectives (PSOs):

PSO:1	**Professional Skills:** Students shall abilities to take challenges associated in the fields of communication, networking, signal processing, embedded and Semiconductor technology.
PSO:2	**Competency:** Student shall qualify at the state, National and International level competitive examination for employment, higher studies and research.

ABSTRACT

Traffic management in urban areas is a pressing challenge due to increasing vehicle density and inefficient traffic signal timings. Traditional fixed-timing systems often fail to adapt to dynamic traffic conditions, leading to congestion and delays. This project aims to design an intelligent traffic signal control system that adjusts signal timings in real-time based on traffic density at intersections, optimizing the flow of vehicles and minimizing waiting times.

The system uses advanced sensors such as infrared, radar, or cameras to monitor traffic density at each lane of an intersection. Data from these sensors is processed through algorithms that calculate optimal signal durations for each phase. By dynamically adjusting the green signal duration, the system ensures that high-density lanes receive more time, while less congested lanes wait, thereby balancing traffic flow effectively.

The proposed solution integrates Internet of Things (IoT) technologies for real-time communication and control. Traffic data is transmitted to a central server, which processes the information using machine learning or heuristic algorithms. Additionally, the system can provide insights to traffic authorities about peak hours and recurring congestion points, enabling long-term planning and infrastructure improvements.

This smart traffic management system is expected to reduce fuel consumption, air pollution, and commuting times significantly. It enhances the efficiency of urban traffic networks and provides a scalable solution for growing cities. By leveraging technology to address real-world problems, this project contributes to sustainable urban development and improved quality of life for commuters.

TABLE OF CONTENTS

CHAPTER 7

LIST OF FIGURES

LIST OF TABLES

CHAPTER -1

INTRODUCTION

Traffic congestion is one of the most significant challenges faced by urban areas worldwide. As cities continue to grow and vehicle density increases, managing traffic efficiently becomes more difficult. Conventional traffic management systems, which rely on pre-set signal timings, often fail to accommodate the dynamic nature of road traffic. These traditional methods lead to congestion, increased fuel consumption, prolonged travel times, and higher levels of air pollution. Addressing these issues requires an intelligent system that can adapt to real-time traffic conditions and optimize signal timings accordingly.

The primary goal of this project is to develop an intelligent traffic signal control system that adjusts signal timings based on real-time traffic density at intersections. The system will utilize advanced sensors such as infrared, radar, or cameras to monitor traffic volume in each lane. The collected data will be processed using sophisticated algorithms to determine optimal green light durations, ensuring that heavily congested lanes receive more time while less crowded lanes experience minimal delays. This dynamic approach to traffic signal control will help balance vehicle flow across intersections, reducing overall congestion.

To enhance the system's efficiency, Internet of Things (IoT) technology will be integrated, allowing real-time data transmission and communication between traffic signals and a central server. By leveraging machine learning and heuristic algorithms, the system can continuously analyse traffic patterns and make data-driven adjustments. This capability enables better decision-making for traffic authorities, providing insights into peak congestion hours, high-traffic zones, and areas that require long-term infrastructure improvements.

By implementing this smart traffic management system, we aim to significantly reduce fuel consumption, air pollution, and commuter waiting times. The project contributes to the broader vision of sustainable urban development, where technology plays a crucial role in improving transportation networks. As urban populations grow, intelligent traffic control solutions will be essential in creating more efficient, eco-friendly, and commuter-friendly road systems.

The proposed system is designed to be scalable and adaptable, making it suitable for both small and large cities. By using a real-time adaptive traffic signal control system, this languages

and development tools. Companies provide embedded system development kits, enabling engineers to create and refine these systems with precision.

languages and development tools. Companies provide embedded system development kits, enabling engineers to create and refine these systems with precision.

While embedded systems require significant investment in development time and design optimization, their importance in critical industries such as automotive, healthcare, telecommunications, and industrial automation makes them highly valuable. As technology advances, embedded systems continue to evolve, incorporating AI, IoT, and machine learning capabilities, further enhancing their efficiency and adaptability in modern smart devices and interconnected environments.

1.1 EMEBEDDED SYSTEMS

Embedded systems are specialized electronic systems that incorporate microcomputers to perform dedicated tasks within a larger device. These systems are designed to control, monitor, or assist the operation of equipment and processes. Unlike general-purpose computers that run multiple applications, embedded systems are optimized for specific functions, making them highly efficient, reliable, and tailored to their intended purpose.

At the core of an embedded system lies a microprocessor or microcontroller, which simplifies system design while offering flexibility for future modifications. By utilizing a microprocessor, embedded systems enable easy bug fixes, firmware updates, and the addition of new features simply by rewriting the dedicated software—eliminating the need for significant hardware modifications. Although these computers remain hidden within products, they are essential to their intelligence and functionality.

Embedded systems are ubiquitous in modern technology. Every day, millions of microchips are produced and integrated into devices we use regularly, such as smartphones, smart appliances, medical equipment, and industrial machinery. These systems function as self-contained programs embedded within hardware, operating autonomously to execute predefined tasks. Unlike traditional computers that can run various applications, embedded systems are programmed for a single or limited set of operations, requiring physical alterations for any major functional changes.

A defining characteristic of embedded systems is their optimal efficiency—they are engineered to perform tasks with minimal processing time and power consumption, ensuring

high-speed execution and reliability. Designers of embedded systems possess a strong understanding of hardware and software integration, utilizing specialized programming languages and development tools. Companies provide embedded system development kits, enabling engineers to create and refine these systems with precision.

While embedded systems require significant investment in development time and design optimization, their importance in critical industries such as automotive, healthcare, telecommunications, and industrial automation makes them highly valuable. As technology advances, embedded systems continue to evolve, incorporating AI, IoT, and machine learning capabilities, further enhancing their efficiency and adaptability in modern smart devices and interconnected environments

1.1.1 CHARACTERISTICS

Single-functioned:

An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager.

Tightly constrained:

All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. It must be of a size to fit on a single chip, must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.

All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. fast enough to process data in real time and consume minimum power to extend battery life.

Reactive and Real time:

Many embedded systems must continually react to changes in the systems environment and must compute certain results in real time without any delay. Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors. It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to control of the car.

Microprocessors based:

It must be microprocessor or microcontroller based.

Memory:

It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer. Connected It must have connected peripherals to connect input and output devices.

Connected:

It must have connected peripherals to connect input and output devices.

HW-SW systems:

Software is used for more features and flexibility. Hardware is used for performance and security.

Advantages

1. Easily Customizable
2. Low power consumption
3. Low cost

Disadvantages

1. High development effort
2. Larger time to market

1.1.2 BASIC STRUCTURE OF EMBEDDED SYSTEM

The following illustration shows the basic structure of an embedded system –

Sensor: It measures the physical quantity and converts it to an electrical signal converter.

A sensor stores the measured quantity to the memory.

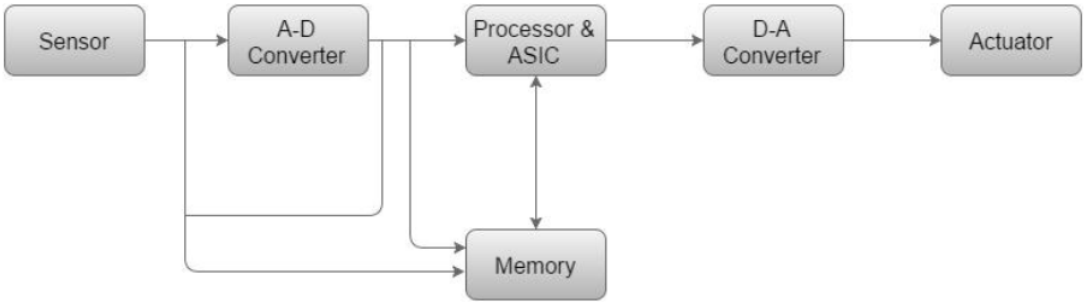

Fig 1.1: Basic structure of Embedded Systems

- **A-D Converter:** The analog-to-digital converter converts the analog signal sent by the sensor into a digital signal.
- **Processor &ASICs:** Processors process the data to measure the output and store it to the memory.
- **D-A Converter:** A digital-to-analog converter converts the digital data fed by the processor to analog data
- **Actuator:** An actuator compares the output given by the D-A Converter to the actual (expected) output stored in it and stores the approved output.

1.2 IOT (Internet of Things)

The Internet of things (IOT) is the inter-networking of physical devices, vehicles (also referred to as "connected devices" and "smart devices"), buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. In 2013 the Global Standards Initiative on Internet of Things (IOT-GSI) defined the IOT as "the infrastructure of the information society."

The IOT allows objects to be sensed or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy, and economic benefit in addition to reduced human intervention. When IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities. Each thing is uniquely identifiable through its embedded computing system but can interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020.

Typically, IoT is expected to offer advanced connectivity of devices, systems, and services that goes beyond machine-to-machine (M2M) communications and covers a variety of protocols, domains, and applications. The interconnection of these embedded devices (including smart objects), is expected to usher in automation in nearly all fields, while also enabling advanced applications like a smart grid and expanding to areas such as smart cities.

“Things," in the IoT sense, can refer to a wide variety of devices such as heart monitoring implants, biochip transponders on farm animals, electric clams in coastal waters,

automobiles with built-in sensors, DNA analysis devices for environmental/food/pathogen monitoring or field operation devices that assist firefighters in search and rescue operations.

Legal scholars suggest looking at "Things" as an "inextricable mixture of hardware, software, data and service". These devices collect useful data with the help of various existing technologies and then autonomously flow the data between other devices.

Current market examples include home automation (also known as smart home devices) such as the control and automation of lighting, heating (like smart thermostat), ventilation, air conditioning (HVAC) systems, and appliances such as washer/dryers, robotic vacuums, air purifiers, ovens or refrigerators/freezers that use Wi Fi for remote monitoring.

As well as the expansion of Internet-connected automation into a plethora of new application areas, IoT is also expected to generate large amounts of data from diverse locations, with the consequent necessity for quick aggregation of the data, and an increase in the need to index, store, and process such data more effectively. IoT is one of the platforms of today's health monitoring systems, Smart City, and Smart Energy Management Systems.

1.3 IMPORTANCE OF TRAFFIC SIGNALS

Traffic light signals play a vital role in maintaining order, safety, and efficiency in urban transportation. They regulate the movement of vehicles and pedestrians, reducing the risk of accidents and ensuring a smooth traffic flow. By managing intersections effectively, traffic lights help prevent collisions and provide pedestrians with safe crossing points. Additionally, well-synchronized signals reduce congestion, especially during peak hours, by optimizing vehicle movement. Modern traffic light systems also prioritize emergency vehicles, such as ambulances and fire trucks, ensuring they reach their destinations quickly without unnecessary delays. This contributes to faster emergency response times, potentially saving lives.

Ensuring Road Safety: Traffic lights help prevent accidents and collisions by providing clear right-of-way instructions for vehicles and pedestrians. By controlling the flow of traffic at intersections, they reduce the risk of crashes, especially in high-traffic areas where multiple roads intersect.

Reducing Traffic Congestion: Properly synchronized traffic signals help in managing vehicle movement efficiently, reducing congestion during peak hours. By optimizing signal timings

based on traffic flow, modern traffic lights improve overall road capacity and minimize unnecessary delays**.**

Enhancing Pedestrian Safety: Traffic lights include dedicated pedestrian signals that allow people to cross roads safely. Without them, pedestrians would be at high risk of accidents, particularly at busy intersections and crosswalks. These signals ensure an organized interaction between vehicles and pedestrians.

Facilitating Emergency Vehicle Movement: Advanced traffic signal systems can prioritize emergency vehicles like ambulances, fire trucks, and police cars, ensuring they pass through intersections quickly. This helps in reducing response times during emergencies, potentially saving lives.

Improving Fuel Efficiency and Reducing Pollution: Unregulated intersections often lead to frequent stopping and acceleration, which increases fuel consumption and emissions. Properly managed traffic signals help vehicles maintain a steady flow, reducing fuel wastage and air pollution, contributing to a cleaner urban environment.

Supporting Smart City Infrastructure: Modern traffic signals use IoT, AI, and real-time traffic monitoring to adjust timings based on road conditions. This improves urban traffic management, supports smart city initiatives, and helps in data-driven planning for future road network improvements.

Maintaining Discipline and Order: Traffic signals enforce strict rules and regulations, ensuring drivers and pedestrians follow proper guidelines. This reduces reckless driving, red-light violations, and chaotic road conditions, promoting safer and more organized urban transportation.

Traffic light signals are an essential part of city infrastructure, ensuring safety, efficiency, and sustainability in urban transportation. With advancements in intelligent traffic management systems, cities can further optimize their signal operations, improving overall mobility and enhancing the quality of life for commuters.

1.4 PROBLEM DEFINITION

Existing traffic signal systems lack adaptability to fluctuating traffic conditions, causing unnecessary congestion and inefficiencies. Fixed-timing mechanisms do not prioritize high-density lanes, leading to long waiting times and suboptimal traffic flow. Without real-time monitoring and control, traffic authorities struggle to respond effectively to peak-hour

congestion and unexpected road conditions. There is a need for an intelligent system that can dynamically adjust signal timings based on real-time traffic data to enhance the efficiency of urban traffic networks.

1.5 OVERVIEW

The proposed system will be implemented at road intersections, where traffic signals will be adjusted dynamically based on real-time sensor data. The scope includes:

- Deployment of sensors to detect vehicle density in each lane.
- Development of an intelligent algorithm to optimize signal durations.
- Integration with IoT for real-time monitoring and data transmission to a central control system.
- Analysis and reporting to assist urban planners in identifying congestion trends.

1.6 OBJECTIVE OF PROJECT

The primary objective of this project is to design and develop an intelligent traffic signal control system that dynamically adjusts signal timings based on real-time traffic density. The system aims to:

- Monitor traffic density using advanced sensors such as infrared, radar, or cameras.
- Optimize signal timings through real-time data processing and algorithm-based decision-making.
- Reduce congestion by prioritizing high-density lanes and improving traffic flow.
- Integrate IoT technologies for seamless communication and control.
- Provide insights to traffic authorities for long-term urban traffic planning.

1.7 SIGNIFICANCE

This project contributes to sustainable urban development by improving traffic flow efficiency, reducing vehicle idle time, and lowering fuel consumption. By implementing intelligent traffic signal control, cities can experience:

- Reduced air pollution due to minimized vehicle idling.
- Lower fuel costs for commuters.

- Improved road safety through better traffic management.
- Enhanced commuter experience with reduced travel delays.

1.8 ORGANIZATION OF THESIS

Chapter-1: It gives the information about motivation, overview, problem definition and objective of project.

Chapter-2: Presents the block diagram and Schematic diagram of the entire System. It gives the basic information and Working nature of portable robot system for dispensing sanitization in hospitals.

Chapter-3: Presents the technical details of the Hardware Components.

Chapter-4: Presents the Software details of the Project and Steps involved in the Simulation of the Code.

Chapter-5: Presents the results of the project with explanation. The report then presents the applications, Conclusion, future Scope along with references and appendix.

CHAPTER-2

LITERATURE OVERVIEW

Traffic congestion is one of the major challenges in urban transportation systems, leading to increased travel time, fuel consumption, and environmental pollution. Traditional traffic management systems rely on fixed-time control, which does not adapt to real-time traffic conditions, resulting in inefficiencies. To address this issue, an intelligent traffic control system based on traffic density is proposed, utilizing Raspberry Pi 3 Model B and a CCTV-like camera.

The proposed system integrates computer vision and deep learning techniques to dynamically adjust traffic signal timing based on vehicle density at intersections. By using the YOLO object detection model, the system can detect and count the number of vehicles in real time and adjust the signal durations accordingly. This ensures a smooth and optimized traffic flow, reducing unnecessary waiting times and improving overall road efficiency.

2.1 EXISTING METHOD

Traditional traffic management systems rely on fixed-time traffic signals, where the signal timings are pre-programmed based on historical traffic data or standard time cycles. These systems operate on a preset schedule, regardless of the actual traffic conditions at a given time. As a result, even during low traffic hours, vehicles may have to wait unnecessarily at red signals, while during peak traffic hours, congestion builds up due to insufficient green signal durations. This lack of adaptability leads to inefficiencies, increased travel time, fuel wastage, and higher levels of air pollution. Additionally, manual traffic control is sometimes used, where traffic police intervene to adjust traffic flow based on observations. However, this method is prone to human error, inefficiency, and inconsistency, making it an unreliable solution for modern urban areas.

Another existing approach includes sensor-based traffic management, where inductive loop sensors or infrared sensors are embedded in the road to detect vehicles and adjust signals accordingly. While this method improves traffic control compared to fixed-time systems, it has several limitations, including high installation and maintenance costs. Moreover, environmental factors such as weather conditions, dirt, or road wear can affect sensor accuracy, leading to incorrect traffic measurements. These traditional methods fail to efficiently handle dynamic traffic patterns, especially in rapidly growing cities where vehicle density fluctuates frequently.

Hence, there is a need for a more intelligent and adaptable system that can respond in real-time to actual traffic conditions rather than relying on fixed or sensor-based inputs.

2.1.1 DRAWBACKS

The existing traffic control methods have several limitations like Inefficient Traffic Flow due to Fixed-timer systems cause vehicles to stop unnecessarily, even when there is little or no traffic from a certain direction. Higher Fuel Consumption occurs when Vehicles idling at signals for extended durations lead to increased fuel wastage and pollution. Infrastructure Costs for Induction loop sensors require road modifications, frequent maintenance, and high installation costs. Inability to Adapt to Real-Time Traffic that's why, system cannot dynamically adjust based on traffic density, causing congestion during peak hours and inefficiency during low-traffic periods.

2.2 PROPOSED METHOD

The proposed method aims to implement an intelligent traffic signal control system that dynamically adjusts signal timings based on real-time traffic density. This system utilizes a Raspberry Pi 3 Model B as the central processing unit, along with a CCTV-like camera to continuously monitor traffic at intersections. The captured video feed is processed using a YOLO (You Only Look Once) model, a deep learning-based object detection algorithm, to detect and count the number of vehicles in each lane. Based on the vehicle density, the Raspberry Pi calculates the optimal green light duration for each direction, ensuring smooth traffic flow and reducing unnecessary waiting times. The traffic lights, implemented using LEDs, are controlled accordingly to reflect the computed signal timings.

Unlike conventional fixed-time systems, this approach ensures adaptive traffic management by prioritizing lanes with higher vehicle density, thus minimizing congestion. The system is cost-effective and requires minimal infrastructure modifications since it relies on computer vision instead of physical sensors. Additionally, it can function efficiently under varying traffic conditions, improving overall road efficiency, reducing fuel consumption, and lowering vehicular emissions. By leveraging machine learning and real-time data processing, this method provides a smart and scalable solution for modern urban traffic control.

Image Processing:

The heart of the proposed system lies in its image processing capabilities, which allow the detection and counting of vehicles in real-time. The CCTV-like camera captures continuous video footage of the intersection, which is then processed using OpenCV and deep learning

models on the Raspberry Pi. The YOLO object detection algorithm is applied to detect cars, buses, and other vehicles, ensuring accurate vehicle count without false detections from pedestrians or static objects.

To enhance detection accuracy, various preprocessing techniques such as noise filtering, edge detection, and brightness adjustments are used. Additionally, background subtraction is applied to eliminate unnecessary elements and focus only on moving vehicles. By processing each frame efficiently, the system ensures real-time responsiveness, making it highly effective in dynamic traffic conditions.

Traffic Density Analysis:

Traffic density analysis is a crucial aspect of the proposed method, enabling real-time decision-making for signal control. The YOLO model not only detects vehicles but also counts them in each lane, helping the system determine which direction has the highest traffic congestion. This count is then compared against predefined thresholds to allocate appropriate green signal durations.

For instance, if one lane has significantly more vehicles than the others, the system extends the green signal to clear congestion. On the other hand, if a lane has very few vehicles, the green duration is shortened, allowing better traffic distribution. This adaptive timing mechanism ensures that no lane experiences prolonged waiting periods, improving overall road efficiency.

Signal Control Mechanism:

The signal control mechanism of the system is designed to dynamically adjust traffic light durations based on real-time vehicle density. The Raspberry Pi 3 Model B acts as the controller, receiving vehicle count data from the image processing module and calculating the optimal timing for green, yellow, and red signals. The traffic lights, implemented using LEDs, are connected to the Raspberry Pi's GPIO pins, which control their switching based on computed durations.

The system follows a priority-based approach, where lanes with heavy traffic receive longer green signals, while less congested lanes receive shorter durations. This not only prevents unnecessary idling and fuel consumption but also enhances traffic flow. The adaptive nature of the system allows it to handle sudden traffic fluctuations, such as unexpected congestion or emergency vehicles, making it highly responsive and efficient.

Power Supply Unit:

The entire traffic control system requires a stable power supply to ensure uninterrupted operation. The Raspberry Pi 3 Model B operates on a 5V DC power supply, which is derived from a regulated power source. A 12V-to-5V voltage regulator is used to step down the power supply, ensuring that the Raspberry Pi receives a steady voltage level. Additionally, the LED traffic lights require separate power regulation, which is handled through resistors and external power sources to prevent fluctuations that could affect performance.

A backup power system, such as a rechargeable battery or UPS, can be integrated to ensure the system remains functional in case of power outages. This ensures continuous operation of the traffic lights and image processing unit, preventing any disruptions in traffic management.

Advantages of the Proposed System:

The proposed system offers several key advantages over traditional traffic control methods. One of the primary benefits is real-time adaptability, where the system dynamically changes signal timings based on actual traffic conditions, reducing congestion and improving road efficiency. Additionally, the use of a vision-based approach eliminates the need for expensive sensors, making the system cost-effective and easy to install.

Another advantage is fuel and time efficiency. By reducing unnecessary wait times at red signals, vehicles spend less time idling, leading to lower fuel consumption and reduced emissions. The system is also scalable, meaning it can be implemented in multiple intersections and interconnected to create a smart traffic network. Furthermore, with remote monitoring and data analytics capabilities, traffic authorities can gain valuable insights into urban traffic patterns, helping in long-term city planning and infrastructure development.

2.3 BLOCK DIAGRAM AND WORKING

The block diagram provides a structured overview of the system's components and their interconnections. The entire system is divided into multiple functional units, each playing a crucial role in ensuring the efficient operation of the traffic signal timing control.

At the heart of the system lies the Raspberry Pi 3 Model B, which serves as the primary processing unit. It is responsible for receiving traffic density data from the server and controlling the LED traffic signals accordingly. The Raspberry Pi is programmed using Python scripts to execute real-time decisions based on the computed traffic density.

Raspberry Pi as the Core Processing Unit:

The Raspberry Pi is connected to multiple GPIO pins that control the Red, Yellow, and Green LEDs. These LEDs function as traffic lights at different lanes, changing dynamically according to the density-based schedule.

CCTV Cameras for Image Capture:

The system incorporates multiple CCTV cameras positioned at key traffic intersections. These cameras continuously capture images of the traffic lanes, providing essential visual data for the vehicle detection process. The cameras are connected to the Raspberry Pi via USB or CSI interfaces, allowing seamless data transfer for image processing.

The captured images are processed using deep learning-based object detection models to identify the number of vehicles in each lane. The information extracted from these images is used for calculating the traffic density in real time.

Vehicle Detection and Image Processing Module:

This module consists of the YOLO-based object detection algorithm deployed on the Raspberry Pi. It processes the images captured by the CCTV cameras and identifies vehicles based on their shape, size, and features.

The image processing module works efficiently to detect vehicles in different lighting and weather conditions, ensuring robust and accurate traffic density calculations. The system updates vehicle count at regular intervals to maintain real-time accuracy.

LED-Based Traffic Lights:

The system employs LED-based traffic lights controlled via the Raspberry Pi's GPIO pins. Each traffic signal consists of Red, Yellow, and Green LEDs. These LEDs change based on the time duration calculated by the system, allowing dynamic control of traffic movement.

The LEDs are powered through appropriate resistors to regulate current flow and prevent damage. The traffic lights are positioned at different lanes and operate synchronously with the density-based timing schedule.

Power Supply Module:

The entire system operates on a regulated power supply. The Raspberry Pi is powered using a 5V DC adapter, while the LEDs are connected via current-limiting resistors to ensure stable operation. The cameras and other components are also powered accordingly to maintain seamless functionality.

The power supply module ensures that all electronic components receive the necessary voltage and current levels for efficient operation. Any fluctuations in power are managed using voltage regulators to prevent disruptions in traffic signal operations.

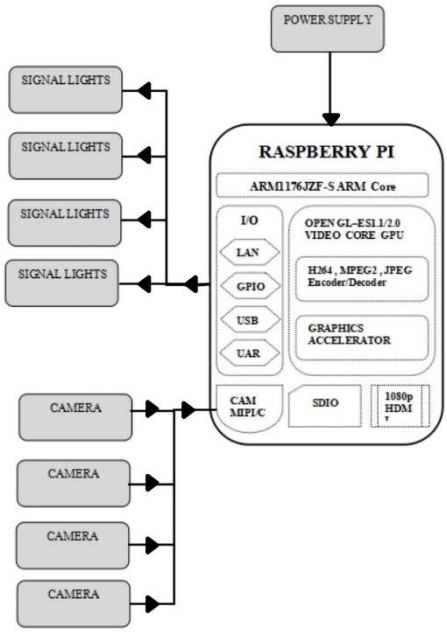

Fig 2.1: Block Diagram of Traffic Density-Based signal Timing System

2.3.1 FLOW CHART

The traffic density-based signal timing system is designed to optimize traffic management using real-time image processing techniques. The system relies on CCTV cameras installed at traffic signals to capture images of the road and analyse vehicle density. Based on the detected density, the green signal duration is dynamically adjusted to reduce congestion and improve traffic flow.

The flowchart of the intelligent traffic control system represents the sequential process followed by the Raspberry Pi to manage traffic efficiently. The system starts by capturing live video feed from the camera and feeding it into the YOLO-based object detection model. The detected vehicles are counted, and based on the count, the Raspberry Pi determines the appropriate signal durations for each lane.

If a lane has higher traffic density, the system extends the green signal duration to clear congestion efficiently. If a lane has lower vehicle density, the green signal duration is reduced

to allow smoother traffic distribution. This process is continuously repeated in a loop, ensuring real-time adaptability.

Capturing Images Using CCTV Installed at Traffic Signals:

The first step in the system involves capturing live images from CCTV cameras positioned at traffic intersections. These cameras continuously monitor the traffic situation and send real-time images to the processing unit. The captured images serve as raw data for analysing vehicle density and optimizing traffic light timings.

The use of high-resolution cameras ensures that even small vehicles and motorcycles can be accurately detected. The cameras are installed at strategic angles to cover a wide field of view, capturing a comprehensive image of the road segment under observation.

Traffic Density Sent to Server for Calculating Green Signal Time:

The calculated traffic density data is transmitted to a central server, where the traffic light timing is adjusted dynamically. The server processes the density information and applies pre-defined algorithms to determine the optimal duration of the green light.

The server uses historical traffic data, machine learning models, and real-time inputs to make intelligent decisions. It ensures a fair distribution of green signal time across multiple intersections, preventing congestion build-up at any specific road segment.

Detecting Vehicles Using Image Processing and Calculating Traffic Density:

Once the images are captured, they are processed using a deep learning-based object detection algorithm such as YOLO (You Only Look Once). This algorithm identifies and counts the number of vehicles in each image frame. The system distinguishes between different types of vehicles, such as cars, buses, and motorcycles, to estimate the traffic density accurately.

After detecting the vehicles, the system calculates the traffic density by dividing the total vehicle count by the available road area. This density value is used to determine the optimal green light duration for the traffic signal. Higher density results in a longer green signal to ease congestion, while lower density shortens the green signal duration to optimize the overall traffic flow.

Using This Time Scheduling is Done:

Based on the server's computation, the system updates the time scheduling of the traffic signals. The newly calculated time settings are transmitted to the Raspberry Pi controller, which manages the LED-based traffic lights.

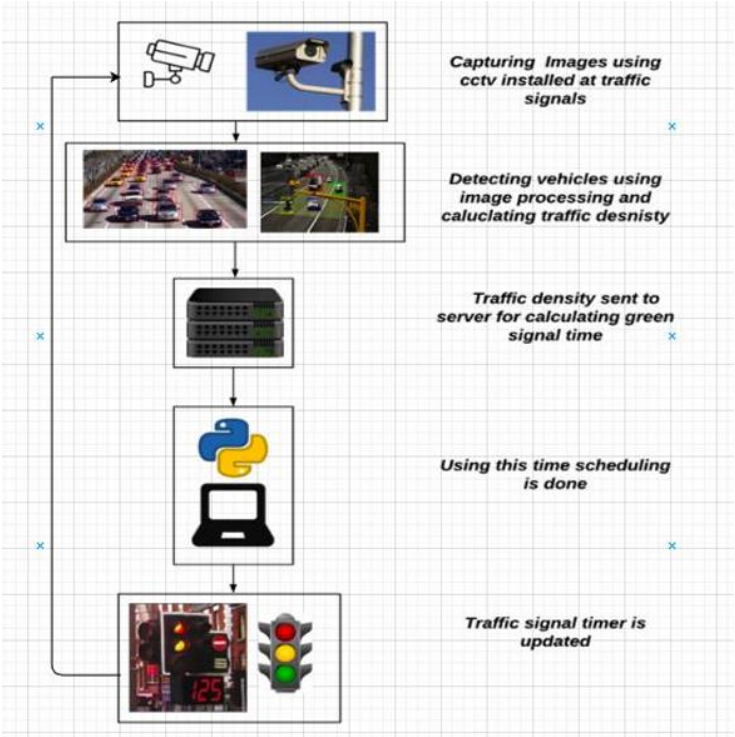

Fig 2.2: Working Flow Chart

The Raspberry Pi processes the received signal timing information and executes the schedule accordingly. The system ensures that the transition between red, yellow, and green lights happens smoothly without abrupt changes, minimizing the risk of accidents.

Traffic Signal Timer is Updated:

Once the new signal timing is calculated, the traffic lights are updated accordingly. The LED traffic lights display the revised timings, allowing vehicles to move efficiently based on real-time traffic conditions.

The timer updates dynamically as new traffic density data is received, ensuring the system remains adaptive to changing traffic conditions. This approach significantly reduces waiting times at intersections and helps in better traffic management across urban areas.

System Integration and Overall Workflow:

All the above-mentioned modules are interconnected to form a complete traffic density-based signal timing system. The system functions in a continuous loop, updating signal timings dynamically based on real-time traffic density measurements.

By integrating deep learning-based image processing, real-time decision-making, and LED-controlled traffic lights, this system provides an efficient and intelligent solution for managing urban traffic congestion. The use of Raspberry Pi as the core controller ensures cost-effectiveness while maintaining high performance and reliability.

2.3.2 CIRCUIT DIAGRAM

The given circuit diagram illustrates the connection of a Raspberry Pi 3 Model B with multiple traffic signal LEDs and cameras to create an intelligent traffic control system. The Raspberry Pi serves as the main processing unit, controlling the LED-based traffic lights based on real-time traffic density analysis. The cameras capture live images of traffic flow, and the Raspberry Pi processes these images using a YOLO-based object detection model to estimate vehicle density. The LED traffic lights, connected to the GPIO pins of the Raspberry Pi, are regulated accordingly, with appropriate resistors ensuring proper current flow to prevent damage.

This system dynamically adjusts traffic signal timings based on real-time data, optimizing traffic flow and reducing congestion. The circuit also includes multiple sets of red, yellow, and green LEDs, each corresponding to different traffic lanes, mimicking an actual traffic light system. The cameras are positioned at strategic points to capture different directions of traffic movement. The integration of software-based image processing and hardware control ensures an automated, efficient, and responsive traffic management solution.

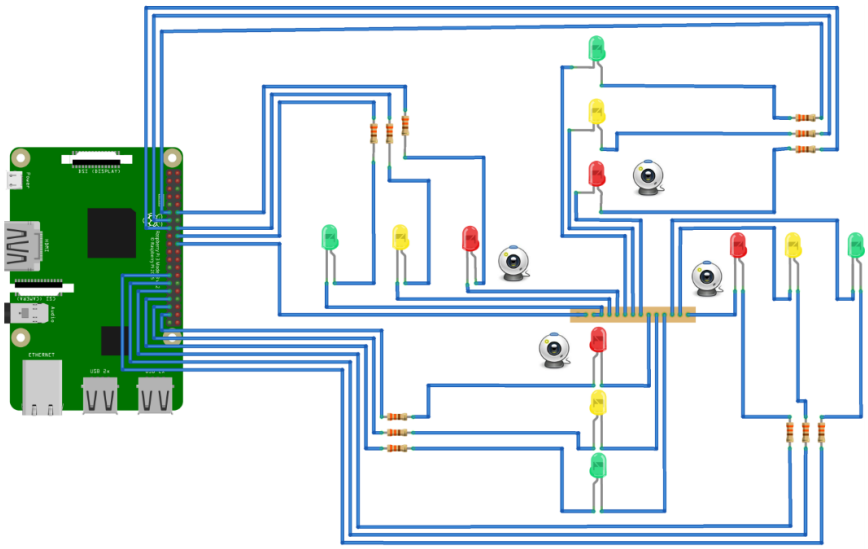

Fig 2.3: Circuit Diagram of the System

CHAPTER 3

HARDWARE COMPONENTS AND INTERFACING

The components and tools which are required for the designing of this project are considered as two types they are hardware and software. The hardware equipment used in this project are Raspberry Pi 3, Camera, LEDs, Resistors and Power Supply.

For the designing of Density based Traffic Light Signals using Raspberry Pi 3 the required hardware specifications are explained below.

3.1 RASPBERRY PI

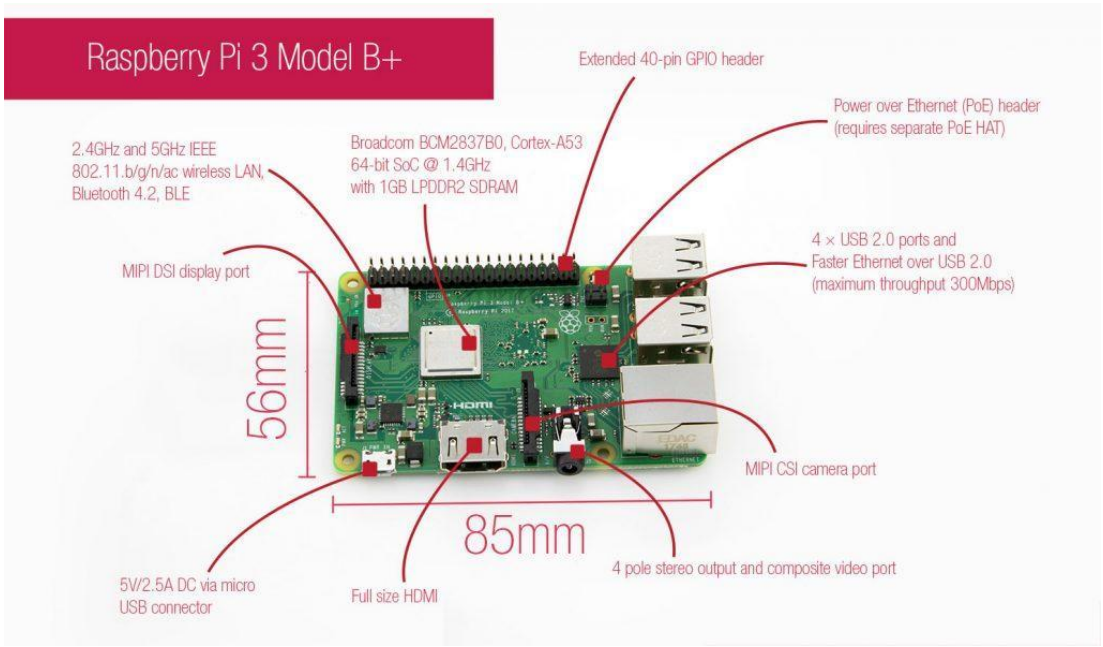

Fig 3.1: Raspberry Pi 3 Model B

The Raspberry Pi 3 Model B is a single-board computer developed by the Raspberry Pi Foundation. It features a quad-core ARM Cortex-A53 processor clocked at 1.4 GHz, 1GB of LPDDR2 RAM, built-in Wi-Fi and Bluetooth, and improved network capabilities.

It provides a low-cost, compact, and versatile computing platform for a variety of applications, including education, embedded systems, IoT, and DIY projects. The Raspberry Pi 3 Model B is an enhanced version of the Raspberry Pi 3 Model B, with several upgrades:

Improved Processor: 1.4 GHz Quad-Core ARM Cortex-A53 CPU, providing better performance.

Better Network Connectivity:

- Gigabit Ethernet (via USB 2.0 interface, max 300 Mbps) for faster wired networking.
- Dual-band 802.11ac Wi-Fi (2.4 GHz & 5 GHz) for better wireless connectivity.

Enhanced Power Management:

- Improved thermal performance for sustained workloads.
- Power-over-Ethernet (PoE) support (via a separate HAT).

Bluetooth:

- Bluetooth 4.2 / BLE (Bluetooth Low Energy) for IoT applications.

GPIO PINs:

- Same GPIO pinout as previous models, ensuring compatibility with existing accessories.

USB and Overcurrent Protection:

The Raspberry Pi 3 Model B features four USB 2.0 ports with built-in overcurrent protection. If an excessive current draw is detected, the system can shut down the affected port to protect the board and connected devices.

Physical Characteristics:

The Raspberry Pi 3 Model B maintains the same form factor as its predecessors, with dimensions of 85.6mm × 56.5mm. It features four mounting holes for easy attachment to enclosures or surfaces.

Table 3.1: Technical Specifications of Raspberry Pi 3 model B

Specification	Details
Processor	1.4 GHz Quad-Core ARM Cortex-A53 (Broadcom BCM2837B0)
RAM	1GB LPDDR2 SDRAM
Operating Voltage	5V via micro-USB
Power Input	5V/2.5A via micro-USB or GPIO header
Wireless Connectivity	Dual-band 802.11ac Wi-Fi, Bluetooth 4.2 / BLE
Ethernet	Gigabit Ethernet (via USB 2.0, max 300 Mbps)
USB Ports	4 × USB 2.0

GPIO Pins	40-pin GPIO header
Display Output	HDMI, MIPI DSI Display Port
Camera Interface	MIPI CSI Camera Port
Audio	3.5mm jack + HDMI
Storage	microSD card slot (supports SDHC and SDXC)
Graphics	Video Core IV GPU (supports OpenGL ES 1.1, 2.0, and Full HD 1080p video)
OS Support	Raspbian, Ubuntu, Windows 10 IoT Core, and other Linux-based distributions
Dimensions	85.6mm × 56.5mm × 17mm
Weight	~50g

Summary:

- Processor: 1.4 GHz Quad-Core ARM Cortex-A53
- Memory: 1GB LPDDR2 RAM
- Wireless: Dual-band 802.11ac Wi-Fi, Bluetooth 4.2
- Ethernet: Gigabit Ethernet (300 Mbps max)
- USB Ports: 4 × USB 2.0
- Power Supply: 5V/2.5A via micro-USB
- GPIO: 40-pin header
- Storage: microSD card slot
- OS Support: Raspbian, Ubuntu, Windows 10 IoT Core

3.1.1 Raspberry Pi 3 Model B PIN Mapping Table

The Raspberry Pi 3 Model B can be powered using a Micro USB power supply or via GPIO pins. It is recommended to use a 5V/2.5A power adapter for stable operation.

External power sources can also be used via the GPIO header. The board includes built-in voltage regulation to ensure stable operation.

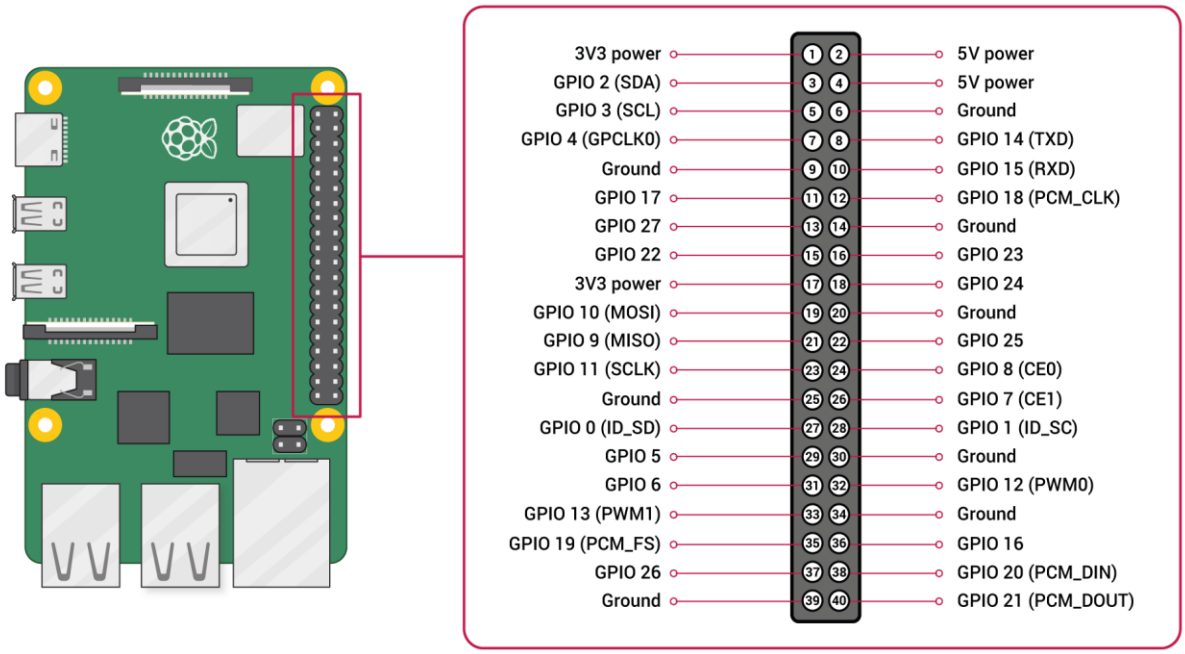

Fig 3.2: Raspberry Pi 3 Model B pin mapping table.

The power pins are as follows:

- **5V (Pins 2 & 4):** Direct 5V power supply for high-power components.
- **3V3 (Pins 1 & 17):** 3.3V regulated power output.
- **GND (Pins 6, 9, 14, 20, 25, 30, 34, 39):** Ground pins for circuit connections.
- **GPIO VIN:** Can be used to power the board via an external power source.

Memory & Storage

- RAM: 1GB LPDDR2 SDRAM.
- Storage: Uses a microSD card for operating system and file storage.
- Boot Options: Can boot from microSD, USB storage, or network (PXE boot).

Input and Output

The Raspberry Pi 3 Model B features a 40-pin GPIO header, which can be used for digital input/output, serial communication, PWM, and interfacing with various sensors and modules.

GPIO Pins

Each GPIO pin can be used as an input or output using programming languages like Python, C, or Scratch.

- GPIO Pins: 28 General Purpose I/O Pins.
- Max Current per Pin: 16mA.
- Voltage Level: 3.3V logic (do not apply 5V directly to GPIOs).
- Internal Pull-up/Pull-down Resistors: Available on GPIOs.

Communication Interfaces

The Raspberry Pi 3 Model B supports multiple communication protocols:

- **Serial (UART):**

  TX (GPIO 14, Pin 8) - Transmit data.

  RX (GPIO 15, Pin 10) - Receive data.

- **I2C (Inter-Integrated Circuit):**

  SDA (GPIO 2, Pin 3) - Data line.

  SCL (GPIO 3, Pin 5) - Clock line.

- **SPI (Serial Peripheral Interface):**

  MOSI (GPIO 10, Pin 19) - Master Out, Slave In.

  MISO (GPIO 9, Pin 21) - Master In, Slave Out.

  SCLK (GPIO 11, Pin 23) - Serial Clock.

  CE0 (GPIO 8, Pin 24), CE1 (GPIO 7, Pin 26) - Chip Enable.

PWM (Pulse Width Modulation)

PWM can be used to control motor speed, LED brightness, and other variable signals:

- PWM0 (GPIO 12, Pin 32)
- PWM1 (GPIO 13, Pin 33)

Networking & Connectivity

- Ethernet: Gigabit Ethernet over USB 2.0 (max 300 Mbps).
- Wi-Fi: Dual-band 2.4GHz / 5GHz IEEE 802.11ac.
- Bluetooth: Bluetooth 4.2, BLE (Bluetooth Low Energy).

USB & Display

- USB Ports: 4 x USB 2.0 ports (for peripherals like a keyboard, mouse, storage).

- HDMI Port: Full-size HDMI port for video output (supports up to 1080p).
- Audio Jack: 3.5mm audio/composite video jack.
- Camera Interface (CSI): For Raspberry Pi Camera Module.
- Display Interface (DSI): For connecting an LCD touchscreen.

Additional Features

- **LED Indicators:**

  ACT (Green LED): Indicates microSD activity.

  PWR (Red LED): Shows power status.

- **Reset & Power Pins:**

  Can be accessed via GPIO header for external reset switches.

Notes on Usage:

1. **Voltage Levels:** GPIO pins operate at 3.3V logic levels; applying 5V may damage the board.
2. **Current Limitations:** Each GPIO pin can source/sink a limited amount of current (typically 16mA max per pin, total 50mA).
3. **Configurable:** GPIOs can be programmed as input, output, PWM, or communication pins using Python, C, or other programming languages.
4. **Use Pull-up/Pull-down Resistors:** Some GPIOs need external pull-up/down resistors to ensure stable logic levels.

This GPIO layout makes the Raspberry Pi versatile for hardware-based projects such as IoT, robotics, automation, and electronics prototyping.

3.2 CAMERA MODULE

The Raspberry Pi Camera Module is a high-quality imaging accessory designed specifically for Raspberry Pi boards. It enables users to capture images and videos for various applications, such as surveillance, photography, machine learning, and computer vision. The camera module connects to the Raspberry Pi through the Camera Serial Interface (CSI), a high-speed connection that allows efficient data transfer for real-time imaging.

Fig 3.3: Camera Module

Over the years, Raspberry Pi has introduced multiple versions of its camera modules, each bringing improvements in resolution, functionality, and image quality. These camera modules include the Camera Module 3, the Camera Module 2, and the High-Quality Camera Module. Additionally, there are NoIR (No Infrared Filter) versions, which are designed to capture images in low-light conditions and are often used for night-vision applications.

The latest Camera Module 3, launched in 2023, features a 12-megapixel Sony IMX708 sensor with autofocus, allowing for sharper and more detailed images. This module is available in four variants, including standard and NoIR versions, as well as wide-angle variants that offer a 120-degree field of view.

The Camera Module 2, released earlier, comes with an 8-megapixel Sony IMX219 sensor and a fixed-focus lens. While it lacks autofocus and HDR capabilities, it remains a reliable option for various imaging tasks. This module also has a NoIR version for infrared-based applications.

For users seeking professional-grade photography and video capabilities, the Raspberry Pi High-Quality Camera Module is an ideal choice. It features a 12.3-megapixel Sony IMX477 sensor and is designed to support interchangeable lenses via C and CS-mounts.

The versatility of the Raspberry Pi Camera Module makes it a powerful tool for various applications. It is commonly used in security systems, where it can be paired with motion sensors to create smart surveillance cameras. It is also widely used in time-lapse photography, wildlife monitoring, and AI-driven robotics projects.

To enhance its functionality, the Raspberry Pi Camera Module can be paired with a range of accessories, including adjustable lenses for the High-Quality Camera, infrared LEDs for night vision, camera cases for protection, and USB capture devices that allow it to be used as a webcam.

Types of Raspberry Pi Camera Modules:

There are multiple versions of the Raspberry Pi Camera Module, including standard and infrared (NoIR) versions.

1. Camera Module 3 (Latest)

- **Variants:**

  Standard Camera Module 3 (Visible Light), Camera Module 3 NoIR (Infrared-sensitive for night vision), Camera Module 3 Wide (Wider field of view), Camera Module 3 NoIR Wide (Wide-angle with infrared).

- **Sensor:** Sony IMX708
- **Resolution:** 12 Megapixels
- **Autofocus:** Yes (PDAF – Phase Detection Autofocus)
- **Field of View:** Standard: 75°, Wide: 120°
- **HDR Support:** Yes, 22-pin CSI connector

2. Camera Module 2:

- **Sensor:** Sony IMX219
- **Resolution:** 8 Megapixels
- **Fixed Focus:** Yes
- **Field of View:** 62.2°, 15-pin CSI connector

3. High-Quality Camera Module:

- **Sensor:** Sony IMX477
- **Resolution:** 12.3 Megapixels
- **Lens Mount:** C and CS-Mount (interchangeable lenses)
- **Field of View:** Depends on the attached lens
- **Aperture Control:** Yes (adjustable lenses)
- **Tripod Mount:** Yes

4. NoIR Camera Modules:

NoIR (No Infrared Filter) versions of all Raspberry Pi camera modules allow for night vision by capturing infrared light. Ideal for low-light photography, security cameras, and wildlife monitoring when combined with infrared LEDs.

Camera Interface & Connectivity:

Uses a flat ribbon CSI-2 cable to connect to the Raspberry Pi's CSI port. Provides high-speed data transfer for image capture. Compatible Raspberry Pi Boards are 4, 3, 2, 1, Zero (with adapter for Pi Zero). Raspberry Pi 5: Uses a new 22-pin connector, requiring an adapter for older cameras.

Table 3.2: Camera Module Overview

Feature	**Camera Module 3**	**Camera Module 2**	**High-Quality Camera**
Sensor	Sony IMX708	Sony IMX219	Sony IMX477
Resolution	12MP	8MP	12.3MP
Autofocus	Yes	No (fixed focus)	Adjustable (manual lens)
Field of View	75° (Standard), 120° (Wide)	62.2	Depends on lens
Infrared (NoIR) Version	Yes	Yes	No
HDR (High Dynamic Range)	Yes	No	No
Connection	22-pin CSI	15-pin CSI	15-pin CSI
Lens Type	Fixed	Fixed	C/CS-mount lenses

3.3 LED (Light Emitting Diode)

Light Emitting Diodes (LEDs) are semiconductor devices that emit light when an electric current passes through them. They are widely used in various applications, ranging from household lighting to advanced electronic displays, automotive lighting, and industrial signaling. LEDs have revolutionized lighting technology due to their energy efficiency, long lifespan, and versatility.

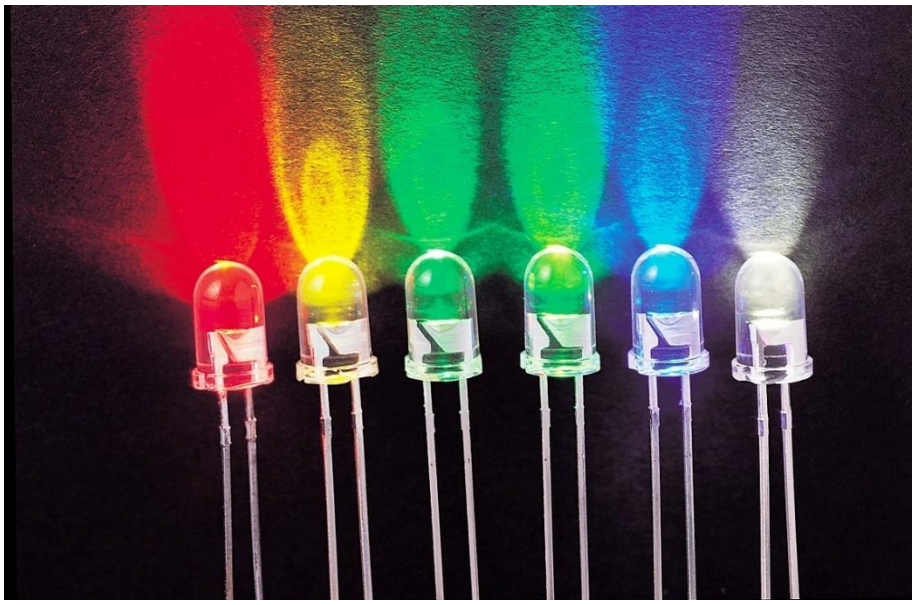

Fig 3.4: Light Emitting Diode (LED)

Light Emitting Diodes (LEDs) are an essential component of modern traffic light systems due to their high efficiency, durability, and low power consumption. Unlike traditional incandescent or halogen bulbs, LEDs offer better visibility, longer lifespan, and the ability to function effectively in various environmental conditions. In a traffic light system, LEDs are used to represent different signal colours—red, yellow (amber), and green—each serving a specific purpose in controlling vehicle and pedestrian movement.

Working Principle:

LEDs operate on the principle of electroluminescence, where electrons recombine with holes in a semiconductor material, releasing energy in the form of photons (light). Unlike traditional incandescent bulbs, which rely on heating a filament to produce light, LEDs generate light through a direct electronic process, making them highly efficient and durable.

Structure and Components:

An LED consists of several key components that work together to produce light:

1. Semiconductor Material: The heart of an LED is a semiconductor chip made of materials like gallium arsenide (GaAs) or gallium nitride (GaN), which determine the colour of light emitted.
2. P-N Junction: The LED has a positively charged (p-type) and a negatively charged (n-type) semiconductor layer that form a junction where light emission occurs.
3. Encapsulation: LEDs are enclosed in a transparent or diffused plastic or epoxy casing that protects the semiconductor material and helps direct the light output.
4. Lead Frame and Heat Sink: These components help in electrical connectivity and heat dissipation to ensure the LED operates efficiently without overheating.

Types of LEDs

LEDs come in various types based on their structure, power, and application:

1. Standard LEDs: These are small, low-power LEDs commonly used as indicator lights in electronic devices.
2. High-Power LEDs: These LEDs produce a much higher intensity of light and are used in applications such as street lighting and automotive headlights.
3. RGB LEDs: These LEDs combine red, green, and blue light sources in a single package to create various colours through mixing.
4. Infrared LEDs (IR LEDs): These LEDs emit infrared light, commonly used in remote controls and security cameras.
5. Ultraviolet LEDs (UV LEDs): Used in sterilization, counterfeit detection, and special lighting applications.
6. Organic LEDs (OLEDs): Made from organic compounds, these LEDs are flexible and used in high-end display screens like smartphones and televisions.
7. Smart LEDs: These LEDs have built-in controllers that allow them to be programmed for dynamic lighting effects, commonly used in smart homes and decorative lighting.

Types of LEDs Used in Traffic Lights

Traffic signals use high-intensity LEDs that are specifically designed to be visible in broad daylight and withstand harsh weather conditions. These LEDs come in three primary colours

This LED is responsible for indicating a stop signal. It is the most critical part of the traffic system as it ensures vehicles halt at intersections, pedestrian crossings, and railway crossings. Red LEDs have a high brightness level to ensure visibility from a distance and under direct sunlight.

The yellow LED serves as a warning signal, alerting drivers to slow down and prepare to stop. This LED is usually illuminated for a short duration and is positioned between the red and green LEDs to provide a transition between stopping and going.

The green LED signals vehicles and pedestrians to proceed safely. It is crucial in maintaining smooth traffic flow. The green LED is designed to have maximum visibility while preventing excessive glare, which can cause confusion among drivers.

Technical Specifications of LEDs in Traffic Lights:

The LEDs used in traffic signals have specific electrical and optical characteristics that ensure optimal performance. These include:

1. Wavelength and Colour Intensity: Red LEDs operate at a wavelength of around 620–630 nm, yellow LEDs at 585–595 nm, and green LEDs at 505–540 nm. These specific wavelengths ensure high visibility.
2. Power Consumption: LEDs in traffic lights are energy-efficient, typically consuming between 8W to 20W per unit, which is significantly lower than traditional bulbs.
3. Viewing Angle: The LEDs have a viewing angle between 30° and 60°, ensuring visibility from different angles without excessive glare.
4. Lifespan: High-quality LEDs used in traffic signals have a lifespan of approximately 50,000 to 100,000 hours, reducing maintenance costs.
5. Weather Resistance: Traffic light LEDs are designed to function in extreme weather conditions, from freezing cold to high heat, and are often encased in weatherproof housings.

LEDs have become a dominant lighting technology due to their superior energy efficiency, durability, and adaptability across various industries. With continuous advancements

in LED technology, their applications continue to expand, making them an integral part of modern lighting and electronic systems. Their ability to provide high brightness, long operational life, and environmental benefits makes them the preferred choice for sustainable and efficient lighting solutions worldwide.

3.4 REGULATED POWER SUPPLY

The power supply is a fundamental requirement for ensuring the proper operation of the traffic light signal system. The power source must provide stable and reliable DC voltage to drive the LEDs and other electronic components involved in the circuit.

The traffic light system requires a regulated DC power supply to ensure smooth and efficient operation. The required voltage level is obtained from a 12V-0-12V center-tapped transformer, which converts the mains AC voltage into a lower AC voltage suitable for rectification. The rectified DC voltage is then filtered and regulated to provide a stable 5V DC output for the microcontroller and LED drivers.

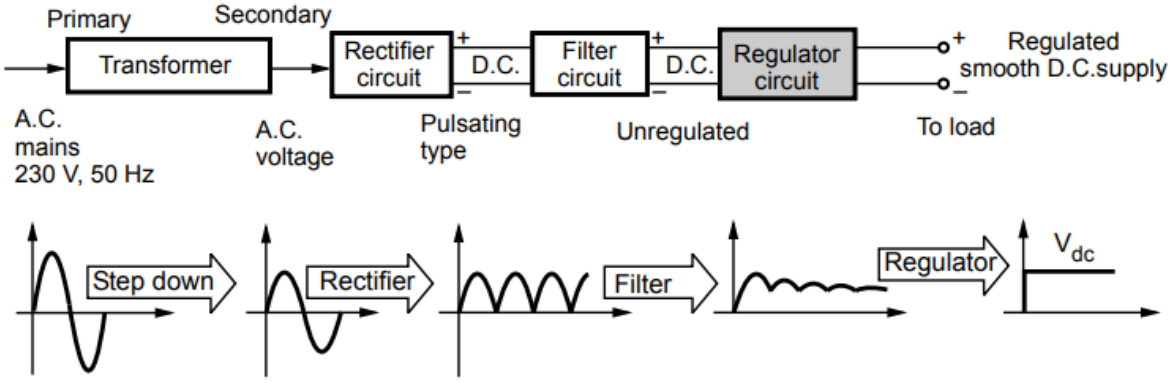

Fig 3.5: Internal Block Diagram of Power Supply

A 12V-0-12V step-down transformer is used to reduce the high AC voltage from the mains (230V AC) to a safer and usable level. The centre-tapped secondary winding provides two 12V AC outputs along with a ground (0V) reference. This allows for efficient full-wave rectification.

The AC voltage from the transformer is converted into DC voltage using a bridge rectifier composed of four diodes. The bridge rectifier allows current to flow in only one direction, converting both positive and negative halves of the AC cycle into a pulsating DC output. This rectified voltage still contains ripples, which must be filtered to obtain a smooth DC output.

A capacitor filter is used to smooth the rectified DC voltage by reducing fluctuations and maintaining a steady voltage level. Typically, a 1000µF electrolytic capacitor is used at this stage to provide effective filtering and minimize voltage ripple.

Most microcontrollers used in traffic light systems operate at 5V and draw minimal current (10-50mA). Each LED typically operates at 20mA per colour, and multiple LEDs require higher current. Based on the number of LEDs and other electronic components, the total power consumption is calculated, ensuring that the 7805 regulators can handle the load without overheating.

CHAPTER 4

SOFTWARE SPECIFICATIONS

The software tools which are used in this project are Raspberry pi OS, Thonny IDE, programming language used in this project is Python and YOLO (You Only Look Once) model.

4.1 RASPBERRYPI OS

Raspberry Pi OS is a Unix-like operating system based on the Debian Linux distribution for the Raspberry Pi family of compact single-board computers. Raspbian was developed independently in 2012, became the primary operating system for these boards since 2013, was originally optimized for the Raspberry Pi 1 and distributed by the Raspberry Pi Foundation. In 2020, the Raspberry Pi Foundation renamed Raspbian to Raspberry Pi OS.

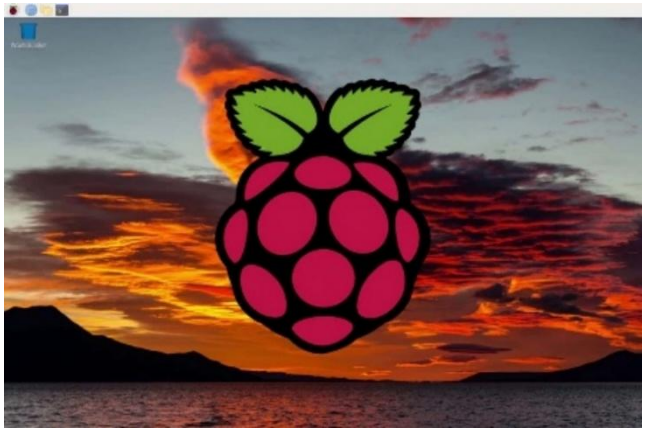

Fig 4.1: Raspberry Pi OS

4.1.1 HISTORY:

Raspberry Pi OS was first developed by Mike Thompson and Peter Green as Raspbian, an independent and unofficial port of Debian to the Raspberry Pi. The first build was released on July 15, 2012.As the Raspberry Pi had no officially provided operating system at the time, the Raspberry Pi Foundation built on the work by the Raspbian project and began producing and releasing their own version of the software. The Foundation's first release of Raspbian, which now referred both to the community project as well as the official operating system, was announced on September 10, 2013.

On May 28, 2020, the Raspberry Pi Foundation announced a beta 64-bit version. However, this version was not based on Raspbian, instead taking its user space software from Debian GNU/Linux. When the Foundation did not want to use the name Raspbian to refer to

software that was not based on the Raspbian project, the name of the officially provided operating system was changed to Raspberry Pi OS.

4.1.2 FEATURES:

User Interface:

Raspberry Pi OS has a desktop environment, PIXEL (short for Pi Improved Xwindows Environment, Lightweight), based on LXDE, which looks similar to many common desktops, such as macOS and Microsoft Windows. The desktop has a background image. A menu bar is positioned at the top and contains an application menu and shortcuts to a web browser (Chromium), file manager, and terminal. The other end of the menu bar shows a Bluetooth menu, Wi-Fi menu, volume control, and clock. The desktop can also be changed from its default appearance, such as repositioning the menu bar.

Package management:

Packages can be installed via APT, the Recommended Software app, and by using the Add/Remove Software tool, a GUI wrapper for APT.

Components:

PCManFM is a file browser allowing quick access to all areas of the computer, and was redesigned in the first Raspberry Pi OS Buster release (2019-06-20). Raspberry Pi OS originally distributed the web browser Epiphany, but switched to Chromium with the launch of its redesigned desktop. The built-in browser comes preinstalled with uBlock Origin and h264ify. Raspberry Pi OS comes with many beginner IDEs, such as Thonny Python IDE, Mu Editor, and Greenfoot. It also ships with educational software, such as Scratch and Bookshelf.

4.1.3 INSTALLATION:

Installing an Operating System:

To use your Raspberry Pi, you'll need an operating system. By default, Raspberry Pi's check for an operating system on any SD card inserted in the SD card slot. Depending on your Raspberry Pi model, you can also boot an operating system from other storage devices, including USB drives, storage connected via a HAT, and network storage.

To install an operating system on a storage device for your Raspberry Pi, you'll need:

- a computer you can use to image the storage device into a boot device

- a way to plug your storage device into that computer

Most Raspberry Pi users choose microSD cards as their boot device. We recommend installing an operating system using Raspberry Pi Imager.

Install using Imager:

You can install Imager in the following ways:

- Download the latest version from raspberrypi.com/software and run the installer.
- Install it from a terminal using your package manager, e.g. sudo apt install rpi-imager.

  Once you've installed Imager, launch the application by clicking the Raspberry Pi Imager icon or running rpi-imager.
- Click Choose device and select your Raspberry Pi model from the list.
- Next, click Choose OS and select an operating system to install. Imager always shows the recommended version of Raspberry Pi OS for your model at the top of the list.
- Connect your preferred storage device to your computer. For example, plug a microSD card in using an external or built-in SD card reader. Then, click Choose storage and select your storage device.

Next, click Next.

- In a popup, Imager will ask you to apply OS customisation. We strongly recommend configuring your Raspberry Pi via the OS customisation settings. Click the Edit Settings button to open OS customisation.

Write:

When you've finished entering OS customisation settings, click Save to save your customisation. Then, click Yes to apply OS customisation settings when you write the image to the storage device. Finally, respond Yes to the "Are you sure you want to continue?" popup to begin writing data to the storage device.

If you want to live especially dangerously, you can click cancel verify to skip the verification process. When you see the "Write Successful" popup, your image has been completely written and verified. You're now ready to boot a Raspberry Pi from the storage device!

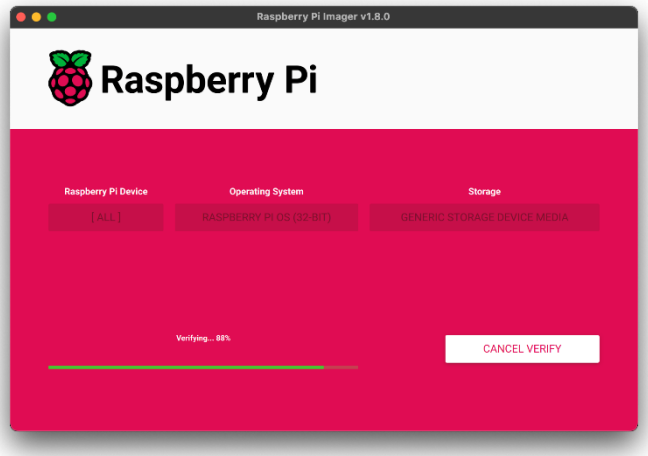

Fig 4.2: Installation of Raspberry Pi OS

4.1.4 SETUP RASPBERRY PI:

After installing an operating system image, connect your storage device to your Raspberry Pi. First, unplug your Raspberry Pi's power supply to ensure that the Raspberry Pi is powered down while you connect peripherals. If you installed the operating system on a microSD card, you can plug it into your Raspberry Pi's card slot now. If you installed the operating system on any other storage device, you can connect it to your Raspberry Pi now.

Then, plug in any other peripherals, such as your mouse, keyboard, and monitor.

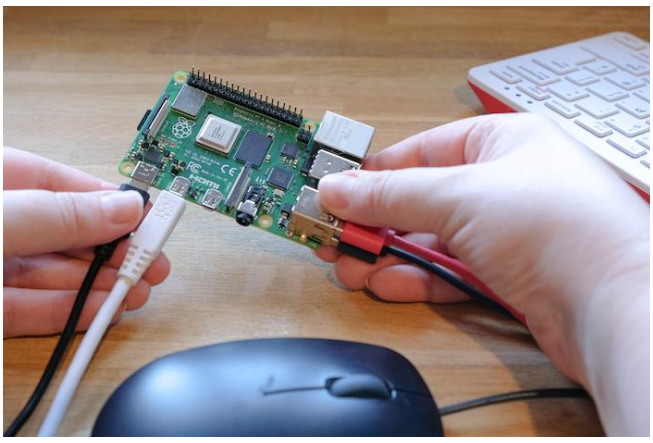

Fig 4.3: Setup Raspberry Pi with Peripherals

Finally, connect the power supply to your Raspberry Pi. You should see the status LED light up when your Pi powers on. If your Pi is connected to a display, you should see the boot screen within minutes.

Configuration on first boot:

If you used OS customisation in Imager to preconfigure your Raspberry Pi, congratulations! Your device is ready to use. Proceed to next steps to learn how you can put your Raspberry Pi to good use. If your Raspberry Pi does not boot within 5 minutes, check the status LED. If it's flashing, see the LED warning flash codes for more information. If your Pi refuses to boot, try the following mitigation steps:

- if you used a boot device other than an SD card, try booting from an SD card
- re-image your SD card; be sure to complete the entire verify step in Imager
- update the bootloader on your Raspberry Pi, then re-image your SD card.

If you chose to skip OS customisation in Imager, your Raspberry Pi will run a configuration wizard on first boot. You need a monitor and keyboard to navigate through the wizard; a mouse is optional.

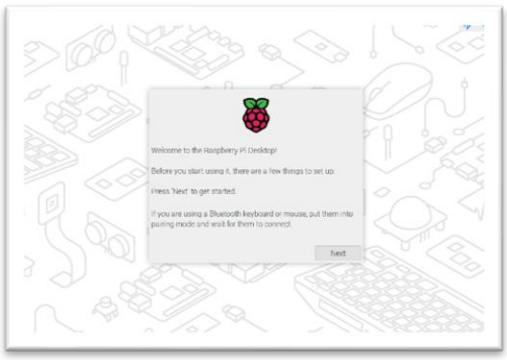

Fig 4.4: Initial Configuration of Raspberry Pi OS

4.1.5 ADVANTAGES:

1. **Optimized for Raspberry Pi**: Raspberry Pi OS is tailored to work seamlessly with Raspberry Pi hardware, ensuring compatibility and efficient performance.
2. **Free and Open Source**: It is free to download and use, and its open-source nature allows users to customize it according to their needs.
3. **Lightweight**: The OS is lightweight, making it ideal for the limited resources of Raspberry Pi devices.
4. **Educational Tools**: It comes pre-installed with tools like Scratch and Python, making it perfect for learning programming and electronics. A large community support.

4.2 VSCODE IDE

Visual Studio Code (VS Code) is a lightweight and powerful code editor developed by Microsoft, known for its speed and versatility. It offers features like IntelliSense, built-in Git support, debugging tools, and a vast extension marketplace, making it ideal for developers. Free and open-source, VS Code supports multiple languages and platforms, enhancing productivity across various coding environments.

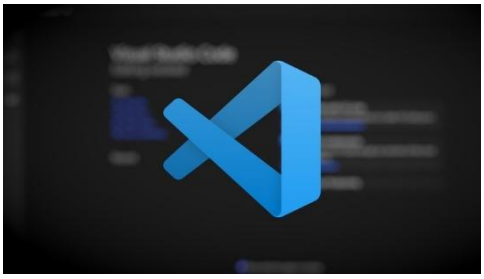

Fig 4.5: VSCODE IDE

4.2.1 FEATURES:

- Cross-Platform Compatibility
- Integrated Git Support
- Extensive Extension Marketplace
- Built-in Debugging
- Customizability
- Powerful Search & Navigation
- Integrated Terminal

4.2.2 ADVANTAGES:

1. **Lightning-Fast Performance** Starts quickly and runs efficiently even on modest machines.
2. **Free & Open Source**: No cost, with a vibrant community contributing to its development.
3. **Seamless Integration**: Works well with popular development tools and frameworks.
4. **Robust Debugging Tools**: Helps developers write error-free code.
5. **Thriving Community & Documentation**: Extensive resources and support available.

4.3 PYTHON (programming language)

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

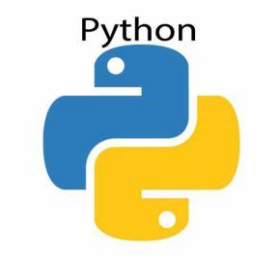

Fig 4.6: Python Programming Language

Python is dynamically type-checked and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

4.3.1 HISTORY:

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life" (BDFL), a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker (he has since come out of retirement and is self-titled "BDFL-emeritus"). In January 2019, active Python core developers elected a five-member Steering Council to lead the project.

The name Python is said to come from the British comedy series Monty Python's Flying Circus.

4.3.2 SYNTAX & SEMANTICS:

Python is meant to be an easily readable language. Its formatting is visually uncluttered and often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are allowed but rarely used. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation:

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents its semantic structure.

Statements and control flow:

Python's statements include:

- The assignment statement, using a single equals sign " = "
- The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else if)
- The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block
- The while statement, which executes a block of code as long as its condition is true.

Expressions:

- The +, -, and * operators for mathematical addition, subtraction, and multiplication are similar to other languages, but the behavior of division differs. There are two types of divisions in Python: floor division (or integer division) // and floating-point / division. Python uses the ** operator for exponentiation.
- Python uses the + operator for string concatenation. Python uses the * operator for duplicating a string a specified number of times.

Methods:

Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self-parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, Ruby). Python also provides methods, often called dunder methods (due to their names beginning and ending with double-underscores), to allow user-defined classes to modify how they are handled by native operations including length, comparison, in arithmetic operations and type conversion.

4.4 YOLO (YOU ONLY LOOK ONCE) MODEL

You Only Look Once (YOLO) is a series of real-time object detection systems based on convolutional neural networks. First introduced by Joseph Redmon et al. in 2015, YOLO has undergone several iterations and improvements, becoming one of the most popular object detection frameworks.

4.4.1 OVERVIEW:

Compared to previous methods like R-CNN and OverFeat, instead of applying the model to an image at multiple locations and scales, YOLO applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Versions:

There are two parts to the YOLO series. The original part contained YOLOv1, v2, and v3, all released on a website maintained by Joseph Redmon.

YOLOv4:

Subsequent versions of YOLO (v4, v5, etc.) have been developed by different researchers, further improving performance and introducing new features. These versions are not officially associated with the original YOLO authors but build upon their work. As of 2024, there are versions up to YOLO11.

4.4.2 OBJECT DECTION:

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

Uses:

It is widely used in computer vision tasks such as image annotation, vehicle counting, activity recognition, face detection, face recognition, video object co-segmentation. It is also used in tracking objects, for example tracking a ball during a football match, tracking movement of a cricket bat, or tracking a person in a video.

Often, the test images are sampled from a different data distribution, making the object detection task significantly more difficult. To address the challenges caused by the domain gap between training and test data, many unsupervised domain adaptation approaches have been proposed. A simple and straightforward solution for reducing the domain gap is to apply an image-to-image translation approach, such as cycle-GAN.

Concept:

Every object class has its own special features that help in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a particular distance from a point (i.e. the centre) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin color and distance between eyes can be found.

Benchmarks:

For simultaneous object localization and classification, a true positive is one where the class label is correct, and the bounding box has an IoU exceeding the threshold. Simultaneous object localization and classification is benchmarked by the **mean average precision** (mAP).

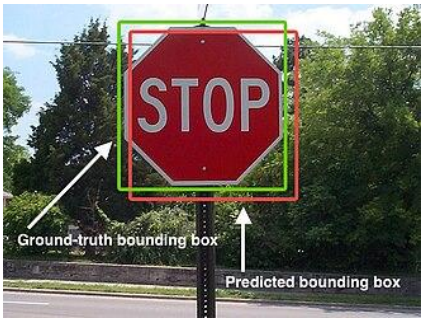

Fig 4.7: Detecting Objects by YOLO AI model

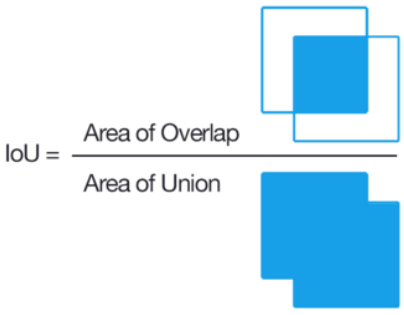

Fig 4.8: mAP Calculation

Intersection over union as a similarity measure for object detection on images – an important task in computer vision

Methods:

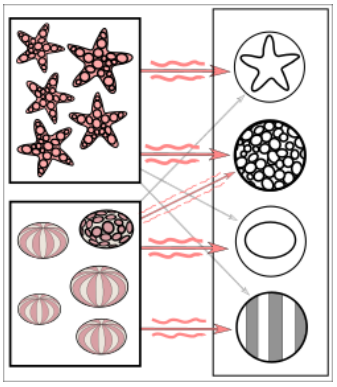

Fig 4.9: Initial Training of YOLO model

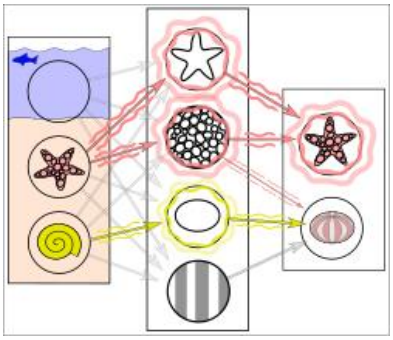

Fig 4.10: Identifying the Objects

Simplified example of training a neural network in object detection: The network is trained by multiple images that are known to depict starfish and sea urchins, which are correlated with "nodes" that represent visual features. The starfish match with a ringed texture and a star outline, whereas most sea urchins match with a striped texture and oval shape. However, the instance of a ring textured sea urchin creates a weakly weighted association between them.

Methods for object detection generally fall into either neural network-based or non-neural approaches. For non-neural approaches, it becomes necessary to first define features using one of the methods below, then using a technique such as support vector machine (SVM) to do the classification. On the other hand, neural techniques are able to do end-to-end object detection without specifically defining features, and are typically based on convolutional neural networks (CNN).

CHAPTER 5

RESULTS

5.1 DETECTING VEHICLES IN TWO LANE ROAD

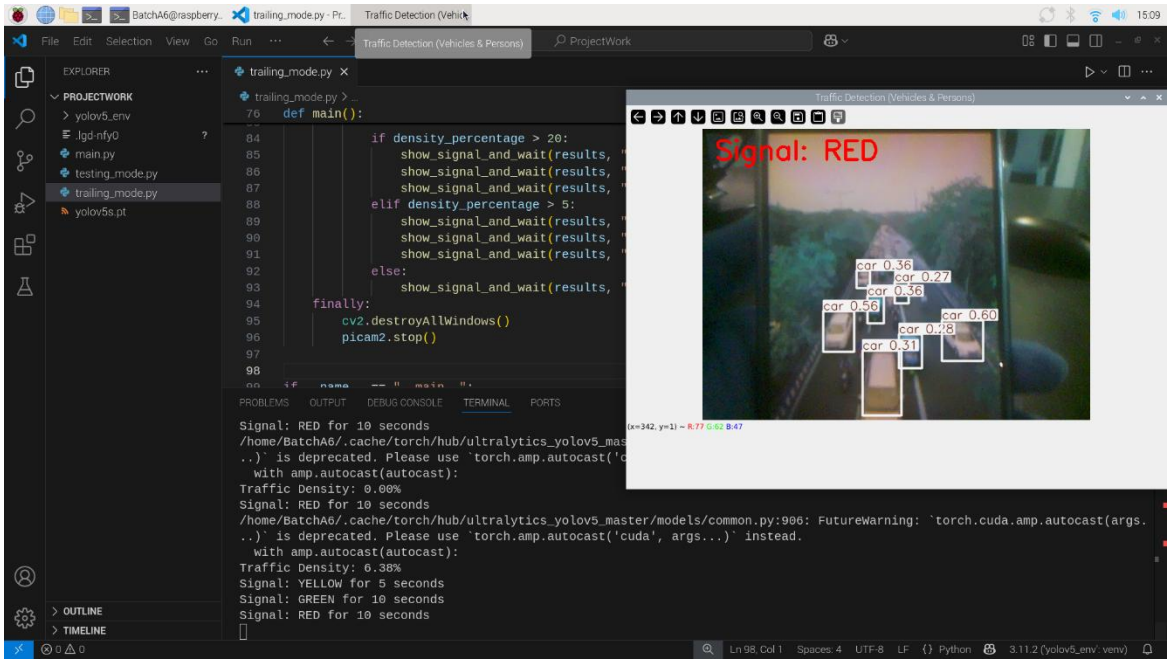

Fig 5.1: Timing for RED Signal in Two Lane Roads

The YOLO AI Model detecting the vehicles in the traffic and setting the timing for signals according to traffic. In this scenario, Vehicles are detecting When the traffic light is in **"RED COLOR".**

5.2 ADJUST TIMING FOR THE SIGNALS

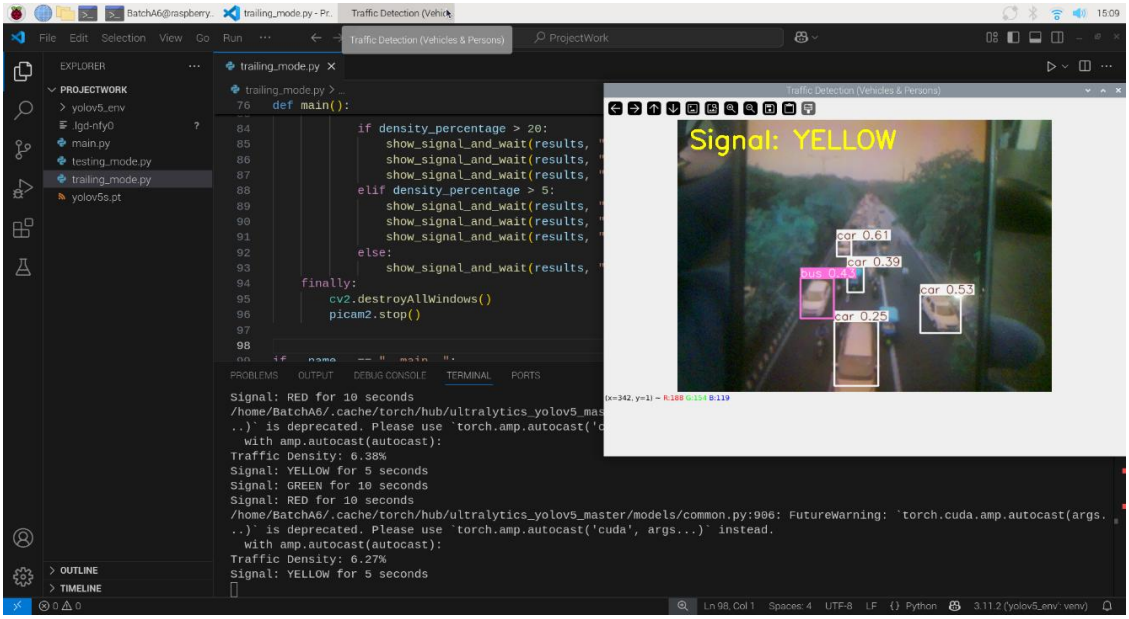

Fig 5.2: Timing for Yellow Signal in Two Lane Roads

In this scenario, Vehicles are detecting When the traffic light is in **"Yellow Color".** It is indicating the Vehicles are need to be ready to go. This Light indication will be only when before the Green Signal. Other wise it will not work.

5.3 APPLY TIMING ON THE TRAFFIC FOR GREEN SIGNAL

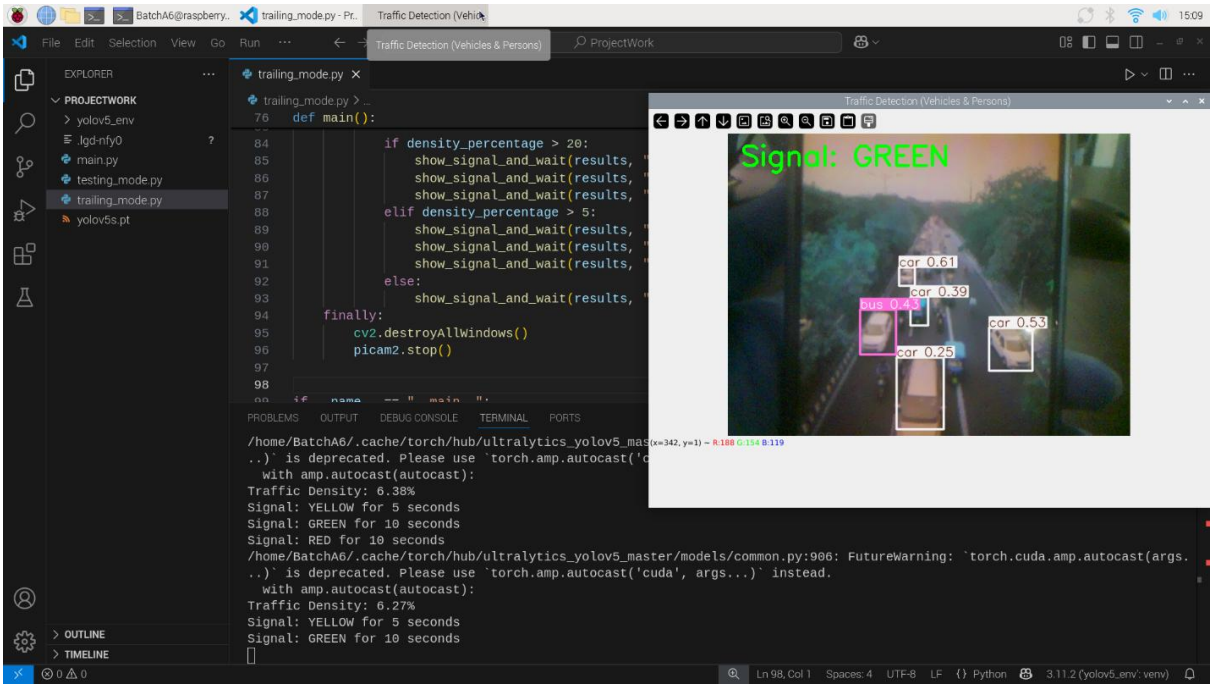

Fig 5.3: Timings for Green Signal Two Lane Roads

A green signal is prominently displayed, indicating that the current traffic density is low enough to allow vehicle movement. Supporting this, the terminal at the bottom logs traffic density percentages such as 6.38% and 6.27%, and shows corresponding signal changes like "YELLOW for 5 seconds" followed by "GREEN for 10 seconds."

Overall, the system demonstrates a well-implemented real-time traffic management solution. It combines computer vision and control logic to dynamically adjust signals based on vehicle detection, contributing toward intelligent traffic systems for smart city applications.

5.4 INITIAL HARDWARE SETUP

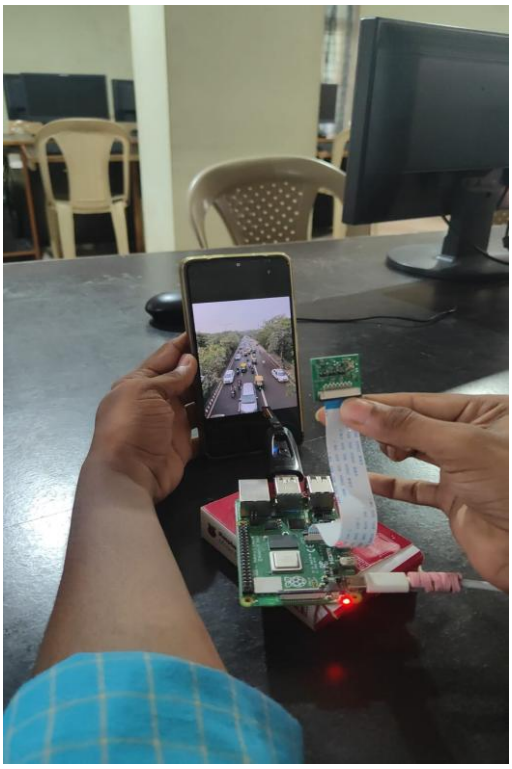

Fig 5.4: Testing the YOLO Model for Traffic Analysis

This system likely feeds detection results into a logic module that controls traffic lights based on real-time congestion levels. The presence of a monitor, development environment, and USB connections further suggests active development and testing. Overall, the image showcases a compact and efficient edge-computing solution for intelligent traffic management, demonstrating practical application of computer vision in smart city infrastructure.

CHAPTER 6

CONCLUSION

The Traffic Density-Based Signal Timing Control System represents a significant advancement in urban traffic management. By leveraging real-time traffic data through the integration of sensors, IoT technology, and machine learning algorithms, the system dynamically adjusts traffic signal timings to optimize vehicle flow. This intelligent approach reduces congestion, shortens travel times, and minimizes fuel consumption, thereby contributing to lower greenhouse gas emissions. In doing so, it not only enhances transportation efficiency but also supports environmental sustainability. Designed with scalability in mind, this system offers a practical and forward-looking solution to meet the evolving traffic demands of rapidly growing urban areas.

CHAPTER 7

FUTURE ENHANCEMENTS

Developing an efficient traffic management system is the need of the hour and with no generalized form is a troublesome task and requires a lot of research. The unavailability of a dataset in any Indian language and having no algorithm that can work upon any language other than English was a big constraint that may be worked upon in the future. As traffic rules and regulation and how cars behave on the street is different in different, it's a challenge to make a generalized model that works for all. A different approach for the data could be taken so that different categorizations can be done for the same and can work upon the early Traffic management system. In further research aggregation method and ensemble could be implemented in a much better way to attain higher accuracy.

REFERENCES

BIBILOGRAPHY

[1] M. M. Gandhi, D. S. Solanki, R. S. Daptardar and N. S. Baloorkar, "Smart Control of Traffic Light Using Artificial Intelligence," 2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE), 2020, pp. 1-6, doi: 10.1109/ICRAIE51050.2020.9358334.

[2] Khiang, Kok & Khalid, Marzuki & Yusof, Rubiyah. (1997). Intelligent Traffic Lights Control By Fuzzy Logic. Malaysian Journal of Computer Science. 9. 29-35.

[3] M. H. Malhi, M. H. Aslam, F. Saeed, O. Javed and M. Fraz, "Vision Based Intelligent Traffic Management System," 2011 Frontiers of Information Technology, 2011, pp. 137 141, doi: 10.1109/FIT.2011.33.

[4] A. Vogel, I. Oremović, R. Šimić and E. Ivanjko, "Improving Traffic Light Control by Means of Fuzzy Logic," 2018 International Symposium ELMAR, 2018, pp. 51-56, doi: 10.23919/ELMAR.2018.8534692.

[5] T. Osman, S. S. Psyche, J. M. Shafi Ferdous and H. U. Zaman, "Intelligent traffic management system for cross section of roads using computer vision," 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), 2017, pp. 1-7, doi: 10.1109/CCWC.2017.7868350.

[6] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. Communications Surveys Tutorials, IEEE, 13(4):584–616, 2011.

[7] L. Villas, A. Boukerche, R. Araujo, A. Loureiro, and J. Ueyama. Network partition-aware geographical data dissemination. In Communications (ICC), 2013 IEEE International Conference on, pages 1439–1443, 2013.

[8] A. M. Souza, G. Maia, and L. A. Villas. Add: A data dissemination solution for highly dynamic highway environments. In Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on, pages 17–23, Aug 2014. 23

[9] Albaladejo, C.; Sánchez, P.; Iborra, A.; Soto, F.; López, J.A.; Torres, R. Wireless Sensor Networks for Oceanographic Monitoring: A Systematic Review. Sens. 2010, 10, 6948 6968. [CrossRef] [PubMed]

[10] A. Vahidi and A. Eskandarian, “Research advances in intelligent collision avoidance and adaptive cruise control,” IEEE Trans. Intell. Transp. Syst., vol. 4, no. 3, pp. 143–153, Sep. 2003.

[11] A. Gonzalez, M. A. Garrido, D. F. Llorca, M. Gavilan, J. P. Fernandez, P. F. Alcantarilla, I. Parra, F. Herranz, L. M. Bergasa, M. A. Sotelo, and P. Revenga de Toro, "Automatic traffic signs and panels inspection system using computer vision," IEEE Trans. Intell. Transp. Syst., vol. 12, no. 2, pp. 485–499, Jun. 2011.

[12] VISSIM Microscopic Traffic and Transit Simulation User Manual, V.3.70. PTV AG, Karlsruhe, Germany, 2004.

[13] Mahmassani, H. S., S. Peeta, T. Hu, and A. Ziliaskopoulos. Dynamic Traffic Assignment with Multiple-User Classes for Real-Time ATIS/ATMS Applications. In Proc., Advanced Traffic Management Conference, St. Petersburg, Fla., 1993.

[14] User's Guide, DynaMIT & DynaMIT-P, Version 0.9: Development of a Deployable Real-Time Dynamic Traffic Assignment System. Massachusetts Institute of Technology, Cambridge, 2000.

[15] Saad, A. A., El Zouka, H. A., & Al-Soufi, S. A. (2016, March). Secure and Intelligent Road Traffic Management System Based on RFID Technology. In Computer Applications & Research (WSCAR), 2016 World Symposium on (pp. 41-46). IEEE.

SOURCE CODE:

```
import cv2

import time

import torch

from picamera2 import Picamera2

import numpy as np

import RPi.GPIO as GPIO


# Load YOLOv5 model

model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

model.conf = 0.25  # Confidence threshold


# GPIO Pin Setup

GREEN_PIN_A = 17

YELLOW_PIN_A = 27

RED_PIN_A = 22

GREEN_PIN_B = 23

YELLOW_PIN_B = 24

RED_PIN_B = 25


GPIO.setmode(GPIO.BCM)

GPIO.setup([GREEN_PIN_A, YELLOW_PIN_A, RED_PIN_A, GREEN_PIN_B,
YELLOW_PIN_B, RED_PIN_B], GPIO.OUT)


# Initialize PiCamera (Direction A)

picam2 = Picamera2()

picam2.preview_configuration.main.size = (640, 480)

picam2.preview_configuration.main.format = "RGB888"

picam2.configure("preview")
```

```
picam2.start()


# Initialize USB Webcam (Direction B)
cap_b = cv2.VideoCapture(0)
cap_b.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap_b.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)


FRAME_WIDTH = 640
FRAME_HEIGHT = 480
ALLOWED_CLASS_IDS = [0, 2, 3, 5, 7]


def draw_signal_on_frame(frame, signal_status):
    color_map = {
        "GREEN": (0, 255, 0),
        "YELLOW": (0, 255, 255),
        "RED": (0, 0, 255)
    }
    overlay = frame.copy()
    overlay.setflags(write=1)
    color = color_map.get(signal_status, (255, 255, 255))
    cv2.putText(overlay, f"Signal: {signal_status}", (20, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1.5, color, 3)
    return overlay


def set_gpio_signal(direction, signal):
    pins = {
        'A': {'RED': RED_PIN_A, 'YELLOW': YELLOW_PIN_A, 'GREEN':
GREEN_PIN_A},
        'B': {'RED': RED_PIN_B, 'YELLOW': YELLOW_PIN_B, 'GREEN':
GREEN_PIN_B}
```

```
    }
    for pin in pins[direction].values():
        GPIO.output(pin, GPIO.LOW)
    if signal in pins[direction]:
        GPIO.output(pins[direction][signal], GPIO.HIGH)
        print(f"Direction {direction}: {signal} light ON")

def process_frame(frame):
    results = model(frame)
    total_area = 0
    frame_area = FRAME_WIDTH * FRAME_HEIGHT

    for *box, conf, cls in results.xyxy[0]:
        label = int(cls)
        if label in ALLOWED_CLASS_IDS:
            x1, y1, x2, y2 = map(int, box)
            area = (x2 - x1) * (y2 - y1)
            total_area += area

    density_percentage = (total_area / frame_area) * 100
    return density_percentage, results

def calculate_dynamic_green_time(density):
    base_time = 10 # seconds
    max_time = 25 # seconds
    return min(max_time, int(base_time + (density / 2)))

def show_signal_with_overlay(results, signal, duration, window_name,
direction):
```

```
    set_gpio_signal(direction, signal)
    results.render()
    img = np.asarray(results.ims[0])
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    overlay = draw_signal_on_frame(img, signal)
    start_time = time.time()
    while time.time() - start_time < duration:
        cv2.imshow(window_name, overlay)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            exit()


def main():
    try:
        while True:
            frame_a = picam2.capture_array()
            ret_b, frame_b = cap_b.read()
            if not ret_b:
                print("Failed to read from webcam")
                break


            frame_a_resized = cv2.resize(frame_a, (FRAME_WIDTH,
FRAME_HEIGHT))
            frame_b_resized = cv2.resize(frame_b, (FRAME_WIDTH,
FRAME_HEIGHT))


            density_a, results_a = process_frame(frame_a_resized)
            density_b, results_b = process_frame(frame_b_resized)


            print(f"Density A: {density_a:.2f}%, Density B:
{density_b:.2f}%")
```

```
            green_time_a = calculate_dynamic_green_time(density_a)
            green_time_b = calculate_dynamic_green_time(density_b)


            if density_a >= density_b:
                show_signal_with_overlay(results_b, "RED", 5,
"Direction B", 'B')
                show_signal_with_overlay(results_a, "YELLOW", 5,
"Direction A", 'A')
                show_signal_with_overlay(results_a, "GREEN",
green_time_a, "Direction A", 'A')
                show_signal_with_overlay(results_a, "RED", 5,
"Direction A", 'A')
                show_signal_with_overlay(results_b, "YELLOW", 5,
"Direction B", 'B')
                show_signal_with_overlay(results_b, "GREEN",
green_time_b, "Direction B", 'B')
            else:
                show_signal_with_overlay(results_a, "RED", 5,
"Direction A", 'A')
                show_signal_with_overlay(results_b, "YELLOW", 5,
"Direction B", 'B')
                show_signal_with_overlay(results_b, "GREEN",
green_time_b, "Direction B", 'B')
                show_signal_with_overlay(results_b, "RED", 5,
"Direction B", 'B')
                show_signal_with_overlay(results_a, "YELLOW", 5,
"Direction A", 'A')
                show_signal_with_overlay(results_a, "GREEN",
green_time_a, "Direction A", 'A')
    finally:
        cap_b.release()
        cv2.destroyAllWindows()
```

```
        picam2.stop()
        GPIO.cleanup()


if __name__ == "__main__":
    main()
```