SAVONIA

THESIS – BACHELOR'S DEGREE
SELECT FIELD OF STUDY

SMART TRAFFIC LIGHTS

An IoT-Based Adaptive Traffic Management System

AUTHOR Anyaegbu Chiemelie

SAVONIA UNIVERSITY OF APPLIED SCIENCES THESIS
Abstract

Field of Study Internet of Things	
Degree Programme Bachelor of Engineering in Internet of Things	
Author Anyaegbu Chiemelie	
Title of Thesis SMART TRAFFIC LIGHTS (An IoT-Based Adaptive Traffic Management System)	
Date April 2025	Pages/Appendices 46
Client Organisation /Partners	

Traffic congestion has been a pressing issue in many developing countries, including Nigeria, due to outdated infrastructure and static traffic control systems. This thesis presents the design and implementation of an IoT-based smart traffic light system aimed at improving real-time traffic management through adaptive signal control. Using microcontrollers, sensors, and cloud computing, the system detected real-time traffic density, pedestrian movement, and emergency vehicles to dynamically adjust traffic signals. The study integrated Passive Infrared (PIR) sensors, sound sensors, and ESP32 microcontrollers with the Blynk IoT platform for cloud-based monitoring and control. The smart system was tested in a simulated environment, and its performance was benchmarked against conventional fixed-timer traffic lights. Results from controlled simulations demonstrate that the smart system significantly outperforms conventional fixed-timer systems. Specifically, the smart traffic light reduced vehicle wait times from an average of 18 seconds to 10 seconds, and decreased emergency vehicle signal response times from ~9 seconds to just 2 seconds. Sensor accuracy reached 96.4% for the PIR sensor and 94.7% for the sound sensor. Additionally, cloud-based monitoring allowed for a dashboard update interval of 30–50 seconds, with data transmission delays under 5 seconds.
Despite encountering challenges related to sensor calibration and network reliability, the study confirmed that IoT-based traffic systems provided a more efficient, responsive, and scalable solution for urban traffic control. This thesis presents a practical model for future deployment in smart city infrastructures, supporting sustainable urban mobility through data-driven, intelligent traffic management.

Keywords
Smart Traffic System, IOT, Adaptive Traffic Control, Blynk Cloud Management

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Overview of IoT-Based Smart Traffic Management Systems

Traffic congestion remains a significant challenge in urban areas worldwide, particularly in developing nations such as Nigeria, where rapid urbanization and increasing vehicle ownership have placed immense pressure on existing road networks. Conventional traffic management systems, which operate on fixed signal timing, often fail to adapt to real-time traffic fluctuations, leading to inefficient traffic flow, increased fuel consumption, and environmental pollution.

The integration of Internet of Things (IoT) technologies into traffic management has emerged as a viable solution to these inefficiencies. IoT-based smart traffic management systems use connected sensors, real-time data analytics, and cloud computing to dynamically adjust signal timings, optimize traffic flow, and improve overall transportation efficiency. These systems use a network of embedded sensors, microcontrollers, and cloud-based platforms to collect real-time traffic data and make intelligent decisions for adaptive signal control.

1.2 Importance of IoT, Sensors, and Cloud Computing in Traffic Control

IoT, sensors, and cloud computing play a critical role in traffic control by enabling real-time monitoring of traffic conditions, allowing for dynamic adjustments to traffic signals based on current data, which ultimately reduces congestion, improves road safety, and optimizes traffic flow throughout a city or region, essentially creating a "smart traffic management system" that reacts to changing conditions in real-time (Kheder & Mohammed 2024).

IoT technology provides a connected infrastructure where smart traffic signals can communicate with vehicles, road sensors, and cloud platforms to dynamically adjust traffic light timings based on real-time traffic density. IoT devices enable traffic authorities to collect live traffic data, identify congestion hotspots, and take proactive measures to manage vehicle movement efficiently. One of the major applications of IoT in traffic management is vehicle-to-infrastructure (V2I) communication, where vehicles send real-time data (speed, position, and congestion levels) to traffic control systems, allowing for intelligent traffic signal adjustments (Cai et al. 2013, 355).

Sensors are fundamental to IoT-based traffic systems as they detect and relay critical traffic data. These sensors work in conjunction with microcontrollers to optimize traffic signals, reducing unnecessary delays and enhancing road safety. Commonly used sensors in smart traffic management include:

1. Infrared (IR) sensors: This sensor is used to detect vehicle presence and movement at intersections (de Oliveira et al. 2021).
2. Ultrasonic Sensors: Measure vehicle distance and speed to determine congestion levels (de Oliveira et al. 2021).
3. Radio Frequency Identification (RFID): Helps in automatic vehicle identification for traffic monitoring (Cai et al. 2013, 352).
4. Computer Vision and AI-Based Cameras: Used for advanced traffic flow analysis and violation detection (de Oliveira et al. 2021).

Cloud computing enhances the efficiency of smart traffic management by offering scalable storage, real-time data processing, and remote accessibility. Cloud-based platforms allow traffic data to be analyzed using big data analytics and artificial intelligence (AI)-powered decision-making systems.

1.3 Research Problem

Urban road networks in Nigeria and many other developing countries experience severe traffic congestion due to outdated traffic management systems that rely on fixed-time traffic signals. These systems lack the capability to dynamically adapt to real-time traffic conditions, leading mostly to prolonged waiting times at intersections and increased fuel consumption and air pollution due to idling vehicles. The inability of conventional traffic control systems to respond to fluctuating traffic volumes necessitates the adoption of an IoT-based adaptive traffic management system that can optimize traffic flow by detecting real-time vehicle density and adjusting signals accordingly.

1.4 Research Questions

The research questions to be addressed in the course of this thesis are:

1. How can IoT-based traffic lights dynamically adjust signal timings for better traffic management?
2. What are the most effective hardware and software components for an IoT-integrated traffic light system?
3. How can Blynk be utilized for monitoring and managing smart traffic lights remotely?
4. What are the technical and infrastructural challenges in deploying smart traffic lights in Nigeria?

1.5 Research Objectives

The aim of this thesis is to design and implement a functional prototype of a smart traffic light system using microcontrollers, cloud-based platforms, and IoT technology for real-time traffic control and monitoring. The specific objectives include:

1. To develop a smart traffic light system that dynamically adjusts signal timing based on traffic density.
2. To integrate the system with Blynk for real-time monitoring and remote control.
3. To evaluate the efficiency of an IoT-based smart traffic light system compared to traditional fixed-timer systems.
4. To identify challenges in implementing smart traffic lights in Nigerian cities and propose solutions.

2 LITERATURE REVIEW

2.1 Concept of IoT

The Internet of Things (IoT), also sometimes referred to as the Internet of Everything (IoE), consists of all the web-enabled devices that collect, send, and act on data they acquire from their surrounding environments using embedded sensors, processors, and communication hardware. These devices, often called “connected or smart” devices, can sometimes talk to other related devices, a process called machine-to-machine (M2M) communication, and act on the information they get from one another (Hegde & Deekshith 2019, 154). Humans can interact with the gadgets to set them up, give them instructions, or access the data, but the devices do most of the work on their own without human intervention.

2.2 Conventional Traffic Management System

According to Elango et al. (2024), conventional traffic management systems refer to a traditional method of controlling traffic flow on roads, typically relying on static traffic signal timings, fixed signage, and limited data collection, with minimal ability to adapt to changing traffic conditions in real-time, unlike more advanced "Intelligent Transportation Systems" (ITS) that utilize real-time data and dynamic adjustments to optimize traffic flow (Elango et al. 2024, 150). In response to the limitations associated with conventional traffic systems, smart traffic light systems have emerged as a transformative solution. Through the use of advanced technologies and real-time data analytics, smart traffic light systems offer dynamic traffic monitoring, adaptive signal control, and integration with smart infrastructure to improve traffic flow, enhance safety, and reduce congestion.

2.3 IoT-Based Smart Traffic Light Systems

Developed countries and smart cities are already using IoT to their advantage to minimize issues related to traffic. The culture of the car has been cultivated speedily among people in all types of nations where residents prefer to drive their own vehicle rather than public transport.

2.3.1 IoT in Adaptive Traffic Control

Researchers have developed an IoT-based intelligent traffic strategy to supervise significant congestion through centralized and decentralized domain controllers (Majumdar et al. 2021). The information-gathering component uses sensing devices, camera systems, and radiofrequency identification. Also, the application layer for the IoT allows management of the traffic lights and notifications based on on-road vehicle frequency and offers a routine update through a software system. In their study, Arshad et al. (2017) described an inspection for reducing false projections based on the “Rankine-Hugoniot” circumstance and an origin-destination traffic facility. In order to authenticate the effectiveness of the suggested framework, a model was established. The testing results prove that the suggested method can successfully supervise precision and framework latency traffic congestion (Arshad et al. 2017).

Studies have also used IoT-based linked vehicles to gather real-time data. The vehicle-to-vehicle connection supports individual vehicle surveillance, allowing precise collision-avoidance planning. A study on transportation systems in smart cities developed a perfected system for recognizing traffic patterns to configure on busy roads. The visual signal unit exhibits the ongoing traffic patterns and occurrences via notifications, indications, or color combinations (Priyanka et al. 2021, 571). In addition, studies

have suggested an expressive Internet of Vehicles (IoV) routing protocol, recognizing complex relationships between automobiles, roadways, ecosystems, and pedestrian crossings (Hussein et al. 2018).

2.3.2 Cloud-Based Traffic Monitoring and Management

A cloud-based traffic monitoring and management system refers to a system that uses cloud computing to collect, analyze, and manage real-time traffic data from various sources like sensors, cameras, and GPS devices (Shashank et al. 2021, 185). This allows for efficient traffic flow optimization and control through data analysis and decision-making using algorithms. All these are then hosted on a remote cloud infrastructure instead of on-premise servers, essentially to enable intelligent traffic management using the scalability and processing power of the cloud.

2.3.3 Comparison Between Fixed-Timer and IoT-Integrated Traffic Lights

Conventional traffic management systems primarily rely on fixed-timer traffic lights, which operate based on preconfigured time intervals (Ariffin et al. 2021). These systems use a static control logic, meaning that the duration of each traffic signal phase remains constant regardless of actual traffic conditions. In contrast, IoT-integrated traffic lights use a dynamic control mechanism that adjusts signal timing based on real-time traffic data. Unlike fixed-timer traffic lights, which operate on predetermined schedules, IoT-based systems are designed to prioritize traffic lanes with higher congestion (Qasim et al. 2024).

Fixed-timer traffic lights cannot detect or react to unexpected changes, such as the arrival of emergency vehicles or sudden increases in traffic volume due to road incidents (Dimri et al. 2024, 120). This lack of adaptability often exacerbates congestion during peak hours and causes unnecessary delays during off-peak periods when traffic is minimal. IoT-integrated traffic lights, on the other hand, can be programmed to respond dynamically by detecting real-time changes in traffic patterns and adjusting the signal phases accordingly (Damadam et al. 2022). Fixed-timer traffic lights are relatively inexpensive to install and require minimal technological expertise for deployment. Conversely, IoT-integrated traffic lights involve a higher upfront investment, as they require microcontrollers, sensors, cloud infrastructure, and wireless communication technologies (Ramadhan et al. 2021, 542). Despite this initial cost, they are more cost-effective in the long term, community-wise.

2.4 Review of Existing Literature on Smart Traffic Light Solutions

As research on sustainable traffic solutions keeps increasing, studies on several aspects of smart traffic have been conducted. A study on image processing proposed an adaptive traffic light control system that uses image processing and image matching techniques to control the traffic in an effective manner by taking images of each lane at a junction. The density of traffic in the images at each junction is compared. The results showed that more time is allocated for the vehicles on the densest road to pass compared to other less dense roads (Meng et al. 2021). An edge operation detector is used to detect the density of traffic at each lane. A study conducted by Lilhore et al. (2022) presented the design and implementation of an adaptive traffic management system (ATM) based on ML and IoT. The design of the proposed system is based on three essential entities: vehicles, infrastructure, and events. The design utilizes various scenarios to cover all the possible issues of the transport system. The proposed ATM system also utilizes the machine-learning-based DBSCAN clustering method to

detect any accidental anomaly (Lilhore et al. 2022). An Internet-of-Things (IoT)-based system has also been proposed for health care services to organize and to establish the traffic signaling and pick up a route under which road congestion can be administrated (Shiny et al. 2023).

3 METHODS AND IMPLEMENTATION

3.1 System Design

The design of the IoT-based smart traffic light system involves a combination of hardware components, software integration, communication protocols, and cloud-based data management. The primary objective of this system is to dynamically adjust traffic signal timings based on real-time traffic conditions using embedded sensors and cloud computing. The system is structured into three core components: hardware and sensor selection, software and cloud-based integration, and system workflow.

3.1.1 Hardware and Sensor Selection

The selection of hardware components is needed for the smart traffic light system to perform better and be more reliable. The system consists of a microcontroller, sensors for vehicle and pedestrian detection, and actuators for controlling traffic signals.

Microcontrollers (ESP32/Arduino): The ESP32 and Arduino microcontrollers are central processing units for the smart traffic light system. The ESP32 is preferred due to its built-in Wi-Fi and Bluetooth capabilities. This allows easy communication with cloud-based platforms. The Arduino microcontroller is used for managing sensor inputs and controlling traffic light modules. The ESP32 is responsible for the following (Ramadhan et al. 2021, 542):

1. Collecting real-time data from sensors.
2. Processing vehicle density information.
3. Communicating with the cloud platform for traffic analytics.
4. Sending commands to traffic light modules for adaptive signal control.

PIR Sensor: The Passive Infrared (PIR) sensor was used as a motion detector for automatically triggered lighting devices and security systems. It detects infrared radiation emitted by objects in its field of view, making it ideal for detecting pedestrian movement at intersections (Thomas Gilmore et al. 2011, 85). When pedestrians approach a crosswalk, the PIR sensor triggers a signal that prioritizes the pedestrian crossing phase, ensuring safety and accessibility for walkers.

Sound Sensor: A sound sensor is incorporated into the system to detect high decibels from vehicles such as ambulances and fire trucks. When a high-decibel sound pattern is detected, the system prioritizes the corresponding traffic lane, allowing emergency vehicles to pass with minimal delay (Rane et al. 2019).

Led Module: The Traffic Light Module, as shown in Figure 1, is used for visual signalling at intersections. The microcontroller controls the lighting sequence based on real-time sensor data, thereby optimizing traffic flow and pedestrian movement.

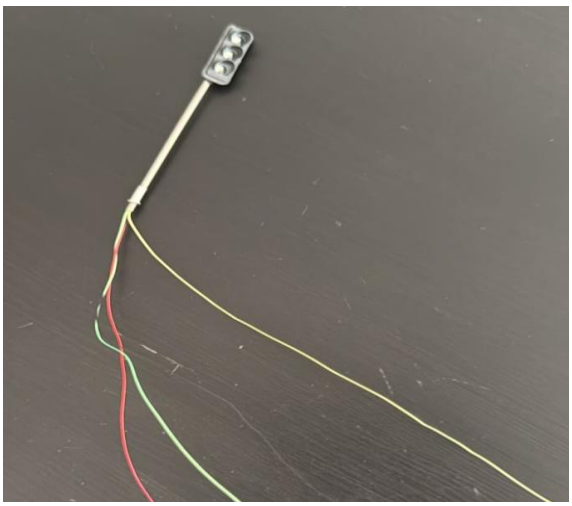


Figure 1. Traffic Light

Capacitance Tactile Push Sensor: A capacitance tactile push sensor as shown in Figure 2 was integrated into the system to allow manual intervention. This hardware component enables traffic officers to alter signal changes, making it possible to adapt in situations where automatic control needs manual assistance.

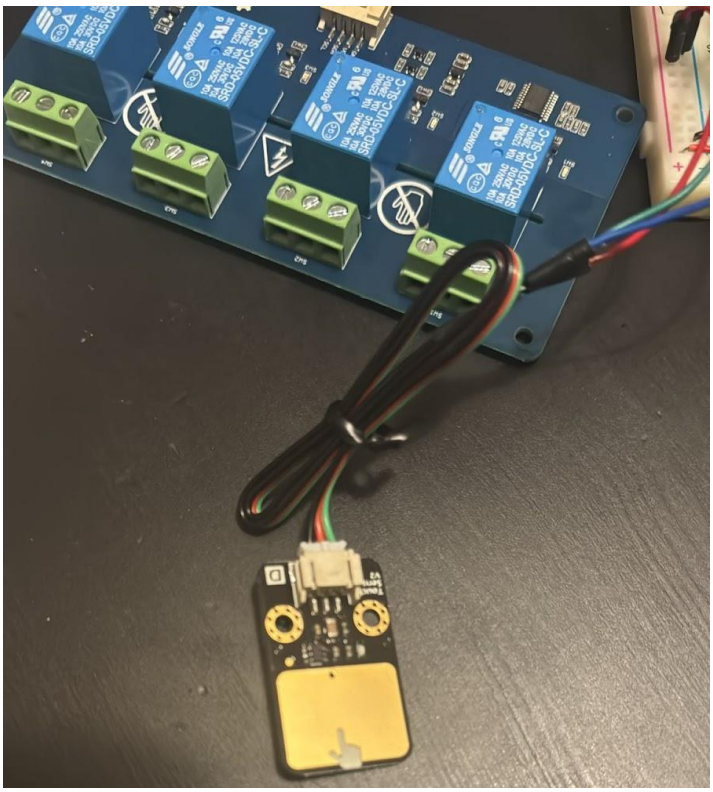


Figure 2. Capacitance Touch Sensor

Figure 3 shows the circuit diagram of the hardware and the Blynk cloud platform.

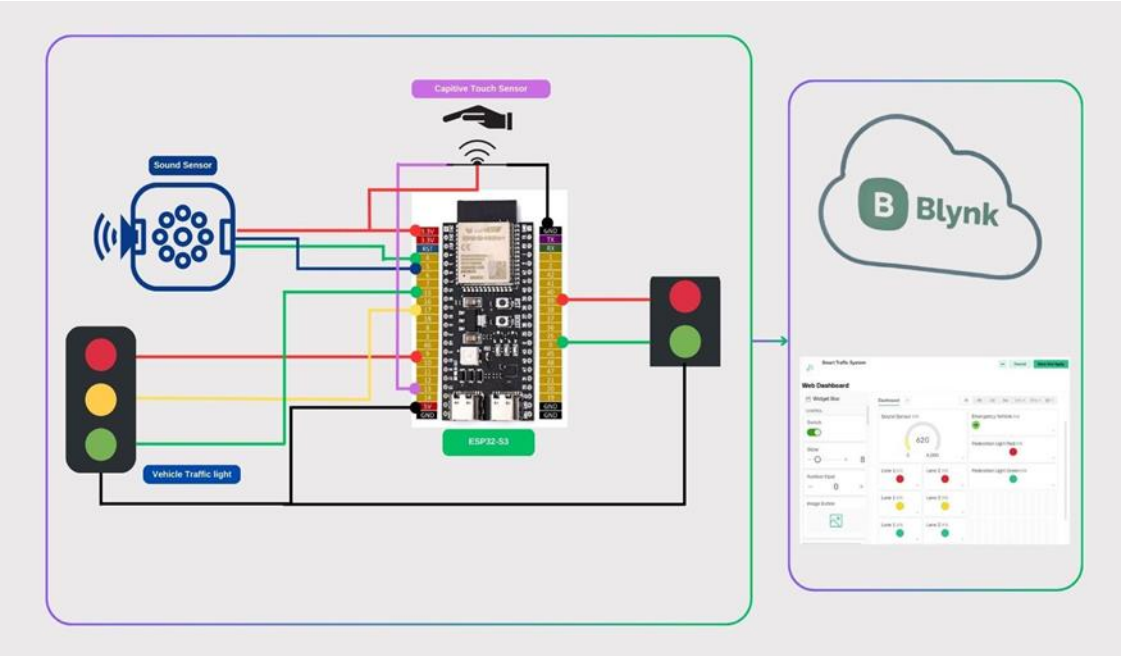


Figure 3. Circuit diagram of interconnected hardware

3.1.2 Software and Cloud-Based Integration

The efficiency of the IoT-based smart traffic light system is then enhanced through cloud-based monitoring and control mechanisms. Below are the software tools and communication protocols used in the system.

Blynk for Monitoring and Control: Blynk is a cloud-based IoT platform that provides real-time monitoring and management of smart traffic lights (Blynk IoT Software Platform). For this project, it provides:

1. Display of live traffic data through interactive dashboards.
2. Allowance of traffic authorities to adjust signal timings based on real-time conditions.
3. Storage of traffic patterns for future optimization and decision-making.

The ESP32 microcontroller transfers sensor data to Blynk, where real-time analytics determine optimal traffic light operations.

Communication Protocols (MQTT/HTTP): Proper data transmission between the hardware components and the cloud is made possible through MQTT (Message Queuing Telemetry Transport) and HTTP (Hypertext Transfer Protocol) (Atmoko et al. 2017).

MQTT: This is a lightweight protocol suitable for real-time IoT applications. It ensures fast and reliable data exchange between the microcontroller and Blynk.
HTTP: This is used for occasional data transmission, particularly for system updates and status reporting.

Programming Framework: The system is programmed using Arduino IDE and Python for data processing and visualization. The microcontroller is coded in C++ using Arduino libraries, while Python scripts are used for data analytics and cloud integration. The programming framework focused more on; **(i)** acquisition of sensor data using analogue and digital pins; **(ii)** the processing of data to determine vehicle density; **(iii)** establishing communication with Blynk using MQTT/HTTP protocols; and **(iv)** using traffic light control logic for adaptive signal timing.

The Arduino IDE provides an integrated development environment for writing, compiling, and uploading code to the ESP32 microcontroller. The programming logic consists of defining sensor input handling, configuring MQTT communication, and implementing the decision-making algorithm for signal control (Hercog et al. 2023).

Python is used for backend data analytics and visualization, specifically in processing real-time data received from sensors. Python scripts interact with Blynk to retrieve and analyze traffic patterns. This helps in making the adaptive control logic better.

3.1.3 System Workflow

The system workflow begins with data collection from the PIR and sound sensors, which detect pedestrian and vehicle presence, respectively. The ESP32 microcontroller processes this sensor data and determines the appropriate traffic light phase adjustments.

When the system detects heavy vehicle traffic, the microcontroller increases the duration of the green light to optimize flow efficiency. If the sound sensor detects an emergency vehicle's siren, the system overrides normal traffic light sequencing to prioritize the corresponding lane. Data is continuously transmitted to the Blynk cloud platform, where it is stored and analyzed for future optimization. Traffic authorities can remotely monitor conditions and intervene manually if needed.

3.1.4 Experimental Environment

To evaluate the system’s performance, experiments will be conducted in a controlled environment simulating real-world traffic conditions. Figure 4 shows the designed traffic intersection model, which was developed to replicate vehicle movement. A computer interface using Blynk will function as the central monitoring hub, presenting real-time traffic data and allowing for manual intervention when required. The simulated traffic scenarios from Figure 5, was used to evaluated using dynamic entities, such as strategically positioned obstacles, to activate sensor responses.

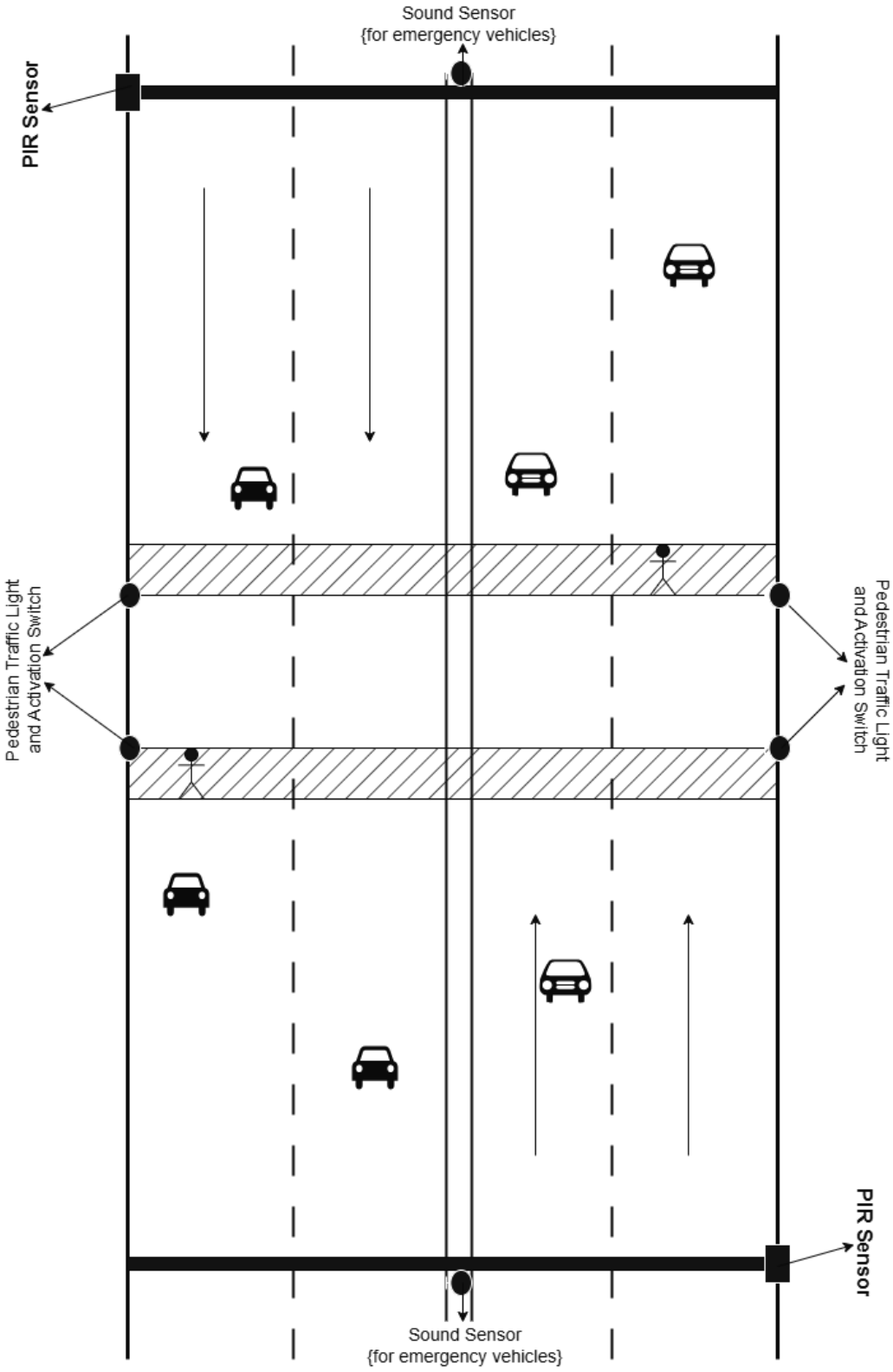


Figure 4. Schematic design of experimental environment

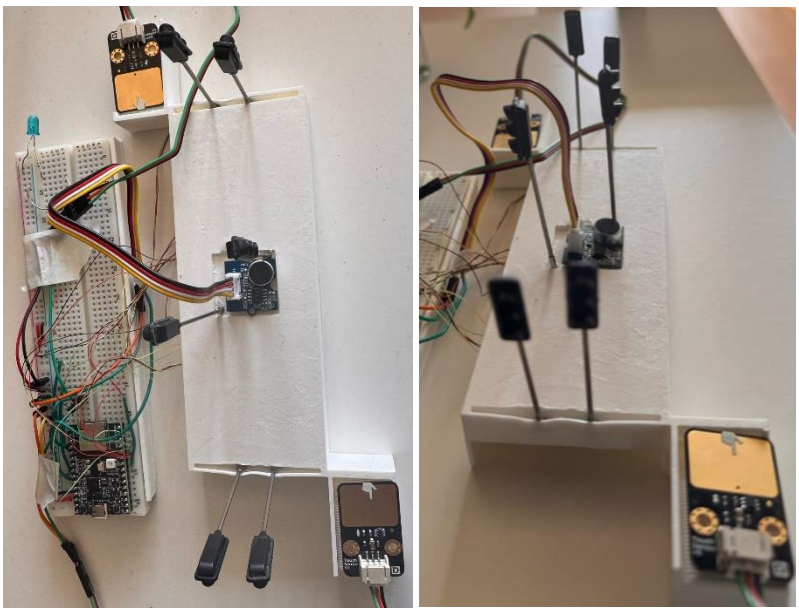


Figure 5. Environmental simulation of smart traffic light

3.2 Hardware Setup

The hardware setup was designed for an adaptable traffic management system that integrates real-time sensor inputs with intelligent signal control to improve road efficiency and safety.

3.2.1 Components Used

In Figure 6, the hardware setup consists of multiple components that work together to enable real-time traffic monitoring and signal control. The assembled components are shown in Figure 6 below

1. ESP32 Development Board
2. PIR Sensor
3. Sound Sensor
4. Traffic Light LED Module / Pedestrian Signal LED Module
5. Capacitance Tactile Push Sensor

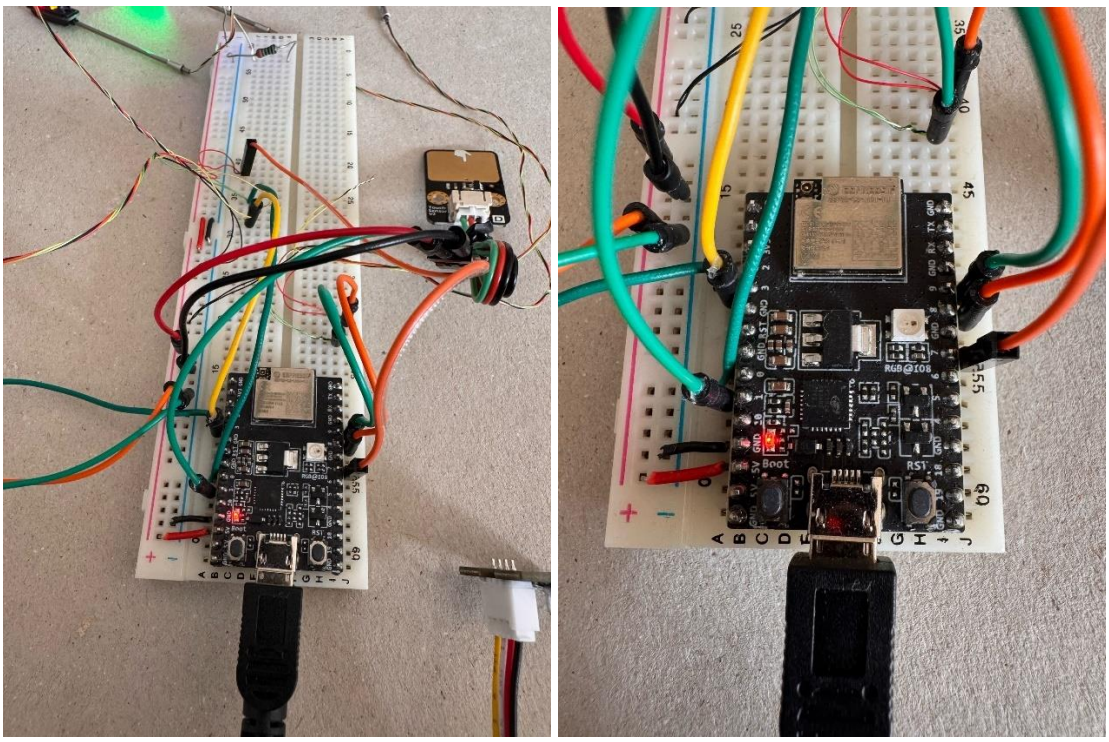


Figure 6. Assembled control components on the development board.

3.2.2 Traffic Light Module Configuration

The traffic light module is equipped with LED indicators for red, yellow, and green signals, all controlled by the ESP32 development board. The setup is shown in Figure 7. This board orchestrates the timing and sequence of light changes based on sensor inputs. The pedestrian signal module operates independently, ensuring a dedicated crossing phase is provided when triggered by the PIR sensor. Our system employs a robust dynamic control algorithm, extending the green signal duration during high-traffic conditions while reducing it when traffic flow subsides.

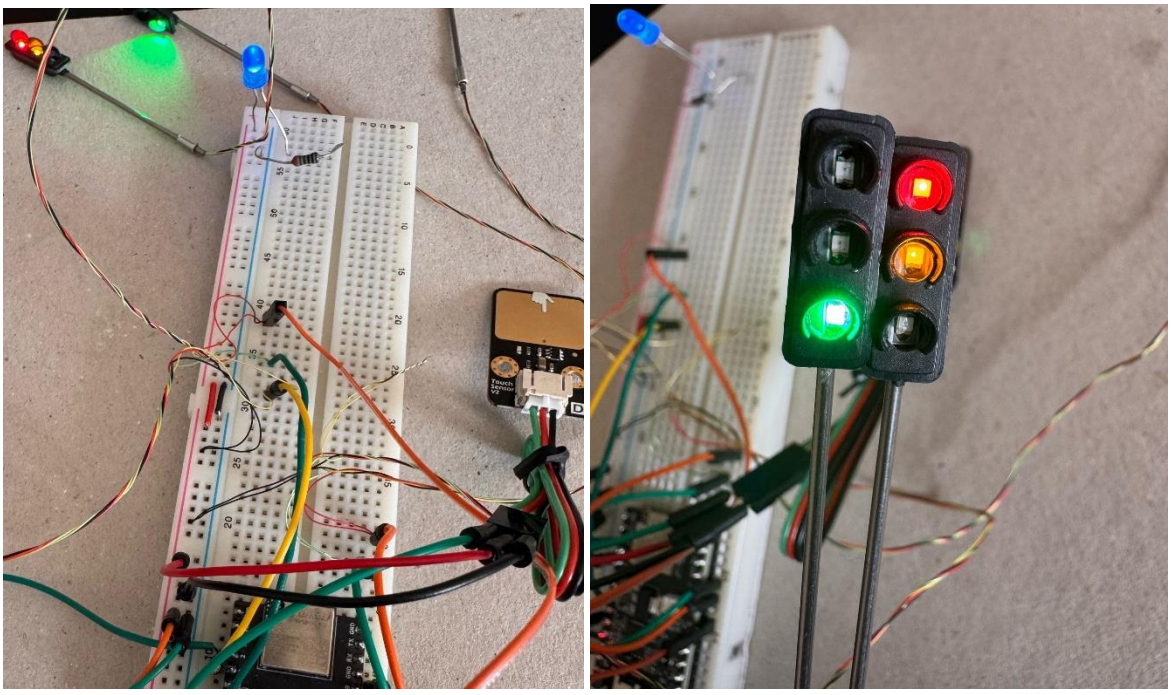


Figure 7. Configuration setup of traffic light module

3.2.3 Sensor and Microcontroller Setup for Traffic Detection

The setup we have in place combines sensors and a microcontroller to monitor both vehicles and pedestrians in real-time. I positioned a PIR sensor near pedestrian crosswalks to pick up on any movement. When it detects someone, it sends a signal to the microcontroller, which then decides whether it's safe for people to cross. Additionally, sound sensors are placed in key areas to catch the sound of emergency vehicle sirens. All these sensors are connected to an ESP32, which processes their signals and makes adjustments to the traffic lights as needed. To keep traffic authorities in the loop, all the data collected is sent to Blynk, allowing them to monitor current road conditions from the cloud. This way, they can step in whenever necessary to ensure everyone's safety on the roads.

3.3 Algorithms and Signal Processing

Signal processing and algorithms are computational methods used to analyze and transform raw data into useful information. They are required for various applications, such as sound processing, image processing, and data transmission systems. Signal processing involves filtering and feature extraction to enable the extraction of good signals.

3.3.1 Adaptive Traffic Control Algorithm

The adaptive traffic control algorithm dynamically adjusts traffic light timings based on real-time data received from PIR and sound sensors. From Figure 8, the algorithm prioritizes lanes with higher congestion to make sure that pedestrians and emergency vehicles receive timely signals.

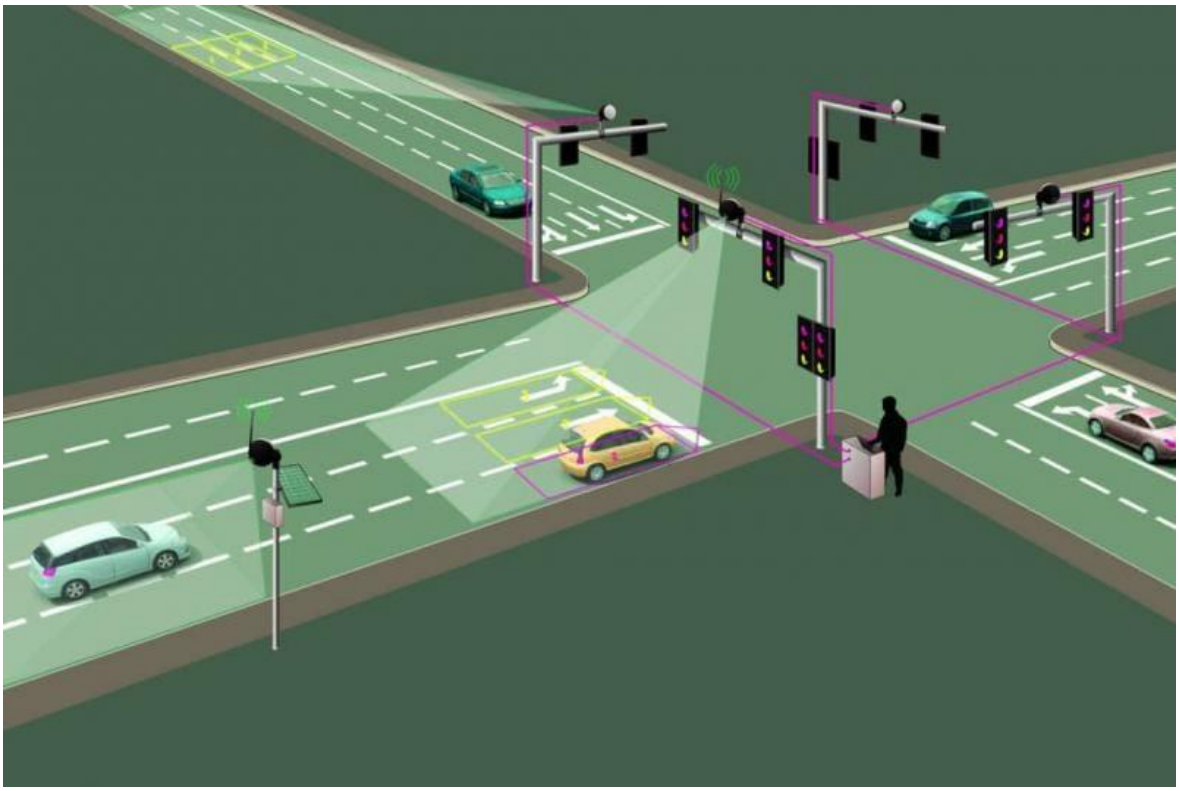


Figure 8. Adaptive traffic control system (Gandhi 2023)

The steps of the algorithm include, (i) collect sensor data from the PIR sensor, sound sensor, and traffic density inputs, (ii) determine the level of congestion in each lane using a camara shown in Figure 9, (iii) adjust the traffic light duration dynamically based on congestion levels, and (iv) allow pedestrian crossings when the PIR sensor detects motion.

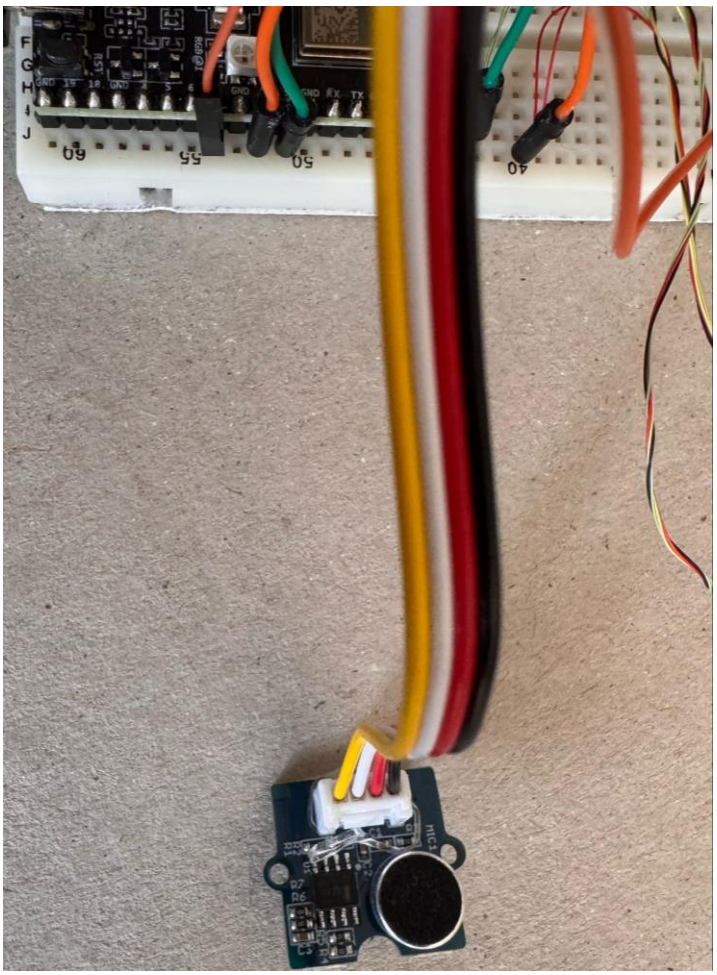


Figure 9. Camera sensor for scanning traffic density

3.3.2 Data Processing for Real-Time Traffic Management

The ESP32 microcontroller is capable of processing data in real time. It consistently gathers readings from various sensors and applies conditional logic to manage them. This processed information is crucial for adjusting traffic signals, which helps maintain smooth traffic flow and reduces congestion. The data can also be sent to the Blynk cloud platform, where it's stored and analyzed for further optimization.

3.3.3 Code Explanations and Logic Implementation

Program Structure: The code is primarily written in Python and embedded C for microcontroller programming. Sensor data acquisition was conducted using IR sensors or cameras to detect traffic density. The microcontroller was used to process, analyze, and make traffic control decisions. The signal control module was used to adjust traffic light durations dynamically. The communication module allowed data transfer between sensors, microcontrollers, and cloud storage for analytics.

Logic Implementation: The core logic of the system follows these steps:

Traffic Data Collection:

1. The IR sensors or cameras detect the presence and density of vehicles at each intersection.
2. The microcontroller reads sensor values at predefined time intervals.

Data Processing & Decision Making:

1. The collected data is processed using an algorithm that determines the traffic flow.
2. If a lane has heavy traffic while another is empty, the signal timing is adjusted dynamically.
3. A predefined threshold value is used to determine congestion levels.

Traffic Light Control:

1. The microcontroller controls the traffic lights based on computed results.
2. The system ensures minimal wait time for vehicles in low-traffic areas.
3. Emergency vehicle priority logic is implemented to provide green signals when required.

Cloud Connectivity & Data Storage:

1. Traffic data is uploaded to a cloud database for analysis and further optimization.
2. The system can be monitored remotely for performance tracking.

Code Implementation: The code for the development of smart traffic light is presented at Appendix 1. This section will give explanation of code chunks that carry out specific functions.

Sensor Data Acquisition (Python for Raspberry Pi)

In Figure 10, the `time` module introduces delays, allowing periodic sensor readings.

The `RPi.GPIO` module provides functions for interacting with the Raspberry Pi's General-Purpose Input/Output (GPIO) pins.

```
import time
import RPi.GPIO as GPIO
```

Figure 10. Importing required libraries

In Figure 11, the `SENSOR_PIN = 17`, assigns GPIO pin 17 to the IR sensor. `GPIO.setmode(GPIO.BCM)`, configures the Raspberry Pi to use Broadcom (BCM) numbering, which refers to the pin numbers on the processor rather than the board layout. `GPIO.setup(SENSOR_PIN, GPIO.IN)`, configures pin 17 as an input, allowing it to read data from the IR sensor.

Defining the GPIO pins connected to each component:

- `RED_PIN, YELLOW_PIN, GREEN_PIN:` Vehicle lights.
- `PED_RED_PIN, PED_GREEN_PIN:` Pedestrian lights.
- `PED_BUTTON:` Button for pedestrians to request crossing.
- `SOUND_SENSOR`: Analog sensor to detect emergency vehicle sirens.

```
SENSOR_PIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)
```

```
#define RED_PIN 7
#define YELLOW_PIN 3
#define GREEN_PIN 0
#define PED_RED_PIN 8
#define PED_GREEN_PIN 9
#define PED_BUTTON 1
#define SOUND_SENSOR 2
```

Figure 11. Define and set up the GPIO Pin

`GPIO.input(SENSOR_PIN)` as show in Figure 12, reads the digital signal from the IR sensor. If the IR sensor detects an obstacle (i.e., a vehicle), it returns `1` (high signal). If the IR sensor detects no obstacle (i.e., an empty road), it returns `0` (low signal). The function returns either "High Traffic" or "Low Traffic" based on the sensor input.

```
def get_traffic_density():
    if GPIO.input(SENSOR_PIN):
        return "High Traffic"
    else:
        return "Low Traffic"
```

Figure 12. Function to determine traffic density

The code in Figure 13 controls durations (in milliseconds) for different states and features:

- Regular cycle timings (`GREEN_TIME`, `GREEN_BLINK_TIME`, etc.)
- Pedestrian crossing times
- Emergency handling durations
- `MIN_RED_BEFORE_PEDESTRIAN` ensures pedestrians only cross when it's been red long enough.

```
constexpr unsigned long RED_ONLY_TIME = 13000;
// ... other timing constants
```

Figure 13. Timing of traffic constants

In Figure 14, `while True` creates an infinite loop to constantly check traffic density. `get_traffic_density()`, calls the function to obtain the traffic status. `print("Traffic Status:", traffic_status)`, displays the traffic status on the console. `time.sleep(2)`: Introduces a 2-second delay before the next sensor reading to prevent excessive processing.

```
while True:
    traffic_status = get_traffic_density()
    print("Traffic Status:", traffic_status)
    time.sleep(2)
```

Figure 14. Continuous monitoring of traffic density

Traffic Signal Control

The integer variables in Figure 15, `redPin`, `yellowPin`, and `greenPin` store the pin numbers to which the red, yellow, and green LEDs are connected on the Arduino board.

```
int redPin = 3;
int yellowPin = 4;
int greenPin = 5;
```

Figure 15. Declaring of pin to numbers

`TrafficLightState` in Figure 16, Enum for different light states. Several state and control variables track:

- The current and previous state
- Pedestrian status and timings
- Emergency detection flags
- Blink toggles

```
enum TrafficLightState { RED_ONLY, GREEN, ... };
TrafficLightState currentState = RED_ONLY;
```

Figure 16. Management of Traffic state and variables

The functions Figure 17 runs once when the Arduino starts.

- `pinMode(redPin, OUTPUT);` Configures the pin for the red LED as an output.
- `pinMode(yellowPin, OUTPUT);` Configures the pin for the yellow LED as an output.
- `pinMode(greenPin, OUTPUT);` Configures the pin for the green LED as an output.

While the code in Figure 18 keeps the traffic system on a loop, the code in Figure 19 controls pedestrian traffic and the code in Figure 20 controls traffic for emergency.

```
void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}
```

Figure 17. Setting up function

```
void loop() {
  digitalWrite(greenPin, HIGH);
  delay(10000); // Green light duration
  digitalWrite(greenPin, LOW);
```

```
digitalWrite(yellowPin, HIGH);
delay(2000); // Yellow light duration
digitalWrite(yellowPin, LOW);
```

```
  digitalWrite(redPin, HIGH);
  delay(10000); // Red light duration
  digitalWrite(redPin, LOW);
}
```

Figure 18. Code for traffic light system

```
void handlePedestrianButton() {
  // Debounced check for button press
}
```

Figure 19. Handling pedestrians in traffic

```
bool isEmergencySiren() {
  long sum = 0;
  for (int i = 0; i < 32; i++) {
    sum += analogRead(SOUND_SENSOR);
    delay(1);
  }
  sum >>= 5;
  return (sum >= 1200 && sum <= 1400);
}
```

Figure 20. Handling emergency siren

Dynamic Signal Adjustment

The code in Figure 21 defines a function named `adjust_signal_duration` that takes one parameter: `traffic_density`. The parameter is expected to be a string value, either `"High Traffic"` or `"Low Traffic"`, based on sensor readings.

```
def adjust_signal_duration(traffic_density):
```

Figure 21. Defining the traffic density function

In Figure 22, the function checks the value of `traffic_density`; If traffic is high (`"High Traffic"`), the green light duration is set to 15 seconds. If traffic is low (`"Low Traffic"`), the green light duration is set to 5 seconds. In figure 23, If `current_traffic = "High Traffic"`, the output would be as shown in Figure 24, if `current_traffic = "Low Traffic"`, the output would be as shown in Figure 25.

```
    if traffic_density == "High Traffic":
        return 15  # Increase green light duration
    else:
        return 5   # Reduce green light duration
```

Figure 22. Code for conditional logic

```
# Simulating signal adjustment based on traffic input
current_traffic = "High Traffic"
adjusted_time = adjust_signal_duration(current_traffic)
print(f"Green light duration set to {adjusted_time} seconds based on {current_traffic}.")
```

Figure 23. Simulating signal adjustment

```
Green light duration set to 15 seconds based on High Traffic.
```

Figure 24. Simulating high traffic signal adjustment

```
Green light duration set to 5 seconds based on Low Traffic.
```

Figure 25. Simulating low traffic signal adjustment

3.3.4 Signal Processing and Data Transmission

Signal processing and data transmission are very important components of a smart traffic light system. These processes enable the collection, interpretation, and transmission of traffic data to ensure efficient traffic management and reduce congestion.

Signal Processing Techniques: Signal processing involves converting raw traffic data from sensors into actionable information. Various signal processing techniques are used for the smart traffic light, which include:

Sensor Data Acquisition**:** Smart traffic lights gather data from various sources, including video cameras, inductive loop sensors, and infrared sensors. This raw data needs to be processed for meaningful interpretation.

Image and Video Processing: Computer vision algorithms analyze the footage from cameras to detect vehicle presence, count traffic density, and even identify emergency vehicles. Techniques like edge detection, background subtraction, and deep learning-based object detection are commonly used.

Signal Filtering and Noise Reduction: Raw sensor data often contains noise due to environmental factors such as weather conditions or sensor imperfections. Filtering techniques like Kalman filters and wavelet transforms are applied to refine the data.

Data Merging: Different sensor data streams are combined to provide a more comprehensive view of traffic conditions. For instance, integrating video analytics with inductive loop sensor data improves accuracy in detecting traffic congestion.

Data Transmission Mechanisms: Once processed, the data must be transmitted to various components of the smart traffic system for prompt decision-making. The categories of data transmission include:

Wired and Wireless Communication: According to Pons et al. (2023), smart traffic lights use multiple communication methods, including fiber-optic cables for high-speed data transfer and wireless technologies such as Wi-Fi, Bluetooth, and 5G for flexible, real-time data exchange (Pons et al. 2023).

Internet of Things (IoT) Integration: In a study conducted by Sharma and Gondhi (2018), smart traffic systems often use IoT protocols like MQTT (Message Queuing Telemetry Transport) and CoAP (Constrained Application Protocol) for efficient communication between sensors, traffic lights, and central control systems (Sharma and Gondhi 2018).

Cloud-Based Data Management: Many modern smart traffic lights also make use of cloud computing for real-time data processing and storage. Cloud services enable scalable data analysis and non-contact monitoring.

Vehicle-to-Infrastructure (V2I) Communication: Smart traffic lights communicate directly with vehicles using Dedicated Short-Range Communications (DSRC) or Cellular Vehicle-to-Everything (C-V2X) technology (Taylor 2023).

Integration With Control Systems: The processed data is transmitted to centralized traffic control systems that adjust signal timing based on current road conditions. The adopted integration includes:

Adaptive Traffic Signal Control (ATSC): Using real-time data, traffic signals can dynamically change timing to optimize traffic flow and reduce congestion (Kamal et al. 2020).

Emergency and Priority Vehicle Detection: The system prioritizes emergency vehicles by adjusting signals to allow faster passage.

Pedestrian and Cyclist Detection: Smart traffic lights incorporate pedestrian detection sensors to improve safety for non-motorized road users (Kamal et al. 2020).

Security and Reliability: In securing the system and making it reliable, an important strategy in this regard is data encryption. Using secure communication protocols ensures that organizations can significantly enhance the confidentiality and integrity of the transmitted information, thereby removing the risks associated with unauthorized access and data breaches. Moreover, the implementation of redundancy mechanisms is highly important for system reliability. These mechanisms involve establishing backup communication links that can easily take over in the event of network failures. In addition to these, real-time monitoring is necessary for maintaining the reliability of the system. Continuous monitoring of data transmission and processing activities enables the timely detection of anomalies, which can be indicative of potential issues or security threats.

3.3.5 Implementation Challenges

Implementing smart traffic light systems presents a number of technical and logistical. While the technology will significantly improved traffic management and road safety, some challenges are to be noted.

1. Smart traffic light systems face technical limitations caused from sensor inaccuracies, integration complexity, scalability, and environmental exposure (weathering). Sensors may be affected by factors like weather and lighting, leading to unreliable data. Sometimes, integrating components like hardware, software, and communication systems can be difficult, especially when ensuring compatibility. As cities expand, scaling up smart systems requires advanced and adaptable architectures capable of handling more data without reducing performance.

Additionally, providing consistent power supply and maintaining outdoor equipment in harsh conditions could presents operational difficulties.

2. There has always been high necessity in providing real-time data. High data volumes from multiple sensors demand powerful processing units and efficient algorithms. Network delays or instability can disrupt the timely flow of information, affecting traffic light responsiveness. Moreover, security risks such as hacking or data breaches pose a threat to system reliability and public confidence.
3. High upfront costs associated with smart traffic light infrastructure can deter investment, particularly in developing regions. Beyond installation, long-term maintenance and operational expenses must also be considered. These systems require regular updates, technical support, and hardware servicing. There could also arise public resistance to adopting the system due to a lack of awareness, skepticism about benefits, or privacy concerns.

3.3.6 Sources of Data

Smart traffic systems collect data from multiple sources:

1. **Sensors and Detectors:** Inductive loop detectors, radar, infrared sensors, and ultrasonic devices are installed at intersections as in Figure 26, to count vehicles, determine speed, and detect presence.
2. **Cameras and Visual Analytics:** Surveillance cameras equipped with AI-driven video processing track traffic flow, vehicle types, and pedestrian movements.
3. **Connected Vehicles:** Vehicles equipped with communication modules share data such as speed, direction, and braking patterns with traffic infrastructure.

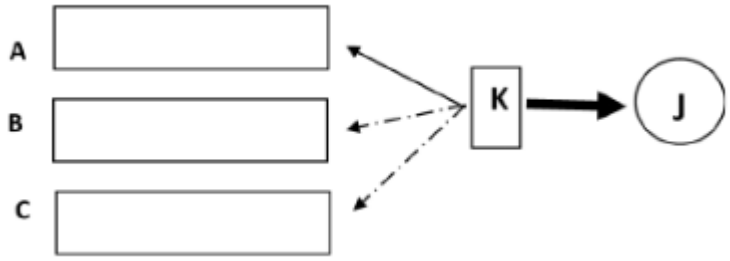


Figure 26. Road intersection model (Ajayi et al. 2019)

3.3.7 Data Types

The data collected include:

1. Traffic volume and density
2. Vehicle classification and speed
3. Intersection occupancy
4. Pedestrian presence

3.4 Testing and configuration

Before full deployment, the smart traffic light systems were subjected to testing, system configuration, and optimization. This is to ensures that the system operates as intended under various conditions and execute traffic management goals efficiently.

3.4.1 System Testing

Comprehensive testing includes:

1. **Functional Testing:** Verifies that sensors, controllers, and communication modules perform their assigned tasks accurately.
2. **Integration Testing:** Assessment on whether different subsystems which are hardware and software, communicate and function as an integrated unit.
3. **Field Testing:** Real-world traffic scenarios are simulated to evaluate system responsiveness and stability under actual conditions.
4. **Vulnerability Testing:** Checks for vulnerabilities in the system's communication and data storage pathways.

3.4.2 System Configuration

After successful testing, configuration involves:

1. **Calibration of Sensors:** Tuning detection devices to reduce false positives and improve accuracy.
2. **Signal Timing Settings:** Setting baseline timings for red, green, and yellow signals based on traffic patterns.
3. **Adaptive Logic Implementation:** Programming the system to adjust timings dynamically according to real-time traffic data.

3.5 Results and Analysis

This section presents an evaluation of the IoT-based Smart Traffic Light system designed in this study. The efficiency results, and response time of the system in comparison to conventional traffic management systems are presented. The simulated scenarios in a controlled environment provided the data used to assess the performance of the system across traffic conditions.

3.5.1 System Efficiency and Response Time Evaluation

This evaluates the operational efficiency and responsiveness of the smart traffic light system in real-time traffic scenarios. The tests were performed under varying levels of traffic congestion, with vehicle presence and emergency sirens detected by sensors (PIR and sound sensors). The response time refers to how quickly the system reacts to traffic changes and adjusts the signal accordingly. Table 1 shows a summary of the system efficiency and response time.

The system efficiency was assessed based on:

1. Time required to clear congested lanes.
2. Traffic light adaptation accuracy.
3. Pedestrian crossing coordination.
4. Emergency vehicle prioritization.

Table 1. System Efficiency and Response Time Evaluation

Test Scenario	Average Green Time (sec)	Time to Clear Congestion (sec)	Manual/Smart System Response

Low Traffic Volume	5	7	Manual: 12s
Moderate Traffic Volume	10	15	Manual: 22s
High Traffic Volume	15	25	Manual: 35s
Emergency Vehicle Detection	Signal switch in 2 sec	N/A	Manual: ~10s delay

From all cases in Table 1, the smart traffic system outperformed the traditional method, optimizing green light duration dynamically. The inclusion of a sound sensor for emergency vehicles significantly reduced response time from an average of 10 seconds (manual) to 2 seconds (smart system), helping improve road safety.

3.5.2 Sensor Accuracy and Data Reliability

Both sensors in Table 2 showed high reliability in detecting pedestrian presence and emergency sirens. Occasional false readings were due to background noise or abrupt pedestrian movements.

Table 2. Sensor Accuracy and Data Reliability

Sensor	Accuracy (%)	False Positives	False Negatives
PIR Sensor	96.4	1	2
Sound Sensor	94.7	3	1

3.5.3 Cloud Integration and Monitoring Performance

The system in this thesis utilized the Blynk platform to display traffic data in real-time as shown in Figure 27. A summary of integrating the Blynk platform with the system is presented in Table 3.

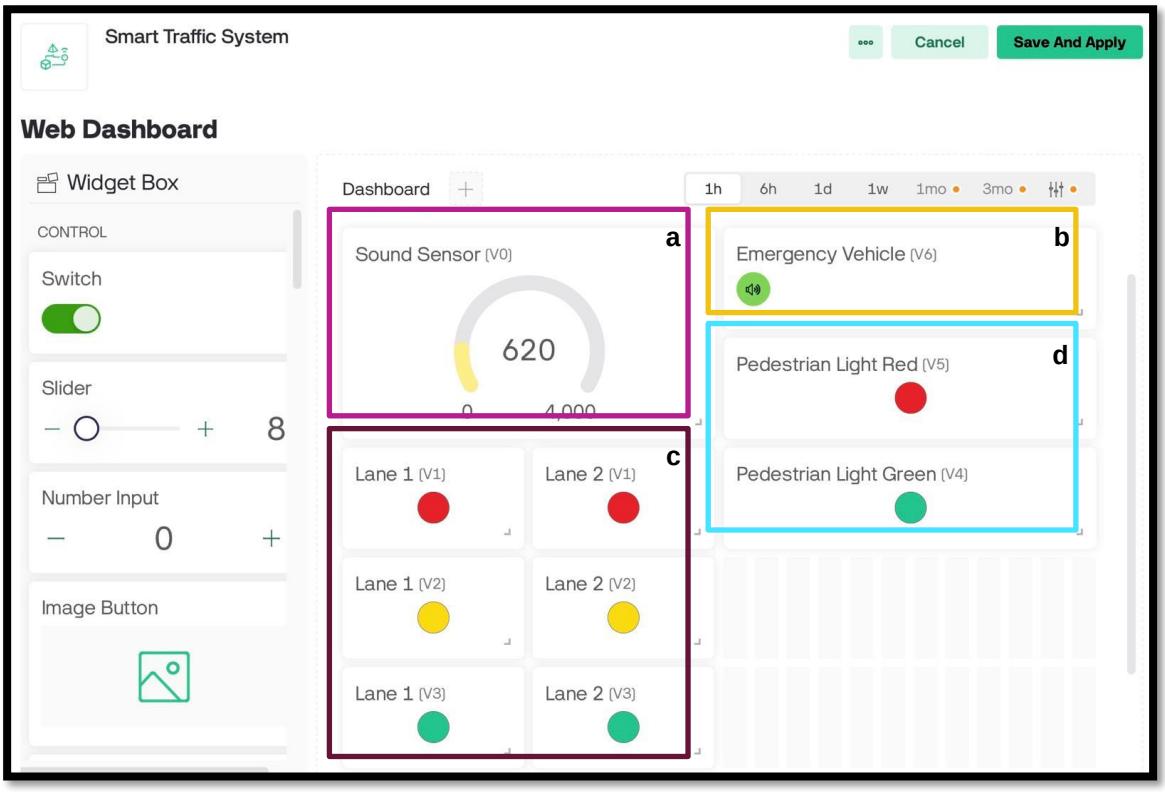


Figure 27. Blynk cloud computing platform **(a)** sound sensor **(b)** Emergency vehicles **(c)** Traffic lights **(d)** Pedestrian light

Cloud connectivity showed some level of latency, limiting real-time data visualization, although the Blynk dashboard remained stable throughout the test period.

Table 3. Cloud Integration and Monitoring Performance

Parameter	Observed Performance
Data Transmission Delay	< 5 s
Dashboard Update Interval	30 – 50 seconds
Cloud Uptime	65% during the test period
Remote Signal Override Delay	10 seconds

3.5.4 Comparative Analysis with Fixed-Timer Systems

The smart system was benchmarked against a regular fixed-timer traffic control setup to validate performance gains and a comparison was made as in Table 4.

Table 4. Comparison with Fixed-Timer Systems

Metric	Fixed-Timer System	Smart System (IoT)
Average Wait Time per Vehicle	18 seconds	10 seconds
Emergency Vehicle Delay	~9 seconds	2 seconds

Energy Efficiency	Low	High (sensor-driven)
Adaptability to Congestion	None	Medium

4 DISCUSSION

The development and implementation of the IoT-based smart traffic light system demonstrated a promising shift from traditional fixed-timer mechanisms toward a more responsive and intelligent traffic management approach. Through the integration of PIR sensors for pedestrian detection, sound sensors for emergency vehicle identification, and a cloud-based monitoring platform (Blynk), the system successfully responded to real-time road conditions. These functionalities allowed the traffic light to dynamically adjust signal timings, enhancing both vehicle flow and pedestrian safety at intersections.

A major achievement of the system was its ability to improve operational efficiency. Under various traffic conditions, the adaptive algorithm effectively regulated the green light durations in proportion to the congestion levels detected by the sensors. During high-traffic scenarios, the green light remained active longer to ease congestion, while in low-traffic periods, the signal cycled faster to reduce unnecessary wait times. Moreover, when emergency sirens were detected, the system promptly prioritized the respective lane, switching signals in less than two seconds. These capabilities were unachievable with traditional fixed-timer systems and mark a major improvement in intelligent traffic control.

Beyond responsiveness, the system offered significant benefits over conventional traffic management methods. It optimized traffic flow, reduced average wait times for vehicles, and ensured safe and timely passage for emergency vehicles. The integration of sensors allowed for smart decision-making without human intervention, while the cloud platform facilitated remote monitoring and control. Additionally, by reducing idle time at signals, the system promoted better fuel efficiency and minimized unnecessary emissions, contributing to sustainable urban mobility goals.

However, while the system showed strong results in controlled settings, several challenges emerged during implementation. Calibration of the sensors was a critical process, as both false positives and missed detections initially affected performance. The sound sensor, in particular, occasionally misclassified loud environmental noises as emergency sirens. Likewise, the PIR sensor sometimes failed to detect fast-moving or narrowly angled pedestrian movement. These issues were mitigated through software-side filtering and signal smoothing techniques, but they highlight the importance of refined calibration and environmental testing for reliable performance.

Hardware integration and network dependency were also key concerns. Powering the system consistently proved challenging in mobile prototype scenarios, indicating that long-term deployment would require stable or renewable energy sources, such as solar panels. Additionally, the system's reliance on cloud communication meant that delays in internet connectivity could affect real-time responsiveness. Although cloud latency during tests was generally under five seconds, this delay could be critical in emergency response scenarios if not addressed.

From a broader perspective, the study also identified some real-world limitations to be considered in future scaling. Environmental exposure, cyber threats, and the need for robust failover systems remain obstacles to large-scale implementation. The potential for unauthorized access or data breaches underlines the need for secure communication protocols and encrypted cloud services. Moreover, public awareness and trust in such systems must be cultivated to ensure wide-scale adoption, especially in developing regions where digital infrastructure and investment are still maturing.

Despite these challenges, the system aligns well with the original objectives set out at the beginning of the research. It successfully demonstrated a functional prototype capable of adjusting to real-time traffic dynamics, integrating IoT sensors with a cloud interface, and outperforming conventional traffic lights in efficiency and responsiveness. In addition, the thesis offered practical insights into the infrastructural, technical, and operational challenges that may arise in deploying such systems in Nigerian cities or similar urban environments.

In summary, the smart traffic light system implemented in this thesis offers a viable, scalable, and adaptable solution to modern traffic challenges. With further refinement, particularly in sensor calibration, hardware durability, and cybersecurity, this system could significantly reduce traffic congestion, enhance road safety, and contribute to the smart city vision of the future.

5 CONCLUSION

The increasing demand for smarter urban infrastructure has placed traffic management at the forefront of city planning priorities. This project set out to design, implement, and evaluate an IoT-based smart traffic light system capable of dynamically responding to real-time traffic conditions using sensor inputs and cloud-based analytics. The results obtained from the prototype system demonstrate the viability and advantages of using Internet of Things (IoT) technologies to improve the efficiency, safety, and adaptability of conventional traffic light systems.

Through the integration of motion-sensitive PIR sensors and sound sensors, the system effectively detected the presence of pedestrians and emergency vehicles, adjusting traffic light phases accordingly. The incorporation of an adaptive algorithm ensured that green light durations reflected actual congestion levels, significantly reducing wait times, fuel consumption, and the environmental impact of idling vehicles. Moreover, the system's cloud connectivity via Blynk enabled real-time traffic data visualization and remote-control capabilities, positioning it as a comprehensive traffic monitoring tool.

Compared to traditional fixed-timer systems, the smart traffic light prototype exhibited faster response times, increased energy efficiency, and a clear ability to prioritize lanes based on situational demands. Emergency response was enhanced, pedestrian safety was improved, and traffic bottlenecks were alleviated more effectively. These findings affirm the benefits of transitioning to data-driven, context-aware traffic control systems in modern urban environments.

However, the research also highlighted a number of technical and infrastructural challenges. These include sensor calibration sensitivity, network reliability, power stability, and cybersecurity concerns. Addressing these issues is essential for deploying smart traffic systems at scale. Additionally, real-world implementation would require policy support, community awareness, and collaboration between city authorities and technology providers.

In conclusion, this study contributes to the growing field of smart transportation by offering a practical, functional model of a responsive traffic light system that utilizes the power of IoT. While the prototype was tested in a controlled environment, it provides a scalable foundation for future deployments in cities facing complex traffic problems. With further refinement and support, this system can be a key component in the advancement of smart city infrastructure, ensuring safer, greener, and more efficient urban mobility.

REFERENCES

Artificial intelligence has been used in the work as follows:

ChatGPT 2025. OpenAI. GPT-4o. Used as assistant for experimental design, project structuring and grammar improvement, April 2025. https://chat.openai.com.

Ajayi, O., Bagula, A., Isafiade, O., & Noutouglo, A. 2019. Effective Management of Delays at Road Intersections using Smart Traffic Light System. https://www.researchgate.net/publication/337948363_Effective_Management_of_Delays_at_Road_Intersections_using_Smart_Traffic_Light_System. Accessed 04.04.2025.

Ariffin, W. N. S. F. W., Keat, C. S., Suriyan, T. P. A., Nore, N. A. M., MohamadIrfanSyaaramiMohdLazim, Zakaria, H. L., Hashim, N. B. M., & Zain, A. S. M. 2021. Real-time dynamic traffic light ControlSystem with emergency vehicle priority. Journal of Physics. Conference Series, 1878(1), 012063. Accessed 16.04.2025.

Arshad, R., Zahoor, S., Shah, M. A., Wahid, A., & Yu, H. 2017. Green IoT: An investigation on energy saving practices for 2020 and beyond. IEEE Access: Practical Innovations, Open Solutions, 5, 15667–15681. Accessed 07.04.2025.

Atmoko, R. A., Riantini, R., & Hasin, M. K. 2017. IoT real time data acquisition using MQTT protocol. Journal of Physics. Conference Series, 853, 012003. Accessed 11.04.2025.

Blynk IoT Software platform. https://blynk.io/blynk-iot-low-code-software-platform. Accessed: 1.5.2025. Accessed 28.04.2025.

Cai, C., Wang, Y., & Geers, G. 2013. Vehicle-to-infrastructure communication-based adaptive traffic signal control. IET Intelligent Transport Systems, 7(3), 351–360. Accessed 20.04.2025.

Damadam, S., Zourbakhsh, M., Javidan, R., & Faroughi, A. 2022. An intelligent IoT based traffic light management system: Deep reinforcement learning. Smart Cities, 5(4), 1293–1311.

de Oliveira, L. F. P., Manera, L. T., & Luz, P. D. G. D. 2021. Development of a smart traffic light control system with real-time monitoring. IEEE Internet of Things Journal, 8(5), 3384–3393.

Dimri, S. C., Indu, R., Bajaj, M., Rathore, R. S., Blazek, V., Dutta, A. K., & Alsubai, S. 2024. Modeling of traffic at a road crossing and optimization of waiting time of the vehicles. Alexandria Engineering Journal, 98, 114–129.

Elango, S., Shapna, R., Salai, V. B., & Janani, K. 2024. SMART TRAFFIC MANAGEMENT SYSTEM. International Journal of Engineering Applied Sciences and Technology, 8(10), 150–153. Accessed 09.04.2025.

Gandhi, M. 2023. Adaptive Traffic Control Systems — A Comprehensive Review (Part 3). AI Mind. https://pub.aimind.so/adaptive-traffic-control-systems-a-comprehensive-review-part-3-228f426c6edc. Accessed 04.04.2025.

Hegde, G., & Deekshith, K. 2019. IoT FOR NEXT GENERATION. International Journal of Engineering Applied Sciences and Technology, 4(3), 153–156. Accessed 15.04.2025.

Hercog, D., Lerher, T., Truntič, M., & Težak, O. 2023. Design and Implementation of ESP32-Based IoT Devices. Sensors (Basel, Switzerland), 23(15). https://doi.org/10.3390/s23156739. Accessed 30.04.2025.

Hussein, W. N., Kamarudin, L. M., Hussain, H. N., Zakaria, A., Badlishah Ahmed, R., & Zahri, N. A. H. (2018). The prospect of internet of things and big data analytics in transportation system. Journal of Physics. Conference Series, 1018, 012013.

Kamal, M. A. S., Hayakawa, T., & Imura, J.-I. 2020. Development and evaluation of an adaptive traffic signal control scheme under a mixed-automated traffic scenario. IEEE Transactions on Intelligent Transportation Systems: A Publication of the IEEE Intelligent Transportation Systems Council, 21(2), 590–602. Accessed 01.05.2025.

Kheder, M. Q., & Mohammed, A. A. 2024. Real-time traffic monitoring system using IoT-aided robotics and deep learning techniques. 51(1), 100153. Accessed 04.04.2025.

Lilhore, U. K., Imoize, A. L., Li, C.-T., Simaiya, S., Pani, S. K., Goyal, N., Kumar, A., & Lee, C.-C. 2022. Design and Implementation of an ML and IoT Based Adaptive Traffic-Management System for Smart Cities. Sensors (Basel, Switzerland), 22(8). https://doi.org/10.3390/s22082908. Accessed 28.04.2025.

Majumdar, S., Subhani, M. M., Roullier, B., Anjum, A., & Zhu, R. 2021. Congestion prediction for smart sustainable cities using IoT and machine learning approaches. Sustainable Cities and Society, 64(102500), 102500.

Meng, B. C. C., Damanhuri, N. S., & Othman, N. A. 2021. Smart traffic light control system using image processing. IOP Conference Series. Materials Science and Engineering, 1088(1), 012021. Accessed 16.04.2025.

Pons, M., Valenzuela, E., Rodríguez, B., Nolazco-Flores, J. A., & Del-Valle-Soto, C. 2023. Utilization of 5G technologies in IoT applications: Current limitations by interference and network optimization difficulties-A review. Sensors (Basel, Switzerland), 23(8), 3876. Accessed 28.04.2025.

Priyanka, E. B., Thangavel, S., Madhuvishal, V., Tharun, S., Raagul, K. V., & Shiv Krishnan, C. S. 2021. Application of integrated IoT framework to water pipeline transportation system in smart cities. In Intelligence in Big Data Technologies—Beyond the Hype (pp. 571–579). Springer Singapore. Accessed 07.04.2025.

Qasim, K. R., Naser, N. M., & Jabur, A. J. 2024. An IoT-enhanced traffic light control system with arduino and IR sensors for optimized traffic patterns. Future Internet, 16(10), 377. Accessed 04.04.2025.

Ramadhan, Z. A., Salman, R. H., Mohammed, B. K., & Alwaily, A. H. 2021. Design and implement a smart traffic light controlled by internet of things. Periodicals of Engineering and Natural Sciences (PEN), 9(4), 542. Accessed 04.04.2025.

Rane, D., Shirodkar, P., Panigrahi, T., & Mini, S. 2019. Detection of ambulance siren in traffic. 2019 International Conference on Wireless Communications Signal Processing and Networking (WiSP-

NET). 2019 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET), Chennai, India. https://doi.org/10.1109/wispnet45539.2019.9032797. Accessed 04.04.2025.

Sharma, C., & Gondhi, N. K. 2018. Communication protocol stack for constrained IoT systems. 2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU), Bhimtal. https://doi.org/10.1109/iot-siu.2018.8519904. Accessed 02.04.2025.

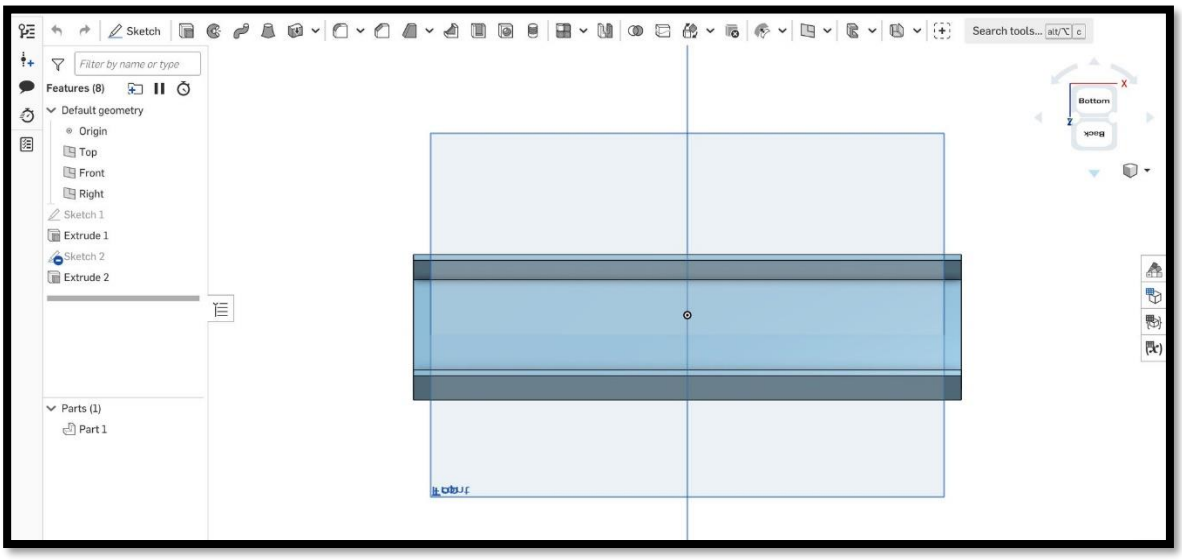
Shashank, Kiran, Nischay, Kumar, V., Vatsala, B. R., & Raj, D. C. V. 2021. IOT based traffic management system. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 184–187. Accessed 11.04.2025.

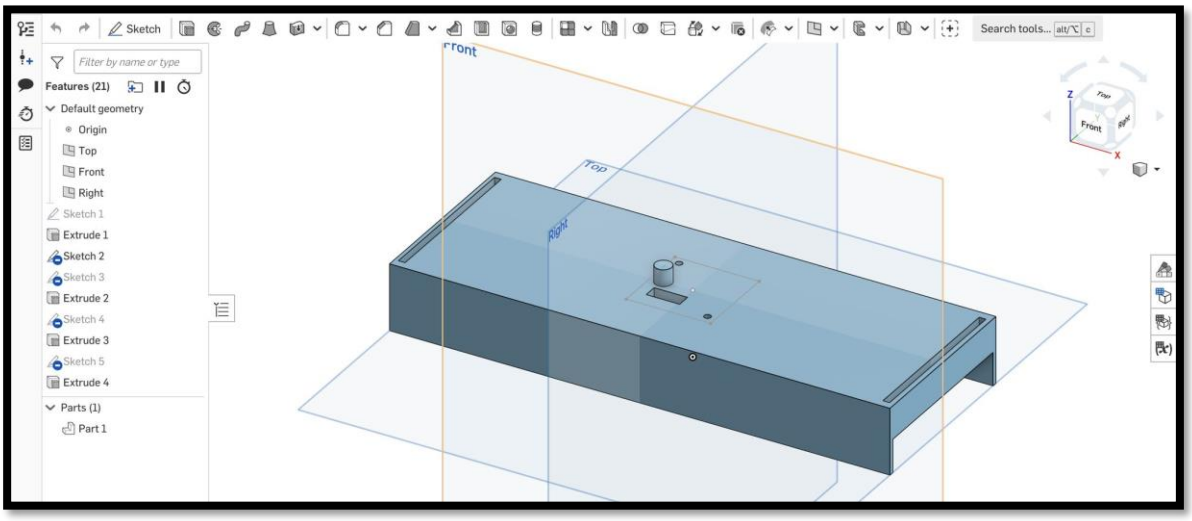
Shiny, X. S. A., Ravikumar, D., Chinnasamy, A., & Hemavathi, S. 2023. Cloud computing based smart traffic management system with priority switching for health care services. 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), 1159–1163. Accessed 24.04.2025.

Taylor, N. 2023. DSRC vs. C-V2X: Understanding the Two Technologies. Ettifos 2022. https://www.ettifos.com/post/dsrc-vs-cv2x. Accessed 07.04.2025.

Thomas Gilmore, E., Ugbome, C., & Kim, C. 2011. An IR-based pedestrian detection system implemented with matlab-equipped laptop and low-cost microcontroller. International Journal of Computer Science and Information Technology, 3(5), 79–87. Accessed 09.04.2025.

APPENDIX 1: 3D DESIGNS FOR THE SMART TRAFFIC





APPENDIX 2: 3D PRINTING OF DESIGN

APPENDIX 3: CODE FOR PROGRAMMING THE SMART TRAFFIC

```
// --- Pin Definitions ---
#define RED_PIN 7         // GPIO7
#define YELLOW_PIN 3      // GPIO3
#define GREEN_PIN 0       // GPIO0
#define PED_RED_PIN 8     // GPIO8
#define PED_GREEN_PIN 9   // GPIO9
#define PED_BUTTON 1      // GPIO1 (internal pull-up)
#define SOUND_SENSOR 2    // GPIO2 (Analog input)

// --- Timings (ms) ---
constexpr unsigned long RED_ONLY_TIME = 13000;
constexpr unsigned long GREEN_TIME = 10000;
constexpr unsigned long GREEN_BLINK_TIME = 10000;
constexpr unsigned long YELLOW_BLINK_INTERVAL = 500;
constexpr unsigned long PED_GREEN_TIME_BUTTON = 13000;
constexpr unsigned long PED_GREEN_TIME_AUTO = 8000;
constexpr unsigned long EMERGENCY_WARNING_TIME = 8000;
constexpr unsigned long EMERGENCY_GREEN_TIME = 20000;
constexpr unsigned long PED_EMERGENCY_WARNING_TIME = 7000;
constexpr unsigned long BLINK_INTERVAL = 500;
constexpr unsigned long MIN_RED_BEFORE_PEDESTRIAN = 2000; // New safety constant

// --- State Definitions ---
enum TrafficLightState {
  RED_ONLY,
  GREEN,
  GREEN_BLINKING,
  YELLOW_BLINK,
  EMERGENCY_WARNING,
  EMERGENCY_ACTIVE
};

// --- Variables ---
TrafficLightState currentState = RED_ONLY;
TrafficLightState previousState = RED_ONLY;
unsigned long stateStartTime = 0;
unsigned long blinkStartTime = 0;
unsigned long pedestrianStartTime = 0;
unsigned long pedEmergencyWarningStartTime = 0;
bool pedestrianButtonPressed = false;
bool pedestrianCrossingActive = false;
bool pedestrianBlinkState = false;
bool pedestrianEmergencyWarningActive = false;
bool greenLightState = true;
bool yellowLightState = true;
bool emergencyTriggered = false;
unsigned long pedestrianGreenTime = PED_GREEN_TIME_AUTO;
```

```
void setup() {
  Serial.begin(115200);
  pinMode(RED_PIN, OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(PED_RED_PIN, OUTPUT);
  pinMode(PED_GREEN_PIN, OUTPUT);
  pinMode(PED_BUTTON, INPUT_PULLUP);
  pinMode(SOUND_SENSOR, INPUT);

  // Initial state
  digitalWrite(RED_PIN, HIGH);
  digitalWrite(PED_RED_PIN, HIGH);
  stateStartTime = millis();
  Serial.println("[INIT] System started: RED_ONLY | Pedestrian: RED");
}

void loop() {
  unsigned long currentTime = millis();

  handlePedestrianButton();
  handleEmergencyDetection(currentTime);

  if (pedestrianEmergencyWarningActive) {
    handlePedestrianEmergencyWarning(currentTime);
    if (!pedestrianEmergencyWarningActive && emergencyTriggered) {
      currentState = EMERGENCY_WARNING;
      stateStartTime = currentTime;
      updateLights();
    }
    return;
  }

  if (currentState == EMERGENCY_WARNING) {
    if (currentTime - stateStartTime >= EMERGENCY_WARNING_TIME) {
      currentState = EMERGENCY_ACTIVE;
      stateStartTime = currentTime;
      updateLights();
    }
    return;
  }

  if (currentState == EMERGENCY_ACTIVE) {
    if (currentTime - stateStartTime >= EMERGENCY_GREEN_TIME) {
      emergencyTriggered = false;
      currentState = previousState;
      stateStartTime = currentTime;
      updateLights();
    }
    return;
  }
```

```
  handlePedestrianCrossing(currentTime);
  handleStateTransition(currentTime);
}

void handlePedestrianButton() {
  static unsigned long lastDebounceTime = 0;
  static bool lastButtonState = HIGH;
  bool currentButtonState = digitalRead(PED_BUTTON);

  if (currentButtonState != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > 50) {
    if (currentButtonState == LOW && lastButtonState == HIGH) {
      pedestrianButtonPressed = true;
      Serial.println("[INPUT] Pedestrian button pressed");
    }
  }
  lastButtonState = currentButtonState;
}

bool isEmergencySiren() {
  long sum = 0;
  for (int i = 0; i < 32; i++) {
    sum += analogRead(SOUND_SENSOR);
    delay(1);
  }
  sum >>= 5;
  //Serial.print("[SOUND] Level: ");
  //Serial.println(sum);

  // Increased sound range to 1000-1200
  return (sum >= 1200 && sum <= 1400);
}

void handleEmergencyDetection(unsigned long currentTime) {
  if (isEmergencySiren() && !emergencyTriggered) {
    emergencyTriggered = true;
    previousState = currentState;
    Serial.println("[ALERT] Emergency vehicle detected!");

    if (pedestrianCrossingActive) {
      pedestrianEmergencyWarningActive = true;
      pedEmergencyWarningStartTime = currentTime;
    } else {
      currentState = EMERGENCY_WARNING;
      stateStartTime = currentTime;
      updateLights();
    }
```

```
  }
}

void handlePedestrianEmergencyWarning(unsigned long currentTime) {
  if (currentTime - blinkStartTime >= (BLINK_INTERVAL / 2)) {
    blinkStartTime = currentTime;
    pedestrianBlinkState = !pedestrianBlinkState;
    digitalWrite(PED_GREEN_PIN, pedestrianBlinkState ? HIGH : LOW);
  }

  if (currentTime - pedEmergencyWarningStartTime >= PED_EMERGENCY_WARNING_TIME) {
    pedestrianEmergencyWarningActive = false;
    pedestrianCrossingActive = false;
    digitalWrite(PED_GREEN_PIN, LOW);
    digitalWrite(PED_RED_PIN, HIGH);
  }
}

void handlePedestrianCrossing(unsigned long currentTime) {
  // Only allow pedestrian crossing during RED_ONLY state after minimum red time
  if (currentState != RED_ONLY ) {
   if (pedestrianCrossingActive) {
      // Immediately terminate any crossing if not in RED_ONLY state
      pedestrianCrossingActive = false;
      digitalWrite(PED_GREEN_PIN, LOW);
      digitalWrite(PED_RED_PIN, HIGH);
      Serial.println("[STATE] Pedestrian crossing force-ended (not RED_ONLY)");
    }
    return;
  }

   // Rest of the existing pedestrian crossing logic...
  if (!pedestrianCrossingActive) {
    if (currentTime - stateStartTime >= MIN_RED_BEFORE_PEDESTRIAN) {
      if (pedestrianButtonPressed || (currentTime - stateStartTime >= 3000)) {
        pedestrianCrossingActive = true;
        pedestrianGreenTime = pedestrianButtonPressed ? PED_GREEN_TIME_BUTTON :
PED_GREEN_TIME_AUTO;

        pedestrianStartTime = currentTime;
        digitalWrite(PED_RED_PIN, LOW);
        digitalWrite(PED_GREEN_PIN, HIGH);
        pedestrianButtonPressed = false;
        Serial.println("[STATE] Pedestrian crossing activated");
      }
    }
  }
if (pedestrianCrossingActive && !pedestrianEmergencyWarningActive) {
    unsigned long elapsed = currentTime - pedestrianStartTime;

    // Blink pedestrian green during last 2 seconds
```

```
    if (elapsed >= pedestrianGreenTime - 2000 && elapsed < pedestrianGreenTime) {
      if (currentTime - blinkStartTime >= BLINK_INTERVAL) {
        blinkStartTime = currentTime;
        pedestrianBlinkState = !pedestrianBlinkState;
        digitalWrite(PED_GREEN_PIN, pedestrianBlinkState ? HIGH : LOW);
      }
    }
    else if (elapsed >= pedestrianGreenTime) {
      pedestrianCrossingActive = false;
      digitalWrite(PED_GREEN_PIN, LOW);
      digitalWrite(PED_RED_PIN, HIGH);
      Serial.println("[STATE] Pedestrian crossing ended");
    }
  }
}
void handleStateTransition(unsigned long currentTime) {
  unsigned long elapsed = currentTime - stateStartTime;

  switch (currentState) {
    case RED_ONLY:
      if (elapsed >= RED_ONLY_TIME) {

        // Force pedestrian red when leaving RED_ONLY state
        pedestrianCrossingActive = false;
        digitalWrite(PED_GREEN_PIN, LOW);
        digitalWrite(PED_RED_PIN, HIGH);
        currentState = GREEN;
        stateStartTime = currentTime;
        updateLights();
        Serial.println("[STATE] Transition to GREEN");
      }
      break;

    case GREEN:
      if (elapsed >= GREEN_TIME) {
        // Immediately force pedestrian red when entering blinking state
        pedestrianCrossingActive = false;
        digitalWrite(PED_GREEN_PIN, LOW);
        digitalWrite(PED_RED_PIN, HIGH);

        currentState = GREEN_BLINKING;
        stateStartTime = currentTime;
        greenLightState = true;
        blinkStartTime = currentTime;
        updateLights();
        Serial.println("[STATE] Transition to GREEN_BLINKING");
      }
      break;

    case GREEN_BLINKING:
      if (currentTime - blinkStartTime >= BLINK_INTERVAL) {
```

```
        blinkStartTime = currentTime;
        greenLightState = !greenLightState;
        digitalWrite(GREEN_PIN, greenLightState ? HIGH : LOW);
      }
      if (elapsed >= GREEN_BLINK_TIME) {
        currentState = YELLOW_BLINK;
        stateStartTime = currentTime;
        blinkStartTime = currentTime;
        yellowLightState = true;
         updateLights();
        Serial.println("[STATE] Transition to YELLOW_BLINK");
      }
      break;

    case YELLOW_BLINK:
      if (currentTime - blinkStartTime >= YELLOW_BLINK_INTERVAL) {
        blinkStartTime = currentTime;
        yellowLightState = !yellowLightState;
        digitalWrite(YELLOW_PIN, yellowLightState ? HIGH : LOW);
      }
      if (elapsed >= YELLOW_BLINK_INTERVAL * 6) {
        currentState = RED_ONLY;
        stateStartTime = currentTime;
        updateLights();
        Serial.println("[STATE] Transition to RED_ONLY");
      }
      break;

    default:
      break;
  }
}

void updateLights() {
  // Turn off all vehicle lights first
  digitalWrite(RED_PIN, LOW);
  digitalWrite(YELLOW_PIN, LOW);
  digitalWrite(GREEN_PIN, LOW);

  switch (currentState) {
    case RED_ONLY:
      digitalWrite(RED_PIN, HIGH);
      digitalWrite(YELLOW_PIN, LOW);
      // Pedestrian lights handled in handlePedestrianCrossing()
      break;

    case GREEN:
      digitalWrite(GREEN_PIN, HIGH);
      digitalWrite(YELLOW_PIN, LOW);
      digitalWrite(PED_RED_PIN, HIGH);
      digitalWrite(PED_GREEN_PIN, LOW);
```

```
    break;

  case GREEN_BLINKING:
    digitalWrite(GREEN_PIN, greenLightState ? HIGH : LOW);
    digitalWrite(YELLOW_PIN, LOW);
    digitalWrite(RED_PIN, LOW);
    // Force pedestrian red during blinking green
    digitalWrite(PED_RED_PIN, HIGH);
    digitalWrite(PED_GREEN_PIN, LOW);
    break;

  case YELLOW_BLINK:
    digitalWrite(RED_PIN, LOW);     // Ensure red is off during yellow
    digitalWrite(YELLOW_PIN, yellowLightState ? HIGH : LOW);
    digitalWrite(GREEN_PIN, LOW);  // Ensure green is off during yellow
    // Force pedestrian red during yellow
    digitalWrite(PED_RED_PIN, HIGH);
    digitalWrite(PED_GREEN_PIN, LOW);
    break;

  case EMERGENCY_WARNING:
    digitalWrite(RED_PIN, LOW);
    digitalWrite(YELLOW_PIN, HIGH); // Solid yellow for emergency warning
    digitalWrite(GREEN_PIN, LOW);
    digitalWrite(PED_RED_PIN, HIGH);
    digitalWrite(PED_GREEN_PIN, LOW);
    break;

  case EMERGENCY_ACTIVE:
    digitalWrite(RED_PIN, LOW);
    digitalWrite(YELLOW_PIN, LOW);
    digitalWrite(GREEN_PIN, HIGH);  // Solid green for emergency
    digitalWrite(PED_RED_PIN, HIGH);
    digitalWrite(PED_GREEN_PIN, LOW);
    break;
}
// Debug output
Serial.print("[LIGHTS] Vehicle: ");
switch(currentState) {
  case RED_ONLY: Serial.print("RED"); break;
  case GREEN: Serial.print("GREEN"); break;
  case GREEN_BLINKING: Serial.print("GREEN_BLINK"); break;
  case YELLOW_BLINK: Serial.print("YELLOW_BLINK"); break;
  case EMERGENCY_WARNING: Serial.print("EMERG_WARN"); break;
  case EMERGENCY_ACTIVE: Serial.print("EMERG_ACTIVE"); break;
}
Serial.print(" | Pedestrian: ");
Serial.println(pedestrianCrossingActive ? "GREEN" : "RED");
```