Group 4:

KGOTHATSO DLAMINI (4326970) , JAPHTA THABO KUBEKA (4323640) , MPHO MAKOA (4314383) , FURRELL JORDAN MEAS (4397252) , THAAKIRAH MOSOVAL (4314422) , THILITSHI MUDZUNGWANE (4335400) , KENNETH NGONO (4139497) , AVUYILE NOMPUMZA (4130844) , MOU'ATH SAIDI (4281148) , JEAN-JACQUES VAN SCHALKWYK (4204301)

Department of Computer Science,

University of the Western Cape

CSC311

For Dr Andre Henny

Table of Contents

UNIVERSITY *of the* WESTERN CAPE

Abstract

Cities need smarter ways to manage the constant dance between cars and people at intersections. I built a traffic light system that puts pedestrians first, using a Raspberry Pi as its brain. It was made so that “pedestrians: Walkers can hit a physical button to stop traffic. The system reacts instantly to this input request, mimicking the functionality of a real traffic light but with added flexibility. Three bright LEDs - red, yellow, green - mimic actual traffic signals, wired directly to the Pi's GPIO pins. The Python code controlling them follows normal traffic patterns until someone needs to cross. That's when things get interesting. The software smoothly transitions to red, gives people enough time to walk safely, then picks up the regular cycle right where it left off. For the user interface, a simple window with a button was created using Tkinter. It looks and works like those pedestrian crossing buttons we all push, just without the "Wait" light that never seems to come on fast enough. Under the hood, the system handles timing perfectly - no matter when you press the button, it won't mess up the traffic flow. Getting the timing right took considerable trial and error, where the process had to be redone multiple times to properly find a valid solution. At first, quick button presses would confuse the system, making lights change unpredictably, however this was fixed through adding clear rules for how and when the system can switch states. After dozens of tests, the system worked exactly as intended where cars get their green lights, pedestrians get safe crossing times, and everything transitions smoothly. It's a small but realistic model of how traffic systems could be more responsive to people on foot.

Introduction

Pedestrian delays at crosswalks as due to modern traffic systems leaning towards vehicle priority represents a common urban infrastructure challenge [1]. This project seeks to develop a responsive traffic light system using a Raspberry Pi to address this issue by instead employing an effective use case of pedestrian-priority functionality. The system uses Python programming to manage typical red, yellow, and green LED traffic lights, resulting in an emulated intersection that responds dynamically to human input for crossings.

The basic function of the project seeks to ensure that the traffic lights are cycling normally until a pedestrian input, pressing the button generated, is received. This ensures safe light changes without causing sudden disruptions to traffic flow, like suddenly switching from green to red causing abrupt stops, this is done by executing controlled transitions to pedestrian priority mode when it is engaged.

The system's inventiveness is in how it strikes a balance between two conflicting demands: preserving effective car circulation while enabling prompt pedestrian access. The prototype operates reliably through careful testing and programming, making it suitable for deployment at real crosswalks. The technical implementation, testing methods, and possible practical uses of this traffic control technology are covered in detail in the sections that follow.

Figure 1: Zebra Cross-walk png

Literature Review

Relevance and Importance

A developing area of study is exploring solutions to tackling problems connected to congestion, safety, and efficiency is smart traffic control. Typically, these issues are subject in causing delays for both cars and pedestrians, because of conventional traffic light systems running on pre-programmed cycles [2]. This project seeks to address those very delays for both cars and pedestrians.

Previous Work and Existing Research

Several research papers and projects have investigated the usage of embedded devices, such as Raspberry Pi, for traffic signal automation [3]. AI-driven traffic control, IoT-based adaptive signalling, and image processing for vehicle recognition have all been used in previous projects [4]. A Raspberry Pi-powered study revealed an automated traffic control system based on real-time vehicle count data, which adjusted signal timings accordingly [4]. To decrease jaywalking and increase safety, another implementation combined laser sensors with Light Dependent Resistors (LDR) for pedestrian crossings [5].
To improve safety and optimize traffic flow, advances in intelligent traffic signal systems have integrated technology like computer vision and artificial intelligence (AI) [6].

In a similar vein to what has been seen, another study created a system that provided a graphical user interface for traffic light control by measuring traffic volume in real-time using image and video processing techniques [7].

Project Differentiation

By enabling real-time crossing requests using a TCP client-server approach, this project aims to promotes pedestrian safety as opposed to the current smart traffic systems that are largely focused on giving priority to vehicles instead. By preventing missing or delayed pedestrian signals that could result in dangerous crossings, effectively putting the pedestrians at risk of harm, TCP maintains data dependability. Furthermore, with only minor hardware changes, this system may be easily integrated into current traffic control infrastructures because of its simple nature and its propensity for scalability.

Key Technologies

1. A Raspberry Pi that serves as the main controller, operating traffic lights attached to GPIO and responding to network requests.

2. Using the TCP network protocol to guarantee precise and dependable communication between the traffic management system and the pedestrian client.
3. Establishes the client-server paradigm, handles requests, and manages the LED traffic lights using Python scripts.
4. A straightforward graphical user interface that initiates the pedestrian-friendly signal phase by allowing pedestrians to transmit crossing requests through pressing a button for pedestrians crossing.

The project looks to contribute to the larger goal of smart urban mobility by utilizing these technologies to increase pedestrian accessibility and traffic efficiency.

Pedestrian Priority in Traffic Management

Ensuring pedestrian safety is a critical aspect of traffic management. [8] Research has explored various approaches to address this issue. For example, a system integrated laser sensors and Light Dependent Resistors (LDR) at zebra crossings to detect vehicles that violated red signals, activating alerts to warn pedestrians. Another project developed an automatic zebra crossing using Raspberry Pi, where motion detection mechanisms facilitated smooth pedestrian movement by synchronizing gate operations with traffic signals.

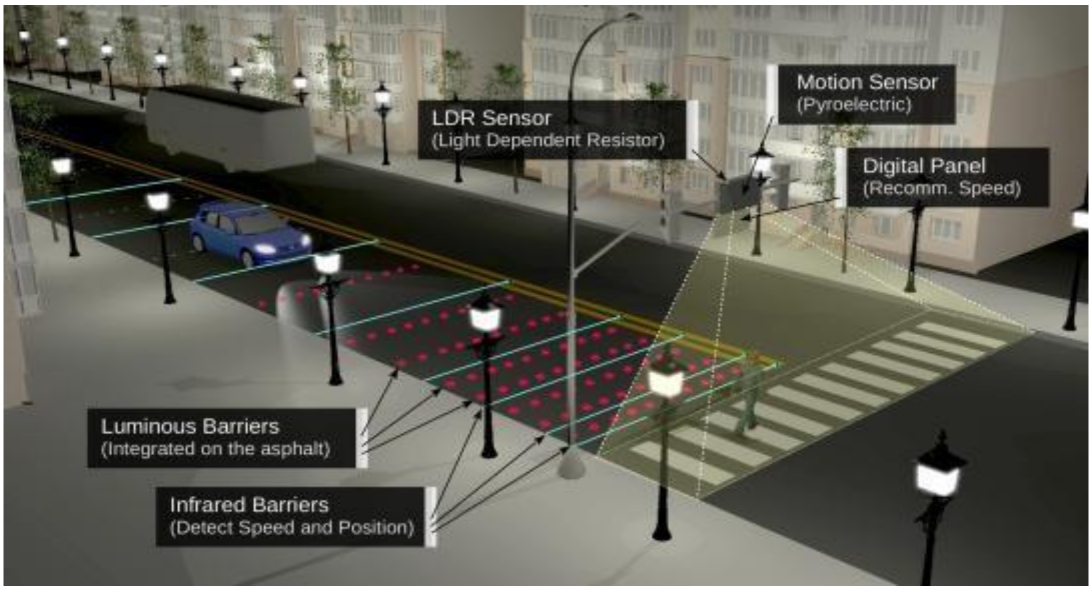

Figure 2:LDR zebra crossing png

Networking and Communication Protocols

Effective communication between the traffic light controller and user interfaces is essential for responsive operation. Implementing client-server architectures using protocols like TCP/IP enables real-time data exchange and control commands. In the context of a client-controlled traffic light system, the Raspberry Pi can serve as a server, managing traffic light states and processing pedestrian requests received from client devices, such as smartphones or dedicated terminals.

Protocol Selection: TCP vs. UDP

Reliable communication is critical in a traffic light control system where pedestrians can request crossings using a client interface. To guarantee pedestrian safety and appropriate traffic control, the Raspberry Pi server must precisely receive and process each request. Unsafe crossing circumstances or inefficient traffic could result from the loss or improper distribution of a pedestrian request.

Given TCP's reliability features, such as guaranteed delivery, error-checking, and data sequencing, it is well-suited for applications where data integrity and order are crucial. Although TCP has more overhead and is slower than UDP, the crucial nature of traffic control systems necessitates this reliability [9] [10]. As a result, using TCP in this context ensures that all pedestrian requests are properly received and managed, preserving the system's safety and effectiveness.

On the other hand, although UDP provides reduced latency and faster data transmission, it is less appropriate for applications where each data packet is crucial due to its absence of delivery guarantees and error-checking features. Despite its speed advantages, UDP is not a suitable option for traffic signal systems because of the possibility of packet loss or misallocation, which could jeopardize pedestrian safety and traffic flow [9].

Rationale for Using TCP in the Traffic Light System

Reliable communication is especially vital in the case of traffic signal management system where pedestrians can request for crossings using a client interface. The Raspberry Pi server must precisely receive and process every request to guarantee pedestrian safety and correct traffic control. The loss or misallocation of a pedestrian request could lead to unsafe crossing conditions or traffic inefficiencies.

Given TCP's reliability features, such as guaranteed delivery, error-checking, and data sequencing, it is well-suited for applications where data integrity and order are crucial. The precise nature of traffic control systems lends to the requirement for this reliability, even though TCP is slower and adds more overhead than UDP, the need for its security of packet checks is undeniable in such a system. As a result, using TCP in this situation guarantees that all pedestrian requests are received and processed appropriately, preserving the efficacy and safety of the system.

While it is given that, although, UDP provides reduced latency and faster data transmission, it is less appropriate for applications where each data packet is crucial due to its absence of delivery guarantees and error-checking features, as any such loss in this instance can lead to injury. Despite its speed advantages, UDP is not a suitable option for traffic signal systems because of the possibility of packet loss or misallocation, which could jeopardize pedestrian safety and traffic flow.

Hardware and Design

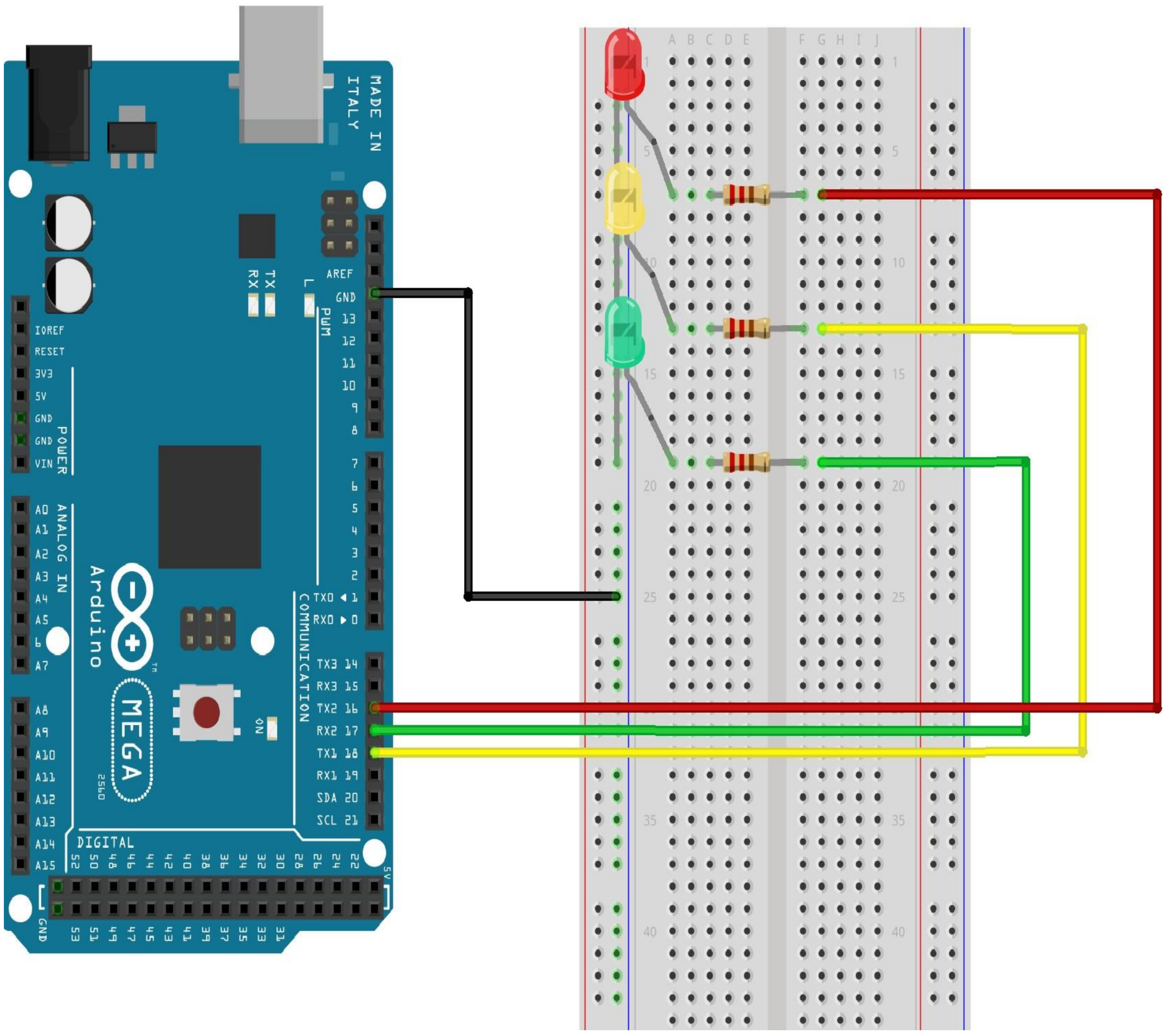

Figure 3: Fritzing of the hardware

Requirements [11]:

- Arduino
- Three LEDs (red, yellow, green)
- Three resistors
- Jumper wires
- Cable to power Arduino
- USB cable to connect mouse
- USB cable to connect keyboard
- HDMI cable to connect to PC screen
- Micro-SD to load the Raspberry Pi OS

Technical Implementation

To set up the traffic light circuit, begin by placing the red, yellow, and green LEDs vertically on the breadboard, ensuring that the long leg (anode) of each LED is in a separate row and the short leg (cathode) is positioned towards the ground rail. Next, connect the cathodes of all three LEDs to the ground (GND) rail on the breadboard. Connect three resistors respectively in series with the anode of each LED, with one end connected to the same row as the anode and the other end placed in a new row to allow jumper wire connections. Then, use jumper wires to connect the Arduino Mega's digital pins to the resistor rows: pin 16 for the red LED, pin 18 for the yellow LED, and pin 17 for the green LED. Connect one of the Arduino’s GND pins to the breadboard’s ground rail to complete the circuit. The server listens for incoming TCP connections on port 65432 and handles commands, specifically, a "pedestrian" request that triggers a special pedestrian crossing sequence. The Python script uses the gpiozero library to control the LEDs and the socket module for TCP communication. Additionally, a Tkinter GUI provides a local button to simulate pedestrian requests.

Conclusion

This project demonstrates the integration of software-driven automation with hardware-level control to create a practical, scalable, and smart traffic management system. By leveraging Python and Raspberry Pi GPIO capabilities, the design emphasizes modularity, safety, and responsiveness.

Key takeaways include:

- **Scalability:** The TCP server can be adapted to receive commands from mobile apps, IoT devices, or city-wide traffic control centres.
- **Accessibility:** Pedestrian safety is prioritized with a simple yet effective override mechanism.
- **Multithreaded Architecture:** Ensures that the user interface, networking layer, and GPIO control operate concurrently without interfering with each other.

By integrating a graphical interface and a TCP-based client, we allowed for diverse methods of interacting with the system. The experience demonstrated the practicality of combining GPIO handling with user input and remote communication, giving insight into how embedded systems can be extended to support real-time interaction in urban infrastructure.

One of the main challenges was handling concurrent inputs from both the GUI and the client connection without causing conflict in light state transitions. This was solved using global flags and managing thread-safe access. Another challenge was button debouncing—when a single press could be misread as multiple signals. Adding a short delay after button press detection helped resolve this.

This system can evolve further with features like pedestrian signal lights, timed sensors, and integration with machine learning models to optimize traffic flow. It reflects the potential for embedded systems to solve real-world challenges in urban planning and safety.

References

[1] P. o. T. David Levinson, “Traffic signals favour cars and discourage walking - study,” *news24,* 11 June 2018.

[2] M. Burdett, “Traffic congestion,” *GeographyCaseStudy.com,* 11 May 2020.

[3] J. D. M. V. T. N. P. P. e. a. Dr. Ch. Rambabu, “Intelligent Traffic Signal Management Using,” *International Journal of Novel Research and Development (IJNRD),* vol. IX, no. 6, pp. 1-5, 2024.

[4] S. P. M. A. Mohammed Sarrab, “Development of an IoT based real-time traffic monitoring system for city governance,” *Glabal Transitions,* vol. ||, pp. 231233.

[5] W. K. S. W. A. J. A.-A. Yasir Hashim, “Design and Implementation of Portable Smart Wireless Pedestrian Crossing Control System,” *IEEE Access,* vol. VIII, pp. 5-9, 2020.

[6] X. Zhang, “Artificial Intelligence in Intelligent Traffic Signal Control,” in *3rd International Conference on Software Engineering and Machine Learning*, 2025.

[7] “Real-time traffic control and monitoring,” *e-Prime - Advances in Electrical Engineering, Electronics and Energy,* vol. V, 2023.

[8] S. D. P. ,. I. O. V. ,. J.-M. ,. J. ,. L.-V. ,. V. a. B. Volodymyr N. Skoropad, “Dynamic Traffic Flow Optimization Using Reinforcement Learning and Predictive Analytics: A Sustainable Approach to Improving Urban Mobility in the City of Belgrade,” *Sustainability,* pp. 2-3, 2025.

[9] N. Hassan, “Exploring the Differences Between TCP and UDP: A Deep-Dive Comparison,” *Medium,* 10 October 2024.

[10] GeeksforGeeks, “What is TCP (Transmission Control Protocol)?,” *GeeksforGeeks,* 1 February 2025.

[11] S. Shanbhag, “YouTube,” 18 March 2021. [Online]. Available: https://www.youtube.com/watch?v=QAESp0pqyJo. [Accessed April 2025].

Code Explanation:

The code implements a responsive traffic light system using the Raspberry Pi. Below is a comprehensive breakdown of everything that was employed to setup the traffic light system and ensure that it runs precisely as it sould:

- **LED Initialization:** The gpiozero.LED class is used to control each LED representing traffic light colors. GPIO pins 16, 18, and 17 are assigned to red, yellow, and green respectively.
- **TCP Server:** A socket is created and bound to IP 0.0.0.0 and port 65432, allowing any device on the local network to connect. This enables remote pedestrian requests through a client device. The handle_client function handles each incoming connection on a new thread, decoding any data received. If the string "pedestrian" is detected, it sets the pedestrian_request flag.
- **Threading:** Python's Thread class is used to run the TCP server and traffic light logic in parallel. This ensures the GUI remains responsive and the traffic logic continues to operate regardless of incoming TCP connections.
- **Pedestrian Request Logic:** A global flag pedestrian_request is checked continuously in the traffic light loop. If it is set to True, the system prioritizes pedestrian crossing by immediately transitioning to a red light and holding it longer (7 seconds), ensuring safe crossing.
- **Traffic Light Sequence:** The normal operation cycles through red (5s), green (5s), and yellow (2s) lights. If a pedestrian request is made during this cycle, the loop gracefully transitions to red before allowing crossing.
- **Graphical Interface:** A simple GUI is created with Tkinter, providing a local button that can simulate a pedestrian crossing request. This is helpful for testing or using the system without a physical button.

The code begins by importing necessary libraries—gpiozero for controlling the GPIO pins, time for delay handling, tkinter for the GUI, threading for running background processes, and socket for client-server communication. Three GPIO pins are used to represent traffic lights: red (pin 16), yellow (pin 18), and green (pin 17). The Raspberry Pi acts as a server, listening for TCP messages on port 65432. When it receives the word "pedestrian" from a client, it sets a global flag to trigger pedestrian priority in the main traffic loop. The traffic_light_loop() function governs the behavior of the lights. In standard operation, it cycles through red, green, and yellow with appropriate delays. When a pedestrian request is flagged, the system transitions safely to red and holds that state longer to allow for safe crossing. After handling the request, the system returns to its regular loop. The GUI consists of a single button labeled “Pedestrian Crossing.” When pressed, it triggers the same pedestrian request behavior as a remote command would. This dual-input system ensures flexibility and enhances realism. The TCP server and the traffic light loop both run on separate daemon threads so that they operate continuously in the background while the GUI remains responsive.

Individual Contributions:

Group Member	Contribution
Mou'ath Saidi (4281148)	Responsible for research for literature review and report compilation, as well as narrating the rationale behind our choice of network protocol in the presentation
KGOTHATSO DLAMINI (4326970)	Responsible for narrating a section of the ppt presentation
JAPHTA THABO KUBEKA (4323640)	Responsible for narrating a section of the ppt presentation
MPHO MAKOA (4314383)	Responsible for research, hardware setup and correcting any grammatical errors that were found in the report. As well as narrating the problem statement.
FURRELL JORDAN MEAS (4397252)	Responsible for research, finding a proper hardware setup fritzing. As well as narrating the over the project video and hardware requirements in the presentation.
THAAKIRAH MOSOVAL (4314422)	Responsible for setting up the PowerPoint presentation, aiding in research as well as presenting the conclusion of the presentation as narration.
THILITSHI MUDZUNGWANE (4335400)	Responsible for editing the video of the project as well as aiding in the hardware setup, aiding with the coding and general proofreading
KENNETH NGONO (4139497)	Responsible for aiding with the coding, compilation of the document as well as the PowerPoint presentation
AVUYILE NOMPUMZA (4130844)	Responsible for narrating a section of the ppt presentation
JEAN-JACQUES VAN SCHALKWYK (4204301)	Responsible for coding and the explanation of that code both in the report and narrated in the presentation.