Cognitive Robotics Lab
Lab 1: Smart Traffic Light Systems using RPI GPIO Pins

Srikar Ganesh 22BAI1080

04-08-2025

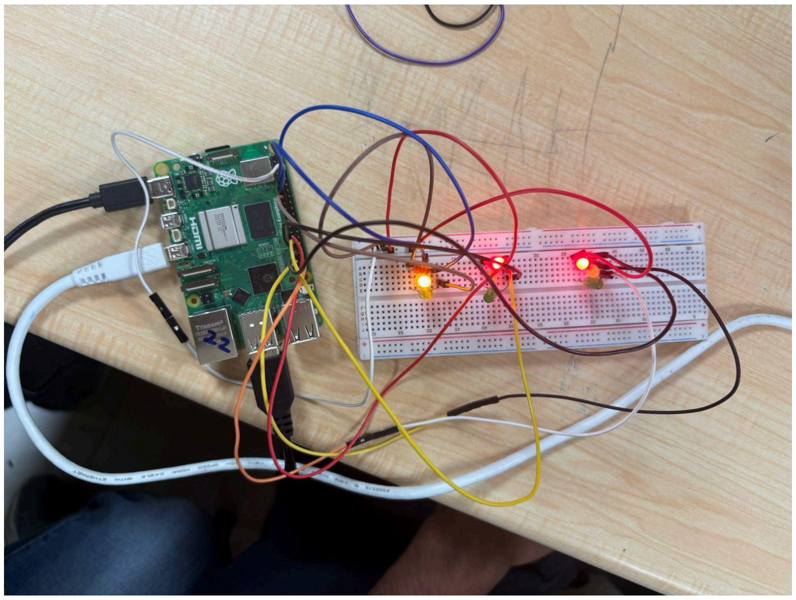
Aim:

To design and implement a smart traffic light system using Raspberry Pi GPIO pins that simulates traffic light operation with basic logic and sensor input handling, thereby understanding GPIO interfacing, traffic flow control, and basic automation principles.

Components:

1. **Raspberry Pi (RPI)** – any model with GPIO support (e.g., RPi 3B+, RPi 4)
2. **MicroSD Card** – with Raspbian OS installed
3. **Monitor** – for displaying the Raspberry Pi desktop interface
4. **HDMI Cable** – to connect Raspberry Pi to the monitor
5. **Keyboard and Mouse** – for user input and program control
6. **Breadboard** – for prototyping the circuit
7. **Jumper Wires (Male-to-Male)** – for connections between the GPIO pins and components
8. **LEDs** – 3 LEDs (Red, Yellow, Green) to represent traffic lights
9. **Resistors** – typically 220Ω or 330Ω for current limiting with LEDs
10. **Power Supply for Raspberry Pi** – 5V 2.5A (via USB-C or micro-USB depending on model)

Output:



Source Code:

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

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Define GPIO pins for each lane's LEDs
PIN_MAP = {
    'A': {'R': 17, 'Y': 27, 'G': 22},
    'B': {'R': 5,  'Y': 6,  'G': 13},
    'C': {'R': 19, 'Y': 26, 'G': 21},
}

# Setup all pins as outputs and turn them off initially
for lane_pins in PIN_MAP.values():
    for pin in lane_pins.values():
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, GPIO.LOW)

# Queues and ambulance flags
queue = {'A': [], 'B': [], 'C': []}
ambulance_flags = {'A': False, 'B': False, 'C': False}

def set_light(lane, color):
    """Set the specified light color ON, others OFF for the lane."""
    pins = PIN_MAP[lane]
    # Turn all off first
    for c in ['R', 'Y', 'G']:
        GPIO.output(pins[c], GPIO.LOW)
    # Turn the requested color ON
    GPIO.output(pins[color], GPIO.HIGH)

def all_red():
    for lane in ['A', 'B', 'C']:
        set_light(lane, 'R')

def update_ambulance_flags():
    for lane in ['A', 'B', 'C']:
        ambulance_flags[lane] = 'A' in queue[lane]

def add_random_cars():
    for lane in ['A', 'B', 'C']:
        new_cars = random.randint(1, 3)
        PR = random.random()
        if PR < 0.1:
            queue[lane] += ['A'] * new_cars
            print(f"Lane {lane}: Ambulance(s) arrived! Added {new_cars} ambulances.")
        else:
            queue[lane] += ['C'] * new_cars
            print(f"Lane {lane}: Added {new_cars} new cars.")
        print(f"Lane {lane}: Total vehicles now: {len(queue[lane])}")

def pick_ambulance_lane():
    for lane in ['A', 'B', 'C']:
        if ambulance_flags[lane]:
            return lane
    return None

def pick_next_lane():
    ambulance_lane = pick_ambulance_lane()
    if ambulance_lane:
        return ambulance_lane
    max_lane = max(queue, key=lambda x: len(queue[x]))
    return max_lane if queue[max_lane] else None

def simulate_car_movement_with_preemptive(lane, max_duration):
    print(f"\n--- Starting Yellow Light on lane {lane} ---")
    set_light(lane, 'Y')
    time.sleep(2)

    print(f"--- Green Light on lane {lane} for up to {max_duration} seconds (ambulance priority
active) ---")
    set_light(lane, 'G')

    seconds_passed = 0
    while seconds_passed < max_duration:
        update_ambulance_flags()
        ambulance_lane = pick_ambulance_lane()

        # Preempt if ambulance in different lane
        if ambulance_lane and ambulance_lane != lane:
            print(f"\n*** Preempting lane {lane} to serve ambulance in lane {ambulance_lane} ***")
            break

        if queue[lane]:
            car = queue[lane].pop(0)
            print(f"Lane {lane}: 1 {'ambulance' if car == 'A' else 'car'} moved out, remaining cars:
{len(queue[lane])}")
        else:
            print(f"Lane {lane}: No cars left to move.")
            break

        seconds_passed += 1
        time.sleep(1)

    print(f"--- Ending with Yellow Light on lane {lane} ---")
    set_light(lane, 'Y')
    time.sleep(2)

    print(f"--- Switching to Red Light on lane {lane} ---")
    set_light(lane, 'R')

    return seconds_passed

def main():
    cycle_duration = 30
    simulation_minutes = 3
    total_seconds = simulation_minutes * 60
    start_time = time.time()
    last_car_add_time = time.time()

    add_random_cars()
    update_ambulance_flags()
    all_red()

    try:
        while time.time() - start_time < total_seconds:
            now = time.time()

            # Add new cars every 30 seconds
            if now - last_car_add_time >= 30:
                add_random_cars()
                update_ambulance_flags()
                last_car_add_time = now

            selected_lane = pick_next_lane()

            if selected_lane:
                print(f"\n*** Selected Lane for green light: {selected_lane} ***")
                simulate_car_movement_with_preemptive(selected_lane, cycle_duration)
            else:
                print("No cars in any lane, all lights Red.")
                all_red()
                time.sleep(5)

    except KeyboardInterrupt:
        print("Exiting, cleaning up GPIO...")

    finally:
        all_red()
        GPIO.cleanup()

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

Explanation:

The provided Python script simulates a smart traffic light system using Raspberry Pi GPIO pins. It controls three traffic lanes (A, B, and C), each with red, yellow, and green LEDs. Vehicles, including ambulances (denoted by 'A') and regular cars ('C'), are randomly added to each lane. The system prioritizes lanes based on vehicle count and preempts the current lane if an ambulance is detected in another. The lights are controlled using GPIO pins, transitioning through standard traffic light sequences. Real-time decisions are made every 30 seconds, simulating dynamic traffic and emergency handling using queues and a priority-based scheduling mechanism.

Result:

The smart traffic light system was successfully implemented on the Raspberry Pi. The system accurately simulated real-world traffic light behavior with automatic switching between red, yellow, and green signals. It dynamically responded to vehicle buildup and prioritized ambulance movement by preempting current green phases, ensuring timely passage for emergency vehicles. The queues for each lane were efficiently managed, and sensor simulation via random generation helped test varying traffic conditions. This experiment deepened understanding of GPIO control, traffic automation algorithms, and real-time decision-making processes in embedded systems.