**Intelligent Traffic Management System with Adaptive Signal Control**

Intelligent traffic management system using Raspberry Pi and YOLO for vehicle detection and adaptive signal control.

**PROJECT OVERVIEW**

**Objective**

* Detect and count vehicles on **two lanes (lane\_1 & lane\_2)** using live video feed from RPi CSI camera.
* Based on vehicle count, dynamically adjust the **LED-based traffic light timers**.
* Display everything in a **Tkinter-based GUI**:
  + Live video with bounding boxes
  + Traffic light status and timing (initial & updated)

**REQUIRED HARDWARE**

Since you already have the Raspberry Pi 5, camera, LEDs, breadboard, and jumper wires, here’s the rest:

**Hardware List**

| **Item** | **Description** | **Quantity** |
| --- | --- | --- |
| Raspberry Pi 5 | Main controller | 1 |
| Raspberry Pi CSI Camera | For real-time video feed | 1 |
| Breadboard | For connecting LEDs and GPIO | 1 |
| LEDs | Red and Green (for 2 lanes) | 4 |
| Resistors (220Ω or 330Ω) | For LED protection | 4 |
| Jumper Wires (Male-Male) | For connections | 10+ |
| Optional: Push button | For manual override (optional) | 2 |
| Power Supply for RPi | Official 5V/3A adapter | 1 |

**SOFTWARE DESIGN (Object-Oriented Structure)**

**📂 Project Folder Structure**

traffic\_light\_project/

├── camera/

│ └── video\_stream.py # Handles live camera feed

├── detection/

│ └── vehicle\_detector.py # YOLO-based vehicle detection logic

├── gpio/

│ └── traffic\_light.py # GPIO interface for controlling traffic lights

├── ui/

│ └── gui\_window.py # Tkinter-based GUI display

├── core/

│ └── controller.py # Main logic: coordination of detection, timing, and GPIO

├── models/

│ └── yolov5s.pt # Pretrained YOLO model (if using torch hub or ONNX)

├── utils/

│ └── timer\_logic.py # Dynamic wait time logic

├── main.py # Entry point to run the application

└── requirements.txt # Python packages needed

**🧱 MODULE BREAKDOWN**

**1. camera/video\_stream.py**

Handles camera initialization and frame grabbing.

class VideoStream:

def \_\_init\_\_(self, use\_usb=False):

pass

def read(self):

"""Returns a frame from the video stream."""

pass

def release(self):

pass

**2. detection/vehicle\_detector.py**

YOLOv5/YOLOv8 detection wrapper.

class VehicleDetector:

def \_\_init\_\_(self, model\_path="models/yolov5s.pt"):

pass

def detect(self, frame):

"""

Returns:

- annotated\_frame: with bounding boxes

- counts: dict like {'lane\_1': 4, 'lane\_2': 2}

"""

pass

**3. gpio/traffic\_light.py**

Controls GPIO output for red/green LEDs per lane.

class TrafficLight:

def \_\_init\_\_(self, red\_pin, green\_pin):

pass

def set\_state(self, state: str):

"""State can be 'RED' or 'GREEN'"""

pass

**4. utils/timer\_logic.py**

Simple heuristic: For every extra car in lane\_1 vs lane\_2, reduce lane\_1 time by 5s.

def compute\_new\_timings(base\_time: int, count1: int, count2: int):

"""

Returns: (new\_time\_1, new\_time\_2)

"""

pass

**5. ui/gui\_window.py**

Tkinter GUI window:

* Live video canvas
* Labels for lights and timers
* Display updated values

class TrafficGUI:

def \_\_init\_\_(self, controller):

pass

def update\_frame(self, frame, count1, count2, timer1, timer2):

pass

def run(self):

pass

**6. core/controller.py**

Main control loop: acquire frame → detect → count → update timers → control GPIO → update GUI

class TrafficController:

def \_\_init\_\_(self):

pass

def process\_frame(self):

"""Core method to handle 1 iteration: detection + LED update + GUI update"""

pass

**7. main.py**

Entry point to start everything:

from core.controller import TrafficController

from ui.gui\_window import TrafficGUI

if \_\_name\_\_ == "\_\_main\_\_":

controller = TrafficController()

gui = TrafficGUI(controller)

gui.run()