Adaptive Traffic Signal Control for Emergency Vehicles Using Deep Learning and Acoustic Analysis code

```
#include <LiquidCrystal.h>

#define RED_LIGHT 3
#define YELLOW_LIGHT 4
#define GREEN_LIGHT 5
#define SOUND_SENSOR A0
#define EMERGENCY_BUTTON 2

// 16x2 LCD (RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

int sirenThreshold = 500;

void setup() {
  pinMode(RED_LIGHT, OUTPUT);
  pinMode(YELLOW_LIGHT, OUTPUT);
  pinMode(GREEN_LIGHT, OUTPUT);
  pinMode(EMERGENCY_BUTTON, INPUT_PULLUP);
  Serial.begin(9600);

  // Initialize LCD
  lcd.begin(16, 2);
  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Traffic Control");

  // Default: Red light ON
  digitalWrite(RED_LIGHT, HIGH);
  digitalWrite(YELLOW_LIGHT, LOW);
  digitalWrite(GREEN_LIGHT, LOW);

  delay(1000);  // Allow LCD to stabilize
}

void loop() {
```

```
    int soundLevel = analogRead(SOUND_SENSOR);
    bool manualOverride = digitalRead(EMERGENCY_BUTTON) == LOW;

    Serial.print("Sound Level: ");
    Serial.println(soundLevel);

    lcd.clear();  // Prevent overlapping text

    if (manualOverride) {
        Serial.println("Manual Override: Green Light ON");
        lcd.setCursor(0, 0);
        lcd.print("Manual Override!");
        lcd.setCursor(0, 1);
        lcd.print("Green Light ON");

        digitalWrite(RED_LIGHT, LOW);
        digitalWrite(YELLOW_LIGHT, LOW);
        digitalWrite(GREEN_LIGHT, HIGH);
    }
    else if (soundLevel > sirenThreshold) {
        Serial.println(" Ambulance Detected! Turning Green Light ON");
        lcd.setCursor(0, 0);
        lcd.print("Ambulance Near!");
        lcd.setCursor(0, 1);
        lcd.print("Green Light ON");

        digitalWrite(RED_LIGHT, LOW);
        digitalWrite(YELLOW_LIGHT, LOW);
        digitalWrite(GREEN_LIGHT, HIGH);
    }
    else {
        Serial.println(" Normal Traffic Mode");
        lcd.setCursor(0, 0);
        lcd.print("Normal Traffic");
        lcd.setCursor(0, 1);
        lcd.print("Red Light ON");

        digitalWrite(GREEN_LIGHT, LOW);
        digitalWrite(YELLOW_LIGHT, LOW);
        digitalWrite(RED_LIGHT, HIGH);
    }

    delay(500);  // Stable LCD updates
}
```

```
import cv2
import torch
import time
from collections import deque

# Load YOLOv5 model from PyTorch Hub
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', force_reload=True)
model.classes = [2, 3, 5, 7]

# Initialize camera
cap = cv2.VideoCapture(0)

# Rolling window to smooth counts
vehicle_window = deque(maxlen=10)

# Signal state control
SIGNAL_STATES = ["RED", "GREEN", "YELLOW"]
current_state = "RED"
last_state_change_time = time.time()

# Weights
VEHICLE_WEIGHTS = {
    2: 3,  # Car
    3: 2,  # Motorbike
    5: 5,  # Bus
    7: 5   # Truck
}

# Signal timing
green_duration = 10
yellow_duration = 5
red_duration = 10

# Get signal duration based on vehicle count
def calculate_dynamic_green_time(vehicle_counts):
    total_time = 0
    for cls, count in vehicle_counts.items():
        total_time += VEHICLE_WEIGHTS.get(cls, 0) * count
    return max(10, min(total_time, 60))

# Count vehicle types in one frame
def count_vehicles(detections):
    counts = {2: 0, 3: 0, 5: 0, 7: 0}
    for det in detections:
        cls = int(det[5])
        if cls in counts:
            counts[cls] += 1
```

```
    return counts

# Display signal state
def draw_signal(frame, state, duration):
    color_map = {"RED": (0, 0, 255), "GREEN": (0, 255, 0), "YELLOW": (0, 255, 255)}
    cv2.rectangle(frame, (10, 10), (160, 90), color_map[state], -1)
    cv2.putText(frame, f"{state}", (20, 60), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 3)
    cv2.putText(frame, f"{int(duration)}s", (90, 60), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
0), 2)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame)
    detections = results.xyxy[0]

    vehicle_counts = count_vehicles(detections)
    vehicle_window.append(vehicle_counts)

    avg_counts = {k: 0 for k in VEHICLE_WEIGHTS}
    for c in vehicle_window:
        for k in c:
            avg_counts[k] += c[k]
    avg_counts = {k: v // len(vehicle_window) for k, v in avg_counts.items()}

    current_time = time.time()
    elapsed_time = current_time - last_state_change_time

    if current_state == "GREEN":
        if elapsed_time >= green_duration:
            current_state = "YELLOW"
            last_state_change_time = current_time
    elif current_state == "YELLOW":
        if elapsed_time >= yellow_duration:
            current_state = "RED"
            last_state_change_time = current_time
    elif current_state == "RED":
        if elapsed_time >= red_duration:
            green_duration = calculate_dynamic_green_time(avg_counts)
            current_state = "GREEN"
            last_state_change_time = current_time

    results.render()
    annotated_frame = results.imgs[0]
    draw_signal(annotated_frame, current_state, max(0, int(green_duration - (time.time() -
last_state_change_time))) if current_state == "GREEN" else int(green_duration))
```

```
    cv2.imshow("YOLO Traffic Control", annotated_frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```