REVIEW -2

Effective Traffic Management System Using OpenCV:

Introduction

Traffic congestion is a pervasive urban problem that significantly impacts quality of life, economy, and environment. Effective traffic management systems (TMS) are crucial to mitigate these issues. This review explores the development of a TMS using OpenCV, a popular computer vision library, coupled with Arduino for real-time control and hardware interaction.

System Architecture

The proposed TMS consists of the following components:

1. **Image Acquisition:** A camera (e.g., webcam, IP camera) captures real-time video footage of the traffic intersection.
2. **Image Processing:** OpenCV is employed to analyse the captured images and extract relevant information, such as vehicle count, vehicle types, and traffic flow direction.
3. **Decision Making:** Based on the extracted data, the system makes intelligent decisions, like adjusting traffic signal timings, diverting traffic, or alerting authorities about congestion.
4. **Hardware Control:** Arduino is used to interface with external hardware, such as traffic lights, sensors, and displays, to implement the decisions made by the system.

Code Snippets and Implementation

1. Image Acquisition and Object Detection:

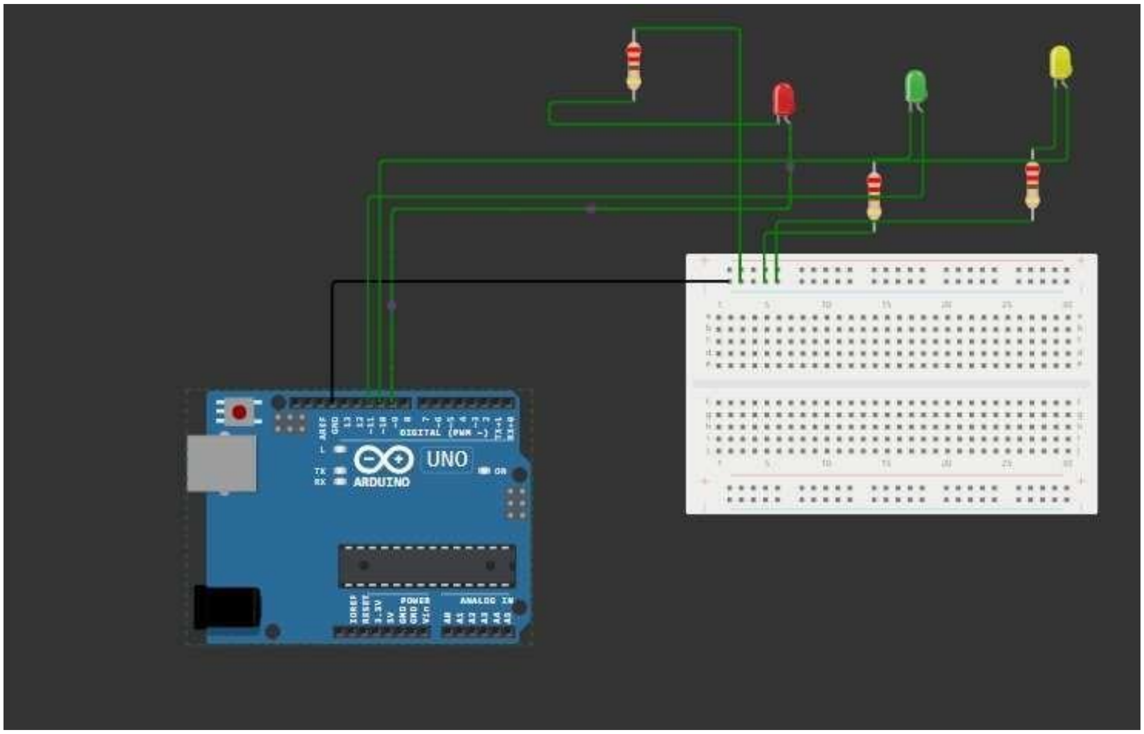
```python
import cv2
import numpy as np
import serial
import time
import struct

# Set up serial communication with Arduino
arduino = serial.Serial('COM5', 9600, timeout=1)

def intersection_over_union(box1, box2):
    x1, y1, w1, h1, _ = box1
    x2, y2, w2, h2, _ = box2
    intersection_area = max(0, min(x1 + w1, x2 + w2) - max(x1, x2)) * max(0, min(y1 + h1, y2 + h2) - max(y1, y2))
    union_area = w1 * h1 + w2 * h2 - intersection_area
    return intersection_area / union_area

# Load the YOLOv3 model
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

# Load the COCO dataset classes
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

# Get the image path from the user
img_path = input("Enter the image path: ")
img_path = img_path.strip('"')

# Load the image
img = cv2.imread(img_path)

# Check if the image was loaded successfully
if img is None:
    print("Error: Unable to read image file.")
    exit()

# Get the image height and width
height, width, _ = img.shape

# Create a blob and set it as the input for the model
blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), swapRB=True, crop=False)
net.setInput(blob)

# Run the model and get the output
outputs = net.forward(net.getUnconnectedOutLayersNames())

# Create a list to store the detected vehicles
vehicles = []

# Loop through the outputs
```

```python
for output in outputs:
    for detection in output:
        # Get the scores, class_id, and confidence
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        # Check if the detected object is a vehicle
        if classes[class_id] in ["car", "truck", "bus", "motorbike"]:
            # Get the bounding box coordinates
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            # Append the vehicle to the list
            vehicles.append((x, y, w, h, confidence))

# Apply Non-Maximum Suppression (NMS)
threshold = 0.4
vehicles_after_nms = []
while len(vehicles) > 0:
    vehicle = vehicles[0]
    vehicles_after_nms.append(vehicle)
    vehicles.remove(vehicle)
    for other_vehicle in vehicles[:]:
        if intersection_over_union(vehicle, other_vehicle) > threshold:
            vehicles.remove(other_vehicle)

# Send the vehicle count to Arduino
vehicle_count = len(vehicles_after_nms)
arduino.write(struct.pack('B', vehicle_count))

# Give Arduino some time to process
time.sleep(2)

arduino.close()
print("Number of vehicles detected:", vehicle_count)

# Draw the bounding boxes around the vehicles
for vehicle in vehicles_after_nms:
    cv2.rectangle(img, (vehicle[0], vehicle[1]), (vehicle[0] + vehicle[2], vehicle[1] + vehicle[3]), (0, 255, 0), 2)

# Display the image
cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. Arduino Integration:

```
int redLight = 9;
int yellowLight = 10;
int greenLight = 11;

int vehicleCount = 0;

void setup() {
  pinMode(redLight, OUTPUT);
  pinMode(yellowLight, OUTPUT);
  pinMode(greenLight, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    vehicleCount = Serial.read();
    if(vehicleCount>50){
      digitalWrite(redLight, LOW);
      digitalWrite(yellowLight, LOW);
      digitalWrite(greenLight, HIGH);
      delay(50000);

      digitalWrite(redLight, LOW);
      digitalWrite(yellowLight, HIGH);
      digitalWrite(greenLight, LOW);
      delay(5000);

      digitalWrite(redLight, HIGH);
      digitalWrite(yellowLight, LOW);
      digitalWrite(greenLight, LOW);
      delay(5000);
    }
    else{
      digitalWrite(redLight, LOW);
      digitalWrite(yellowLight, LOW);
      digitalWrite(greenLight, HIGH);
      delay(1000*vehicleCount);

      digitalWrite(redLight, LOW);
      digitalWrite(yellowLight, HIGH);
      digitalWrite(greenLight, LOW);
      delay(5000);

      digitalWrite(redLight, HIGH);
      digitalWrite(yellowLight, LOW);
      digitalWrite(greenLight, LOW);
      delay(5000);
    }
  }
}
```

3. **Hardware Setup:**

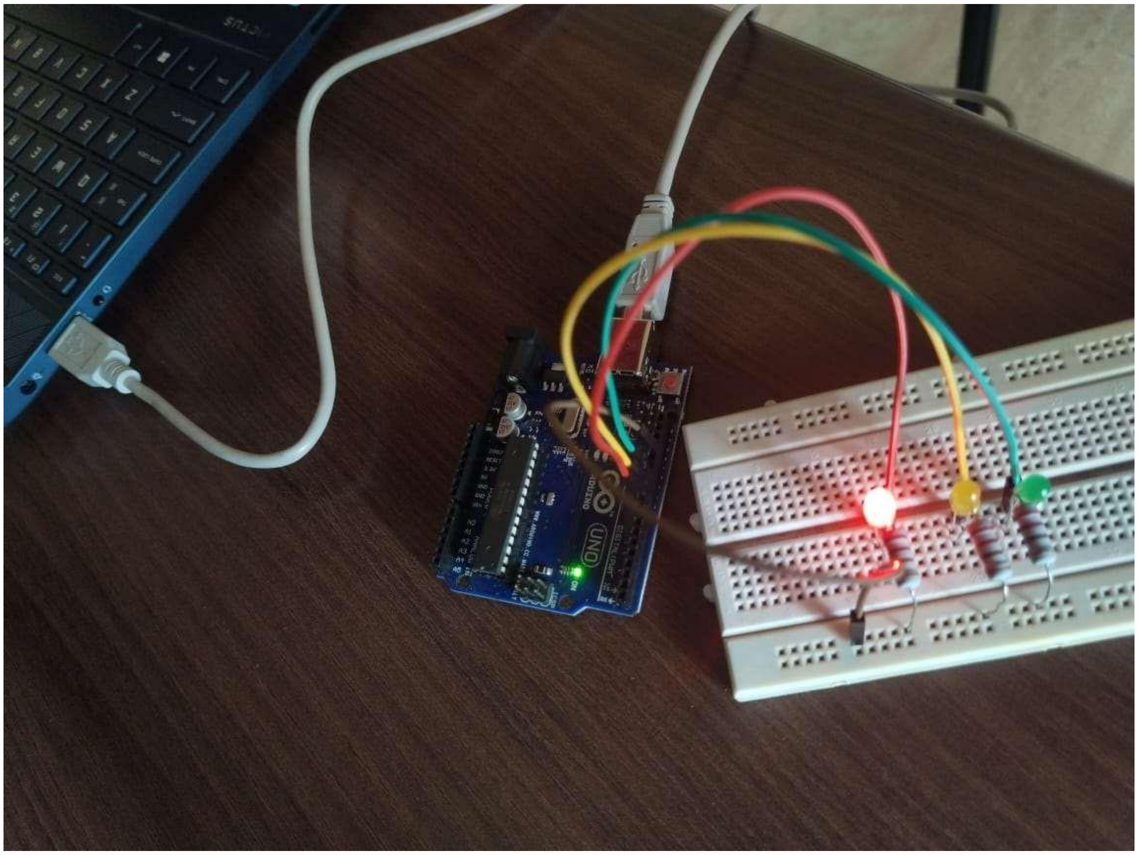


breadboard with LEDs, resistors, and Arduino

Hardware components:

- Arduino Uno
- Breadboard
- LEDs (red, yellow, green)
- Resistors
- Jumper wires

Circuit diagram:



circuit diagram for traffic light control using Arduino

5. Integration and Testing:

- Connect the Arduino to your computer.
- Upload the Arduino code.
- Run the Python script.
- Test the system's functionality by simulating traffic scenarios.

Conclusion

The development of an effective traffic management system using OpenCV and Arduino offers a promising solution to address urban traffic congestion. By combining computer vision techniques with real-time control capabilities, this system can improve traffic flow, reduce accidents, and enhance overall quality of life. Future advancements in this field may include incorporating more sophisticated algorithms, integrating with IoT devices, and leveraging machine learning for adaptive traffic management.