**OBJECT TRACKING PROJECT**

# 1. Objective

The goal of this task is to develop an object tracking-based application. This application will:

1. Track objects in a video
2. Calculate and display specific metrics
3. Visualise tracking results
4. Export and display the output in an HTML window.

**2. Approach**

* **Detection**: Use YOLOv8 for object detection due to its speed and accuracy.
* **Tracking**: Match object centroids across frames to maintain consistent object IDs.
* **Visualization**:
  + Draw bounding boxes around detected objects.
  + Mark centroids for tracking.
  + Visualize object movement with trails.

**3. Tech Stack**

* **Programming Language**: Python
* **Object Detection Model** : YOLO (yolov8)
* **Libraries**:
  + **OpenCV**: For video processing and visualization.
  + **NumPy**: For mathematical operations.
  + **OS**: For handling file paths and system operations.
* **Output Formats**:
  + MP4 for processed video.
  + HTML for embedding and displaying results in a browser.

**4. Implementation Details**

**1. YOLO Model Selection:**

YOLO (You Only Look Once) is a real-time object detection algorithm developed by Joseph Redmon and Ali Farhadi in 2015. It is a single-stage object detector that uses a convolutional neural network (CNN) to predict the bounding boxes and class probabilities of objects in input images.

**YOLOv8** is the latest iteration in the YOLO series of real-time object detectors, offering cutting-edge performance in terms of accuracy and speed.

In Yolov8 , we use PreTrained Model **Yolov8n.pt** to detect object in real time.

2 . **Video Processing**:

* Read frames from the input video using OpenCV.
* Extract video properties like FPS, width, and height.

3. **Object Tracking**:

* Use a centroid-based approach to track objects across frames.
* Match objects based on proximity to the last known centroid.

4. **Visualization**:

* Draw bounding boxes, centroids, and trails for tracked objects.
* Label each object with its class name (e.g., "person").

5. **Metrics Calculation**:

* Count unique objects detected.
* Compute time spent in the video for each object.

6. **Output Generation**:

* Save the processed video.
* Create an HTML file with the embedded video for browser viewing.

**5. CODING STEPS:**

**1. Import Packages:**

import cv2

import numpy as np

from ultralytics import YOLO

from collections import defaultdict

import os

**2. Import Model:**

model = YOLO("yolov8n.pt")

**3. Video Capture & its Properties:**

cap = cv2.VideoCapture(input\_video\_path)

fps = int(cap.get(cv2.CAP\_PROP\_FPS))

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

input\_video\_path = "input\_video.mp4"

output\_video\_path = "output\_video.mp4

**Output Video**

fourcc = cv2.VideoWriter\_fourcc(\*"mp4v")

out = cv2.VideoWriter(output\_video\_path, fourcc, fps, (frame\_width, frame\_height))

**4. Object tracking data**

object\_data = defaultdict(lambda: {"frames": [], "centroids": []})

next\_object\_id = 1

Object\_id\_mapping = {}

**5. Function to calculate centroid of a bounding box**

def calculate\_centroid(box):

x\_min, y\_min, x\_max, y\_max = box

return int((x\_min + x\_max) / 2), int((y\_min + y\_max) / 2)

**6. Collect Bounding boxes , Class IDs , Class Names**

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

results = model(frame)

detections = results[0].boxes.xyxy.cpu().numpy()

class\_ids = results[0].boxes.cls.cpu().numpy().astype(int)

class\_names = [model.names[class\_id] for class\_id in class\_ids]

frame\_data = []

for det, class\_name in zip(detections, class\_names):

x\_min, y\_min, x\_max, y\_max = map(int, det)

centroid = calculate\_centroid((x\_min, y\_min, x\_max, y\_max))

**Match with existing objects**

matched = False

for obj\_id, data in object\_data.items():

if data["frames"] and np.linalg.norm(np.array(data["centroids"][-1]) - np.array(centroid)) < 50: # Match threshold

object\_data[obj\_id]["frames"].append(len(object\_data[obj\_id]["frames"]) + 1)

object\_data[obj\_id]["centroids"].append(centroid)

frame\_data.append((obj\_id, (x\_min, y\_min, x\_max, y\_max), centroid, class\_name))

matched = True

break

**if not matched:**

**# Assign new object ID**

global next\_object\_id

object\_data[next\_object\_id]["frames"].append(1)

object\_data[next\_object\_id]["centroids"].append(centroid)

frame\_data.append((next\_object\_id, (x\_min, y\_min, x\_max, y\_max), centroid, class\_name))

next\_object\_id += 1

**7. Draw bounding boxes, centroids, trails, and labels**

for obj\_id, box, centroid, class\_name in frame\_data:

x\_min, y\_min, x\_max, y\_max = box

cv2.rectangle(frame, (x\_min, y\_min), (x\_max, y\_max), (0, 255, 0), 2)

cv2.circle(frame, centroid, 5, (0, 0, 255), -1)

cv2.putText(frame, class\_name, (x\_min, y\_min - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2)

**# Draw trail**

for i in range(1, len(object\_data[obj\_id]["centroids"])):

cv2.line(frame, object\_data[obj\_id]["centroids"][i - 1], object\_data[obj\_id]["centroids"][i], (255, 0, 0), 2)

**output video**

out.write(frame)

**8.** **Export metrics**

metrics = {

"unique\_object\_ids": len(object\_data),

"time\_spent\_per\_object": {

obj\_id: len(data["frames"]) / fps for obj\_id, data in object\_data.items()

},

}

**9. Print metrics**

print("Metrics:")

print(f"Total Unique Objects Detected: {metrics['unique\_object\_ids']}")

for obj\_id, time\_spent in metrics["time\_spent\_per\_object"].items():

print(f"Object {obj\_id}: {time\_spent:.2f} seconds")

**10. HTML for output**

<!DOCTYPE html>

<html>

<head>

<title>Object Tracking Output</title>

</head>

<body>

<h1>Object Tracking Results</h1>

<video controls width="640" height="480">

<source src=" outputfile.mp4" type="video/mp4">

Your browser does not support the video tag.

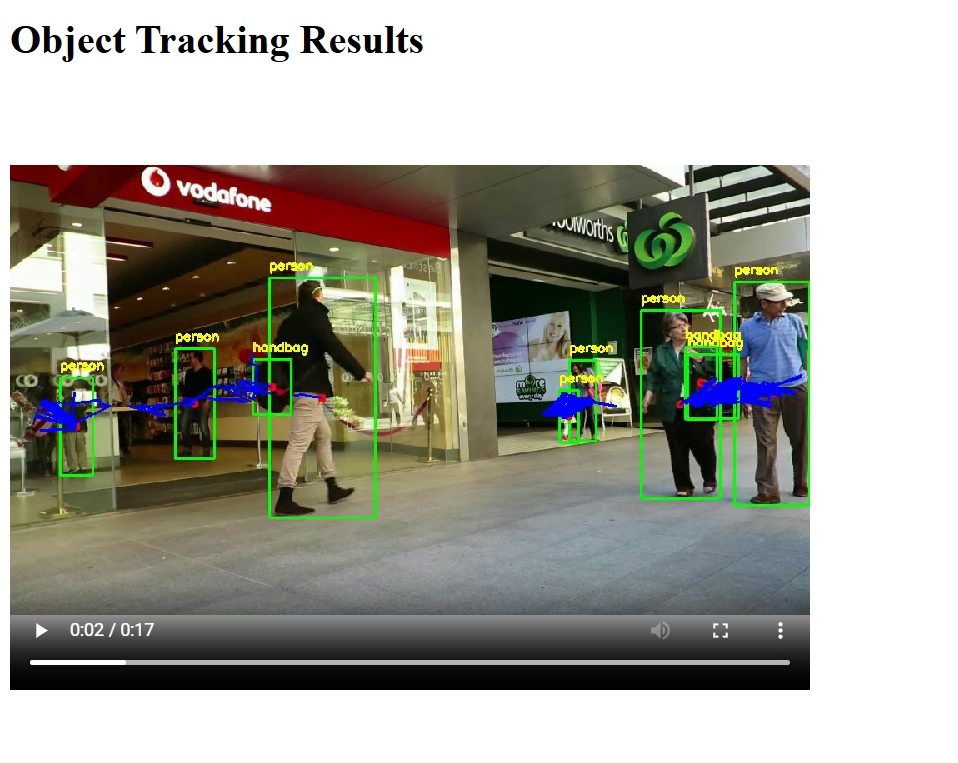
</video>

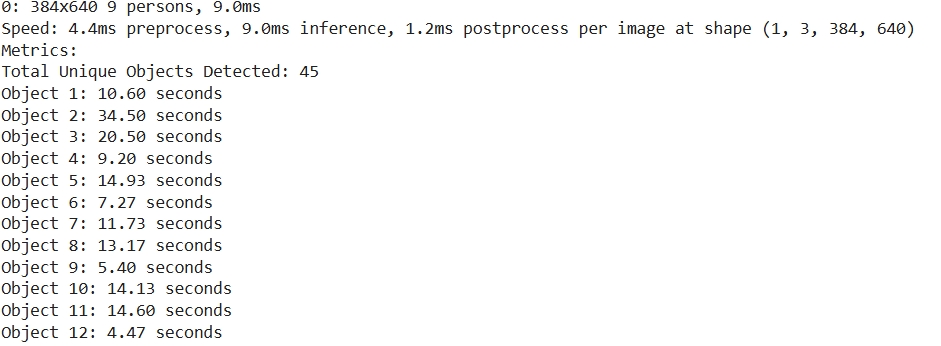
</body>

</html>

HOSTED LINK ->[**https://brilliant-gelato-2db36d.netlify.app/**](https://brilliant-gelato-2db36d.netlify.app/)

**11. Output Screenshot**





**12. Conculsion.**

Thus, The Object Tracking Project is completed with the help of YOLO Object Detection Model.