**MacV Object Tracker Task**

**Overview**

The MacV Object Tracker Task is a project focused on object detection and tracking in video feeds. It leverages state-of-the-art models and algorithms to identify objects, assign unique IDs, track their movements, and measure metrics such as the total time spent in the video frame by each object. Additionally, the project generates a comprehensive HTML report summarizing the results and includes the processed video.

**Objective**

To develop an object tracking system capable of:

* Detecting objects in video frames using YOLOv8.
* Tracking objects consistently with unique IDs using DeepSORT.
* Calculating metrics such as total time spent by objects in the video.
* Visualizing the processed results in the form of bounding boxes, centroids, trails, and timestamps.
* Generating an HTML report with detailed metrics and embedding the processed video.

**Tech Stack**

* **Programming Language:** Python
* **Frameworks and Libraries:**
  + **YOLOv8:** For real-time object detection.
  + **DeepSORT:** For multi-object tracking.
  + **OpenCV:** For video processing and visualization.
  + **HTML and CSS:** For report generation.
* **Dependencies:**
  + ultralytics (YOLOv8)
  + deep\_sort\_realtime (DeepSORT tracking)
  + opencv-python (Video processing)
  + numpy (Numerical operations)

**Implementation Details**

**Step 1: Object Detection**

* **Model:** YOLOv8 Medium (yolov8m.pt), a pre-trained object detection model.
* **Process:**
  + Detect objects in each frame.
  + Use a confidence threshold to filter low-confidence detections.
  + Extract bounding boxes, confidence scores, and class labels for detected objects.

**Step 2: Object Tracking**

* **Algorithm:** DeepSORT for tracking detected objects.
* **Process:**
  + Assign a unique ID to each detected object.
  + Track objects across frames using bounding boxes and centroids.
  + Maintain a history of object movements for trail visualization.

**Step 3: Metrics Calculation**

* Calculate the total time spent in the video for each tracked object by counting the number of frames the object is visible.
* Convert frame-based time to seconds using the video’s frame rate (FPS).
* Track the number of unique objects detected in the video.

**Step 4: Visualization**

* Annotate video frames with:
  + Bounding boxes around objects.
  + Unique object IDs.
  + Centroids of objects.
  + Trails showing the movement of objects.
  + Timestamp of how long each object has been tracked.
* Use dynamic colors for better distinction between object IDs.

**Step 5: Output Generation**

* **Processed Video:**
  + Save the annotated video as an MP4 file (output\_video.mp4).
* **HTML Report:**
  + Generate an HTML file (report.html) summarizing the results.
  + Include a table with object IDs, class labels, and total time spent in seconds.
  + Embed the processed video in the HTML report.

**File Structure**

* **main.py:** The primary script for detection, tracking, and report generation.
* **output\_video.mp4:** The processed video with annotations.
* **report.html:** HTML report summarizing the tracking results.
* **requirements.txt:** List of dependencies.
* **README.md:** Documentation for the project.

**Key Features**

* Dynamic color assignment for object IDs.
* Real-time trail visualization for tracking movements.
* Comprehensive HTML report with embedded video and tabular metrics.
* Accurate object association across frames.

**Challenges**

* Handling re-identification of objects when they leave and re-enter the frame.
* Ensuring consistent tracking in scenes with overlapping objects or occlusions.

**Results**

* **Metrics:**
  + Total unique objects detected.
  + Time spent by each object in the scene.
* **Output Files:**
  + Annotated video file (output\_video.mp4).
  + HTML report (report.html) with metrics and embedded video.