**Object Detection and Tracking System**

**Abstract**

This project implements a real-time object detection and tracking system using state-of-the-art computer vision techniques. The system utilizes YOLOv8 (You Only Look Once version 8) for efficient object detection, combined with the DeepSORT (Deep Learning-based SORT) algorithm for tracking multiple objects across video frames. The project processes video input to detect and track objects, assigning unique custom IDs to each object and visualizing their movement paths with fading trails. Additionally, the system computes and displays the total time each object remains within the video, providing valuable metrics such as the time spent by each tracked object and the total number of unique objects detected. This real-time tracking and visualization setup is designed for applications that require continuous object monitoring, such as surveillance systems and crowd analysis. The project also exports tracking metrics in JSON format, enabling further analysis and performance evaluation. Through this approach, the system offers both a practical solution for object tracking and an efficient method for evaluating object movement patterns over time.

**Concept Overview**

The core concept of this project is to create a real-time object detection and tracking system that utilizes advanced computer vision techniques to monitor and analyze moving objects in video footage. The system integrates two powerful technologies: **YOLOv8 (You Only Look Once version 8)** for fast and accurate object detection and **DeepSORT (Deep Learning-based SORT)** for robust object tracking across multiple frames.

**Key Concepts:**

1. **Object Detection with YOLOv8**:
   * YOLOv8 is a state-of-the-art deep learning model designed for real-time object detection. It detects various objects in each frame of a video by identifying bounding boxes and assigning class labels. In this project, YOLOv8 detects objects with a confidence threshold to ensure that only relevant detections are processed.
2. **Object Tracking with DeepSORT**:
   * DeepSORT is a tracking algorithm that maintains object identities across frames, ensuring that the same object is consistently tracked over time. DeepSORT assigns each detected object a unique ID and uses the object's appearance and motion to predict and update its position in subsequent frames.
3. **Custom ID Mapping**:
   * Each object detected in the video is assigned a custom ID. This ensures that even if objects temporarily disappear and reappear in the scene, they can still be uniquely identified and tracked. The IDs are managed dynamically, and the system prevents reusing IDs of objects that have permanently left the scene.
4. **Fading Trails Visualization**:
   * As objects move across the frame, their past locations are visualized using trails. These trails gradually fade, providing a historical path of the object's movement. This is useful for analyzing object movement patterns, especially in crowded or complex scenes.
5. **Time Tracking**:
   * The system calculates the total time each object remains visible in the video, allowing for real-time analysis of how long each object stays within the frame. This time information is displayed alongside each object, offering insights into object behavior over time.
6. **Metrics Export**:
   * After processing the video, the system generates a JSON file that contains valuable metrics such as the total number of unique objects tracked and the total time each object spent in the video. This data can be used for further analysis and reporting.

**Methodology**

The methodology for this object detection and tracking project involves the integration of multiple computer vision and machine learning techniques to detect, track, and analyze objects in a video stream. The steps followed in the project are outlined below:

**1. Data Collection**

* A pre-recorded video (macv-obj-tracking-video.mp4) is used as the input for object detection and tracking.
* The video is processed frame by frame to detect objects, track them, and calculate various metrics such as time spent in the frame and object trails.

**2. Preprocessing**

* **Video Capture:** The video file is read using the cv2.VideoCapture() function from OpenCV, which extracts individual frames.
* **Trail Overlay:** A blank image (trail\_overlay) is created to store the fading trails of objects that are tracked. This overlay is updated in each frame to give the appearance of moving trails behind tracked objects.

**3. Object Detection Using YOLOv8**

* The YOLOv8 object detection model (yolov8s.pt) is used to detect objects in each video frame. This model has been pre-trained on a large dataset, allowing it to detect various objects with high accuracy.
* **Confidence Threshold:** To improve detection accuracy, objects with a detection confidence below 0.75 are ignored, ensuring that only reliable objects are considered.
* **Bounding Box Calculation:** The detected objects are bounded by rectangles (bounding boxes), and each detected object is assigned a class ID and bounding box coordinates.

**4. Object Tracking Using DeepSORT**

* **DeepSORT Tracker:** The DeepSORT tracker is used to track the detected objects across video frames. It works by associating new detections with existing tracks based on appearance and motion, using algorithms like Intersection over Union (IoU) to handle matching.
* **Tracking Logic:** For each frame, the tracker receives a list of bounding boxes and updates the object positions. The tracker assigns unique identifiers (IDs) to each object and continues tracking them throughout the video.

**5. ID Assignment and Object Coloring**

* **Custom ID Mapping:** Each object detected for the first time is assigned a unique custom ID to ensure that objects are tracked consistently across frames.
* **Random Colors:** Each object is assigned a random color to visually differentiate between different tracked objects. This color is used to draw bounding boxes and display object-specific information.

**6. Object Metrics Calculation**

* **Time Tracking:** The time spent by each object in the video is recorded. This is done by calculating the time difference between consecutive frames and accumulating the time for each tracked object.
* **Displaying Time on Frame:** The total time an object has been present in the video is displayed on the frame, helping to visualize the duration of the object’s movement.
* **Trail Updates:** The centroid of each object is tracked and stored. A trail of centroids is maintained for each object, and the trail is drawn on the frame, with fading to simulate movement.

**7. Visualization and Overlay**

* **Bounding Box and Centroid Display:** For each tracked object, a bounding box is drawn around it, and the centroid is marked with a small circle. The object’s ID and the time spent are displayed alongside the bounding box.
* **Trail Rendering:** A trail is drawn behind each object, which fades as the object moves across the frame. This trail helps visualize the object's motion over time.

**8. Frame Processing and Output**

* After processing each frame, the processed frame is written to an output video file (output.mp4) using OpenCV’s VideoWriter().
* The frame is also displayed in real-time for monitoring using cv2.imshow().

**9. Performance and Metrics**

* The time taken to process each frame is calculated and displayed in milliseconds to evaluate the performance of the algorithm.
* The **FPS (frames per second)** is calculated and displayed on each frame to give an indication of the processing speed.
* **Metrics Export:** After processing the entire video, metrics such as the total number of unique objects detected and the time spent by each object are saved to a JSON file (metrics.json) for further analysis.

**10. Completion**

* The video processing is terminated when all frames are processed or if the user presses the "q" key.
* The resources (video capture and writer) are released, and all windows are closed using OpenCV’s cv2.destroyAllWindows().

**Summary of the Methodology:**

1. **Object detection** is performed using the YOLOv8 model to detect objects in the video frames.
2. **Tracking** of objects across frames is done using the DeepSORT tracker.
3. **Metrics** like time spent by each object are calculated and displayed on the video.
4. **Fading trails** are drawn to represent object movement over time.
5. The **processed video** is saved and displayed in real-time.
6. **Performance metrics** (e.g., FPS and time per frame) are tracked to assess the system’s efficiency.
7. **Results** and metrics are saved to a JSON file for analysis.

This methodology integrates deep learning, computer vision, and real-time tracking techniques to build an efficient object detection and tracking system.

**Hardware and Software Requirements**

* **Purpose:** To provide the necessary specifications for running the project.
* **Description:** List the system requirements for both hardware and software:
  + **Hardware:** CPU/GPU specifications (e.g., Intel Core i7, NVIDIA GPU for faster processing).
  + **Software:** Python version, required libraries (e.g., opencv-python, ultralytics, deep\_sort\_realtime), and dependencies (e.g., CUDA for GPU acceleration).
  + **Operating System:** Mention compatibility with Windows/Linux/MacOS.

**Challenges and Limitations**

One of the primary challenges faced during the development of this project was handling occlusions, where one object temporarily hides behind another. DeepSORT attempts to mitigate this by re-identifying objects once they reappear, but this can lead to some inaccuracies in crowded environments. Another limitation is the potential for misidentification when objects share similar appearances or move closely together. Additionally, while the system performs well on high-quality video input, its real-time performance can be affected by the resolution of the video and the computational resources available. The system is also limited by the number of object classes it can detect, depending on the model's training.

**Technologies Used**

The project leverages a combination of computer vision, machine learning, and programming frameworks to perform real-time object detection and tracking. Below is a detailed overview of the key technologies used:

**1. YOLOv8 (You Only Look Once)**

* **Purpose:** Used for object detection in each video frame.
* **Description:** YOLOv8 is an advanced version of the YOLO family of deep learning models known for its high speed and accuracy in real-time object detection. It is capable of detecting multiple objects in a single pass and provides the bounding boxes and class IDs for the detected objects.
* **Library/Framework:** ultralytics Python package, which provides pre-trained YOLO models.

**2. DeepSORT (Deep Learning-based SORT)**

* **Purpose:** Used for object tracking across video frames.
* **Description:** DeepSORT is an extension of the SORT (Simple Online and Realtime Tracking) algorithm. It incorporates deep learning to improve the accuracy of object tracking. The algorithm associates detected objects in successive frames and assigns unique track IDs to maintain the identity of each object. It also handles occlusions and re-identifications.
* **Library/Framework:** deep\_sort\_realtime Python package, which integrates DeepSORT with a real-time tracking system.

**3. OpenCV (Open Source Computer Vision Library)**

* **Purpose:** Used for video processing, drawing bounding boxes, displaying frames, and handling the video input/output.
* **Description:** OpenCV is an open-source library that provides tools for real-time computer vision tasks, such as image and video processing. It is used to capture video frames, draw bounding boxes, update object trails, and display the output video frames.
* **Key Functions Used:**
  + cv2.VideoCapture() for reading video files.
  + cv2.rectangle() and cv2.circle() for drawing bounding boxes and centroids.
  + cv2.putText() for displaying text (e.g., object IDs, FPS, and time).
  + cv2.addWeighted() for blending images (e.g., blending trail overlays).

**4. Python Programming Language**

* **Purpose:** Primary programming language used for the development of the project.
* **Description:** Python is a versatile and widely-used language in the fields of data science, computer vision, and machine learning. It provides various libraries and frameworks, such as OpenCV, PyTorch, and TensorFlow, which are essential for implementing computer vision and deep learning models.
* **Key Features Used:**
  + Handling video frames and performing object detection and tracking logic.
  + Data structure manipulation (e.g., dictionaries for object ID mapping).
  + File handling for saving metrics and output videos.

**5. NumPy (Numerical Python)**

* **Purpose:** Used for numerical and matrix operations.
* **Description:** NumPy is a fundamental package for scientific computing in Python. It is used in the project for creating arrays (e.g., for storing trail positions) and performing operations like addition, averaging, and transforming frames.
* **Key Functions Used:**
  + np.zeros\_like() for creating arrays (e.g., trail overlays).
  + np.addWeighted() for image blending.

**6. JSON (JavaScript Object Notation)**

* **Purpose:** Used for storing and exporting object tracking metrics.
* **Description:** JSON is a lightweight data interchange format that is easy to read and write. In the project, JSON is used to store metrics such as the total unique objects detected and the time spent by each object in the video. The results are saved to a metrics.json file for further analysis.
* **Key Functions Used:**
  + json.dump() for saving metrics to a JSON file.

**7. Video Writer**

* **Purpose:** Used for writing processed frames back into a video file.
* **Description:** OpenCV’s cv2.VideoWriter() is used to write the processed video frames to an output file (output.mp4). It ensures that the modified video, including bounding boxes, centroids, trails, and FPS information, is saved for later viewing or analysis.
* **Library/Framework:** OpenCV.

**8. Datetime**

* **Purpose:** Used for measuring processing time and calculating FPS.
* **Description:** The datetime module is used to calculate the processing time for each frame and measure the frames per second (FPS) to evaluate the performance of the tracking and detection algorithms.
* **Key Functions Used:**
  + datetime.now() to capture the start and end time for frame processing.

**Summary of Technologies:**

* **YOLOv8** for real-time object detection.
* **DeepSORT** for object tracking across video frames.
* **OpenCV** for video processing and visualization.
* **Python** as the primary programming language.
* **NumPy** for numerical operations and handling arrays.
* **JSON** for storing and exporting metrics.
* **Datetime** for calculating performance metrics (FPS and processing time).

**Conclusion**

In conclusion, this project demonstrates the integration of object detection and tracking techniques to provide a real-time solution for monitoring and analyzing video footage. By utilizing YOLO for detection and DeepSORT for tracking, the system is capable of accurately detecting and tracking objects, displaying detailed information about each object’s movement, and saving performance metrics. The system's ability to process video in real-time, combined with its visual and statistical outputs, makes it a useful tool for applications in security, traffic monitoring, and crowd analysis. Future work will focus on optimizing performance and extending functionality for more complex scenarios.