**Multi object tracking football drill analysis**

**Objective**

The goal of this assignment was to implement a computer vision pipeline to detect and track multiple footballs (both stationary and action) in football drill videos. The primary requirement was to dynamically overlay bounding boxes on the stationary balls and track the motion of the action ball with an ID-consistent trail, across all frames of two input videos. Emphasis was placed on maintaining consistent tracking identities and avoiding ID switches during occlusion or overlapping conditions.

**Methodology**

The solution involved building a complete video analysis pipeline consisting of object detection, multi-object tracking (MOT), motion analysis, classification of stationary vs action balls, and visualization. The implementation was done using Python, OpenCV, and Ultralytics YOLOv8.

**1. Object Detection**

To identify footballs in each frame, YOLOv8 (You Only Look Once, Version 8) was used. YOLOv8 is a real-time object detection model known for its balance between speed and accuracy. We used the yolov8l.pt model for faster inference while maintaining reasonable accuracy. It was loaded using the ultralytics Python package.

Key considerations for detection:

* Only objects with label "sports ball" or "ball" were considered.
* Detections were filtered to remove noise based on confidence threshold and bounding box size.

This detection was performed on every frame of the video and served as input to the tracking module.

**2. Tracking Algorithm: SORT (Simple Online Realtime Tracking)**

To maintain object identity across frames, I used the SORT algorithm. SORT is a classical tracking-by-detection method that uses Kalman filtering and the Hungarian algorithm for data association. It is lightweight, fast, and well-suited for real-time tasks with minimal computational overhead.

Key properties of SORT used:

* Kalman filter predicts object position in next frame.
* Hungarian algorithm assigns new detections to existing tracks based on IoU (Intersection over Union).
* Each object receives a unique ID that is maintained across frames unless occlusion or disappearance occurs.

SORT was integrated using the original open-source implementation and adapted to handle edge cases like no detections and empty frames.

**3. Handling Empty Frames & Robust Detection**

A critical part of ensuring robustness was handling frames with zero detections. If such frames are passed directly to SORT without proper formatting, it leads to index errors. Therefore, the code explicitly constructs empty NumPy arrays with the expected shape when no objects are detected. This guards against crashes and maintains tracking continuity.

**4. Action Ball vs Stationary Ball Classification**

One of the assignment’s requirements was to distinguish between the action ball (being actively moved by the player) and the stationary balls (used as markers).

To classify the balls:

* I maintained a dictionary of coordinates for each tracked ID.
* For each object ID, I calculated cumulative displacement (Euclidean distance) over recent N frames (default = 10).
* The object with the highest cumulative motion was tagged as the action ball.
* Others were considered stationary if their motion was below a specified threshold.

This motion-based heuristic works effectively since the stationary balls rarely move, and the action ball has high spatial displacement.

**5. Visualization and Overlay**

Each frame was annotated using OpenCV:

* **Stationary Balls:** Green bounding boxes with labels like ID# | Stationary Ball.
* **Action Ball:** Red bounding box with a label ID# | Action Ball.
* **Trail:** A trailing red line was rendered behind the action ball to visually indicate motion over time.

This visual representation helps reviewers interpret the ball roles clearly and verify ID consistency manually.

**6. ID Consistency & Occlusion Handling**

SORT handles short occlusions (i.e., missed detections for a few frames) using its max\_age parameter. By configuring it to 30 frames, I ensured that a track doesn’t disappear immediately if detection drops briefly.

Additionally, the min\_hits parameter was set to 3 to reduce false-positive tracks by requiring a track to be visible for at least three frames before being officially assigned an ID.

No IDs were reused or swapped unless tracking was fully lost and re-initialized. During testing, identity switches were rare and occurred only in ambiguous overlap conditions.

**Tools & Libraries Used**

* Python 3.9
* OpenCV
* Ultralytics YOLOv8
* SORT (Kalman filter + Hungarian matching)
* NumPy
* Matplotlib (optional, used for debugging)

**Project Folder Structure**

flickit\_assignment/

├── data/

│ └── raw\_videos/ # test1.mp4, test2.mp4

├── outputs/ # Annotated output videos

├── models/ # Saved YOLOv8l.pt

├── src/

│ ├── main.py # Entry script

│ ├── detection.py

│ ├── tracking.py

│ ├── utils.py

│ └── sort/ # SORT tracker source code

├── report/

│ └── report.pdf

└── requirements.txt

**Conclusion**

This project demonstrates a robust application of multi-object tracking techniques to real-world football drill videos. Using YOLOv8 for detection and SORT for tracking provided a reliable and lightweight solution with minimal ID swaps. The motion-based action ball classification worked well without needing pose estimation or deep learning-based trajectory prediction.

Further improvements can include using DeepSORT (for better ID stability) and integrating player pose to more reliably link the action ball with foot movement.