Player Detection and Tracking Project – Detailed Report

1. My Approach and Methodologies

The objective of my project was to automatically detect, track, and re-identify players in video footage, and to generate an augmented video with bounding boxes and consistent IDs.

I designed a multi-stage pipeline combining the following components:

Detection

- Initially experimented with YOLOv11 using the best.pt model provided by the evaluators.
- Wrapped YOLO inference inside a YoloDetector class with a configurable confidence threshold to control detection sensitivity.

Tracking

- Integrated DeepSORT (Deep Cosine Metric + Kalman Filter) to track detections across frames.
- Developed a Tracker class encapsulating DeepSORT configuration:
    - embedder="mobilenet" to enable fast embedding inference.
    - max_cosine_distance=0.8 to adjust re-identification matching tolerance.
    - max_age=20 and n_init=2 to manage how long tracks persist.

Visualization

- Developed a visualization script to:
    - Draw bounding boxes and IDs onto each video frame.
    - Display real-time FPS metrics.
    - Show an augmented video output live as the pipeline runs.

This approach resulted in a modular pipeline capable of performing detection, tracking, and visualization in real time.

---

2. Techniques I Tried and Their Outcomes

Over the course of this project, I experimented with several techniques:

Early Experiments

- TorchReID with OSNet embeddings and cosine similarity
    - Outcome: Successfully extracted embeddings for all detected player crops. However, embedding extraction was extremely slow on CPU and impractical for real-time deployment.
    - Ultimately replaced this approach because it required significant pre-processing and additional matching logic outside the detection loop.
- Frame deduplication using perceptual hashing
    - Outcome: Very effective at removing near-duplicate frames, reducing the dataset size from hundreds of thousands of images to a manageable subset. This significantly improved processing efficiency.
- Custom pipeline with merging and clustering embeddings
    - Outcome: Conceptually worked as intended, but integration with live detection and video augmentation was overly complex. Managing intermediate folders and results introduced delays and increased maintenance overhead.

Later and Final Technique

- YOLOv11 combined with DeepSORT
    - Outcome: Delivered fast and reliable detection and tracking with consistent IDs across frames. Enabled real-time augmentation of each frame without requiring additional pre-processing or manual merging steps.
    - This combination proved simpler, faster, and easier to maintain than earlier approaches.

---

3. Challenges I Encountered

Several challenges arose while developing and refining this system:

Hardware Constraints

- Embedding extraction using TorchReID was prohibitively slow on CPU hardware.
- Memory usage was high when processing large numbers of crops in parallel.

YOLOv11 Integration

- YOLOv11 did not have the same level of mature tooling and integrations as YOLOv5 or YOLOv8, requiring me to build a custom detection wrapper and handle pre- and post-processing manually.

MediaPipe Issues

- While attempting to integrate pose estimation, MediaPipe failed to function reliably due to incompatibility with Intel Python distributions.
- To resolve this, I precomputed the pose_vectors.npy file in a clean environment and then loaded it at runtime:
- pose_vectors = np.load("output/pose_vectors.npy")
- This approach avoided cross-environment dependency issues and ensured reproducibility.
- To document this for evaluators, I included a clear README note:

Pose vectors (pose_vectors.npy) are precomputed due to known compatibility issues with MediaPipe in Intel Python. Please refer to the pose-extraction/ folder if re-generation is required in a clean environment.

Visualization

- Early versions of the output video did not display IDs correctly due to overlay rendering bugs.
- Debugging and refining the visualization logic was necessary to ensure bounding boxes and labels appeared as intended.

False Multiples

- In detect.py and visualize_detection.py, I observed multiple overlapping detections for the same player.
- This issue will be addressed by applying Non-Maximum Suppression (NMS) to remove redundant bounding boxes.

Model Failures and Reconfiguration

- The initial model using a ResNet-18 backbone for re-identification was less robust.
- I replaced it with TorchReID for improved embedding quality and re-identification consistency.

---

4. Incomplete Components and Next Steps

Incomplete

- Long-term re-identification: At present, DeepSORT maintains IDs only within a continuous video segment. If a player leaves the frame and re-enters later, their ID may change.
- Video saving: The pipeline displays frames live but does not yet export an annotated video file.
- False positive overlaps: Duplicate detections can still occur because NMS has not yet been integrated.

How I Would Proceed with More Time and Resources

1. Long-Term Re-Identification
    - Integrate a more advanced appearance embedding pipeline (e.g., TorchReID) in combination with DeepSORT to maintain consistent IDs across occlusions and non-contiguous video segments.
2. Robust Output Video
    - Add cv2.VideoWriter support to save annotated video outputs automatically.
3. Performance Optimization
    - Move all inference to GPU to significantly improve processing speed and frame rates.
    - Optimize data loading and frame handling to further reduce latency.
4. Improved Detection Filtering
    - Apply Non-Maximum Suppression immediately after detection to remove duplicate bounding boxes.
    - Refine confidence threshold tuning to improve the balance between recall and precision.

Conclusion

This project provided extensive experience in combining object detection, tracking, and re-identification into a practical video analytics pipeline. My final system, based on YOLOv11 and DeepSORT, achieved real-time player tracking with consistent IDs across video frames and proved much more efficient and maintainable than the earlier embedding-based pipeline.

With additional time and access to GPU hardware, I would focus on completing long-term re-identification, implementing robust video output, and fine-tuning detection and filtering to achieve production-quality results.