Project Report: Smart Player Tracker

1. Introduction

This report documents the design, development, and implementation of the **Smart Player Tracker** project. The goal of this project is to perform **real-time object detection and multi-object tracking** on sports video footage, specifically focusing on tracking players accurately throughout the duration of a video.

The system is built using **YOLOv8** for object detection and **Deep SORT (Simple Online and Realtime Tracking)** for assigning consistent IDs to players across frames. The final output is a video with bounding boxes around each player and an ID label that allows re-identification and motion tracking.

---

2. Objectives

- To build a system that can detect multiple players in a video using a trained deep learning model.
- To implement multi-object tracking such that each player retains a unique ID across the video frames.
- To produce a playable output video with visual annotations indicating each player's position and ID.
- To ensure robustness in video processing, handling frame decoding, model inference, and file saving without failure.

---

3. Tools and Technologies Used

- **Programming Language**: Python 3.10+
- **Libraries**:

    - OpenCV (for video processing and annotation)
    - Ultralytics YOLOv8 (for real-time object detection)
    - Deep SORT Real Time (for object tracking and ID management)
- **IDE**: Visual Studio Code
- **Other Tools**:
    - Git and GitHub for version control
    - Git LFS (Large File Storage) for uploading large `.pt` model files
    - Google Drive (for sharing video and model files due to GitHub file size limits)

---

4. Project Overview

The pipeline is structured as follows:

1. **Input**: A sports video file in `.mp4` format.
2. **Detection**: Each frame is passed through a trained YOLOv8 model to detect objects (players).
3. **Tracking**: Detected bounding boxes are passed to Deep SORT, which assigns unique and consistent IDs.
4. **Annotation**: Bounding boxes and ID labels are drawn on the frame.
5. **Output**: All processed frames are written to a new annotated `.mp4` video file.

---

5. Key Learnings

Through the execution of this project, the following concepts and skills were gained:

- **Deep Learning Inference**: Running inference with a YOLOv8 `.pt` model and understanding the model output structure.
- **Computer Vision Fundamentals**: Working with video streams, frame decoding, annotations, and output encoding using OpenCV.
- **Object Tracking**: Understanding how object detection outputs are extended by a tracking algorithm to maintain identities over time.
- **Error Handling & Debugging**:
    - Diagnosing path issues
    - Detecting and solving `NoneType` errors
    - Verifying video encoding pipelines
- **Testing Techniques**: Creating dummy videos with random frames to isolate and test video writing issues.

---

6. Challenges Faced

a. YOLO Model Loading Errors

Initially, the model was not loading due to incorrect paths and improper file references. This was resolved by using absolute paths and validating model loading with the YOLO CLI (`yolo predict`).

b. Empty or Corrupted Output Videos

Multiple attempts resulted in output files of 0 or very small byte size (e.g., 44 bytes). The issues were caused by:

- Invalid FPS or resolution values
- Not writing frames correctly
- Incompatibility with the video codec

Resolved by:

- Verifying input video dimensions
- Using consistent codec (`mp4v`) and FPS
- Testing frame output with dummy data

c. Path and Directory Confusion

Project files were deeply nested (e.g., `src/models/videos/...`), causing path reference failures. Used `os.path.exists()` to validate all paths before runtime.

d. GitHub File Limitations

The YOLOv8 `.pt` model file exceeded GitHub's 100MB limit. Resolved by:

- Installing and configuring Git LFS
- Ensuring `.gitignore` properly excluded large files from normal Git tracking

e. Git Conflicts and Push Errors

Multiple errors during push due to:

- Merge conflicts
- Push rejections from remote updates
- LFS quota limits

These were resolved through:

- Manual conflict resolution
- Force pushing where necessary
- Moving large files to external platforms (Google Drive)

---

7. Outcomes

- A fully working object detection + tracking system for player analysis in sports videos.
- The ability to identify, annotate, and track each individual player with persistent IDs.
- A clean, playable `.mp4` video output demonstrating the effectiveness of the model.
- A functional, production-ready codebase hosted on GitHub (excluding large files).

---

8. About the Developer

Name: Dhairya Bhatia
Course: B.Tech in Computer Science & Engineering (AI & ML)

Relevant Skills:

- Python, Machine Learning, and Deep Learning
- Azure Certified and AI Fundamentals Certified
- Experience with computer vision libraries such as OpenCV and Ultralytics
- National-level hackathon finalist and state-level winner

9. Conclusion

This project evolved from a simple object detection task into a full-fledged learning experience in deep learning deployment, computer vision engineering, and project debugging. The 12+ hours of consistent problem-solving not only resulted in a successful implementation but also provided real-world insights into how AI solutions are built and stabilized.

It serves as a showcase of perseverance, debugging expertise, and the ability to deliver functional AI projects under constraints.

“12 hours. 0 breaks. 1 complete Computer Vision pipeline”