Cross-Camera Player Mapping using YOLO and Feature Matching Report

Done By

Abdul Muiz

Introduction

Title: Cross-Camera Player Mapping using YOLO and Feature Matching

Objective:
To track and match football players captured from two distinct video perspectives—broadcast and tacticam—using YOLO object detection and feature-based player re-identification techniques.

Context:
Multi-camera setups are common in professional sports for analysis and strategy. Identifying the same player across these cameras is challenging due to perspective shifts. This project addresses this by:

- Detecting entities using a custom-trained YOLOv11 model.
- Extracting visual features from each player crop.
- Matching players by comparing feature vectors using cosine similarity.

Inputs:

- Two videos: broadcast.mp4 and tacticam.mp4
- Tracked versions of these videos
- A YOLO model trained to detect players, goalkeepers, referees, and the ball

Setup & Dependencies

1. Environment Setup

- Python 3.10+
- Install dependencies:

pip install -r requirements.txt

2. Project Structure

.

├── yolo.py # YOLO object detection logic

├── extract_features.py # Feature extraction for each detection

├── match_players.py # Cosine-based matching script

├── test_visualize.py # Match visualization on frame

├── outputs/ # Tracked videos and result files

├── screenshots/ # Sample frame images

3. Input Files

- Tracked videos are placed in the outputs/ folder.
    - broadcast_tracked.mp4
    - tacticam_tracked.mp4

4. How to Run

python extract_features.py

python match_players.py

python test_visualize.py

Methodology

1. Player Detection

- Used YOLOv11 for detecting classes: player, goalkeeper, referee, ball
- Outputs include bounding boxes and class confidences

2. Feature Extraction

- Each player crop is reduced to a 3D vector via RGB average
- These vectors serve as the identity signature for each player

3. Matching Logic

- Cosine similarity is calculated between each pair of vectors (tacticam vs broadcast)
- Best match (lowest score) below a set threshold is selected

4. Visualization

- Player matches rendered on an image frame using OpenCV
- Format: player → ID 4
- Output saved as outputs/matched_players_overlay.jpg

Techniques & Outcomes

Technique	**Outcome**
RGB Mean Vector	Lightweight feature for comparison
Cosine Similarity	Effective for comparing normalized vectors
YOLOv11 Detection	Accurate player and object detection

OpenCV Visualization Clear annotated matches on image

Visual Results

- Tabular data generated with matplotlib
- Overlayed bounding boxes with labels using cv2.putText()

Performance

- ~300 frames processed per video
- Total runtime per file ~4 minutes on standard GPU

Challenges Encountered

1. Inconsistent Track IDs

- Without Deep SORT, frame-based IDs were used
- Led to occasional mismatches on similar players

2. File Format Issues

- MKV and some MP4 formats were unreadable via OpenCV until converted

3. JSON File Errors

- Detection skipping led to empty outputs
- Fixed by adding detection presence checks

4. Feature Similarity Conflicts

- RGB features can be ambiguous for similarly dressed players
- No positional context used in this implementation

Improvements & Future Work

What Remains

- Integrate Deep SORT to maintain consistent track IDs
- Improve matching via ResNet embeddings or body pose features
- Frame-by-frame sync of videos for visual proof

Future Enhancements

- Add web GUI for match visualization
- Combine spatial and color features for accuracy
- Export video overlays for entire match timelines

Conclusion

This project demonstrates a basic yet effective way to map players across two distinct camera feeds using object detection and RGB-based features. With proper tracking and deeper features, the system can scale to professional-level sports analysis.