📝 **Title:**
Cross-Camera Player Tracking and Re-Identification in Sports(Football)

👤 **Author:**
Lakshay Bhatia
Email: lakshaybhatia46@gmail.com
Phone: +91 78273 37055
GitHub: github.com/Lakshayb143
Date: June 29, 2025

📄 **Abstract:**
This project tackles the challenging task of tracking and re-identifying football players across two distinct camera views — broadcast and tacticam footage.

Leveraging a YOLO-based object detection model and a lightweight ReID backbone (`osnet_x0_25`), the system detects players, tracks them with view-specific IDs, and assigns consistent global identities through appearance embeddings and spatial mapping via homography.

A modular, low-level design architecture was implemented with strong logging, Docker containerization, and reproducibility using `uv` . While accuracy can still be improved, this solution lays a strong foundation for further research and real-time enhancements, including future integration of Vision-Language Models for context-aware matching.

🧭 Table of Contents

Introduction

This project addresses the challenge of player re-identification across multi-view football footage using a modular and scalable architecture.

It combines YOLO-based detection, TorchReID embeddings, spatial reasoning via homography, and global ID tracking. I prioritized clarity, reproducibility (via Docker and `uv`), and clean low-level design.

The system quantifies cross-view matching using standard evaluation metrics — Precision, Recall, and F1-score — ensuring transparency and measurable performance.

While the current solution doesn't achieve perfect mapping accuracy, it demonstrates a robust, extensible pipeline that assigns global IDs, maintains temporal consistency, and supports synchronized multi-camera analysis.

The architecture is designed for future enhancements such as real-time VLM-based tracking and tracklet-level reasoning.

Personal Learnings & Takeaways

Through this project, I gained deep practical experience in multi-view geometry — especially the use of homography for perspective alignment across camera feeds. I strengthened my understanding of modular low-level design (LLD) using patterns like Strategy and Facade, which made the pipeline maintainable and extensible.

I also learned how to integrate pretrained models (YOLO, TorchReID), handle noisy and imperfect detections.

Importantly, I experienced how to engineer AI systems that balance real-world constraints like accuracy, latency, and robustness, while staying clean, reproducible, and scalable.

Steps to Run

You can run the Cross-Camera-Player-Mapping project directly on your local machine using uv, a fast Python packaging and task runner.

✅ Prerequisites

- **Python 3.11+**
- **uv installed globally**
- **ffmpeg** must be installed on before running - Download

```
pip install uv
```

```
git clone https://github.com/Lakshayb143/Cross-Camera-Player-Mapping

cd Cross-Camera-Player-Mapping


uv venv

source .venv/bin/activate  # On Windows: .venv\Scripts\activate


uv pip install -r requirements.txt


uv run application.py # running the main pipeline
```

Project Structure

```
Cross-Camera-Player-Mapping/

|

├── artifacts/                    # Contains input videos and model weights
```

```
|   ├── broadcast.mp4
|   ├── tacticam.mp4
|   └── best.pt                    # YOLO pretrained model
|
├── docs/                          # documentation of the project
|
├── src/                           # Core logic and low-level processing
|   |
|   ├── components/                    # strategy implementations
|   |
|   ├── interfaces/                    # Interfaces for design patterns
|   |
|   ├── steps/                         # Pipeline steps
|
├── utils/                     # Utility functions (logging, helpers)
|
├── Research.md                    # Knowledge Base Used in This Project
├── README.md                      # Project Documentation
|
├── pyproject.toml                 # Information about project
|
├── application.py                 # Entry Point of the application
├
```

```
└── requirements.txt              # Required Python packages
```

🎥 Input Description

- **Video 1 :** Broadcast video (maybe from side angle)
- **Video 2 :** Tacticam video (top view or higher view)

🧱 System Architecture

- Frame Extraction and Synchronization (SSIM Cross-Correlation)
- Normalization and transformation
- YOLO Detection
- Tracking Module
- Feature Extraction
- Cross View Matcher Re-ID Matching
- Visualization

Components

| Module | Purpose |

|---|---|

| FrameExtractor.py | Frame extraction from videos using `ffmpegcv`

| Synchronizer.py | Synchorization between frames. |

| ViewTransformer.py | Calculates Homography Matrix and Warps views.

| PlayerTracker.py | Use ByteTrack/Torchreid for ID tracking |

| FeatureExtractor.py | Extract features (color histograms, etc.) |

| CrossViewMatcher.py | Match players across views |

| IDManager.py | Global ID Manger across views. |

Design Decisions

- Implmented various `LLD (Low Level Design)` techniques to make code simple, clean, modular and scalable.
- Integrated comprehensive `logging` across the application, capturing events to both `standard output` and `log files` for enhanced observability and debugging.
- Chose `YOLO` for detection due to pretrained model availability.
- Torchreid model - `osnet_x0_25` selected for real-time, ID-preserving tracking.
- Re-ID through `features` generated using combination of appearance embedding, player coordinates, bounding box crops and color histograms.
- Implmented Global `ID Manager` for managing and tracking id across multiple camera's.
- Used `Docker` for contanerization of the application.

Implementation Steps

✅ Step 1: Preprocessing

- Frame sampling every X frames using `ffmpegcv`
- Video synchronization
- Normalizaton and transformation using homography with `cv2.RANSAC`
- Used field image shown in figure 1 for a common ground for both images for homography and warp persective.

✅ Step 2: Detection and Tracking

- Detection and tracking of players on each frame with YOLO `.pt` model
- Used `torchreid` model (`osnet_x0_25`) for id-preserving tracking.

✅ Step 3: Feature Extraction

- Used various features are like field coordinates, appearance embedding using models, etc.
- Applied feature weight as hyper parameter.

✅ Step 4: Cross-Camera Matching

- Used feature based cost matrix and `scipy.optimize.linear_sum_assignment` for optimization.

“field.jpg” could not be found.

Figure 1: Field Picture

Challenges

Homography Inaccuracy

- **Analysis:** Initial attempts to map one camera directly to another resulted in significant warping errors. The core issue was the lack of a stable, independent reference frame.
- **Solution:** Re-architected the entire calibration process to use a **canonical 2D field map** as the "common ground." This immediately stabilized the geometry. Further refined the solution by using a large set of **24 correspondence points** with the RANSAC algorithm, making the transformation resilient to minor human error in point selection.

Cross-View Matching Ambiguity

- **Analysis:** Relying on a single feature for matching (e.g., only appearance) was fragile and failed during player occlusions or when players with similar appearances were close together.
- **Solution:** Engineered a **multi-feature system** with a weighted cost matrix. This makes the matching logic resilient by design. If the appearance feature is weak for a given pair (e.g., they are far away), a strong match in their on-field coordinates can compensate, leading to a correct overall assignment.

Evaluation and Metrics

Engineering & Performance Metrics

- **Latency / Processing Speed** -This measures the computational efficiency of the solution.

- Used `ffmpeg` rather than opencv for better and faster frame extraction.

Evaluation Metrices

1. Matching Accuracy

Out of all the players that should have been mapped, what percentage did our model map correctly.

It measures the ratio of correct predictions to the total number of players present in the ground truth.

Formula:

$$\text{Accuracy} = \left(\text{Number of Correct Matches} / \text{Total Number of Players in Ground Truth} \right)$$

2. Precision

Out of all the mappings our model made, how many were correct.

High precision means that when the model claims a match, it is very likely to be correct.

Formula:

$$\text{Precision} = (\text{Number of Correct Matches} / \text{Total Number of Matches Made by Model})$$

3. Recall

Out of all the possible correct matches that existed, how many did our model find.

Recall (or Sensitivity) measures the model's ability to find all the relevant mappings. High recall means the model is good at not missing players.

Formula:

$$\text{Recall} = \left(\text{Number of Correct Matches} / \text{Total Number of Players in Ground Truth} \right)$$

4. F1-Score

The F1-Score is the harmonic mean of Precision and Recall. It is a useful metric when you want to find an optimal blend of the two, especially if there's an imbalance between the number of positive and negative cases.

Formula:

$$\text{F1-Score} = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$$

Matching Metrices

- **True Positive (TP)**: The model correctly maps a player from the tacticam view to their corresponding identity in the broadcast view.

  Example: Right id's are (tacticam_5 -> broadcast_10) but the model predicts (tacticam_5 -> broadcast_10).

- **False Positive (FP)**: The model incorrectly maps a player to the wrong identity in the other view.

  Example: Right id's are (tacticam_5 -> broadcast_10) but the model predicts (tacticam_5 -> broadcast_12).

- **False Negative (FN)**: The model fails to produce a mapping for a player who is present in both views and should have been mapped.

  Example: Let's say, there is a mapping for tacticam_5, but the model produces no mapping for tacticam_5.

Future Work

Realtime object detection & tracking with VLM (Vision Language Model)

- I am actively researching how Vision-Language Models (like Moondream or OWL-ViT) can enhance object re-identification by embedding player context and multi-modal attributes. These will likely improve matching robustness during occlusion or low-visibility frames.
- This will enable richer contextual understanding and spatial reasoning across broadcast and tactical camera feeds.
- To guide this direction, I am referencing existing research implementations such as the Moondream Object Tracking repository.
- The goal is to bridge visual perception with semantic grounding for more accurate and interpretable tracking pipelines.

Others

- **Transition to Tracklet-Level Matching:** Shift from frame-by-frame matching to matching entire tracklets.

- **Integrate State-of-the-Art Re-ID Models:** Using the research paper "An enhanced `Swin Transformer` for soccer player re-identification".
- **Real-Time Performance Optimization:** Investigate and implement optimizations for live-streaming analysis, including GPU acceleration at all stages.

References

📚 Knowledge Base Used in This Project

- Kindly refer to docs/Research.md.
- Detailed list of research papers, tools, and documentation reviewed and applied during development.

👤 Author

Lakshay