Cross Mapper - Kinshuk Singh Bist

Approach and Methodology

Reading the Problem Statement, it was intuitive to me that the solution would have something to do with some sort of 3D mapping as there were 2 camera feeds of essentially the same football match.

Honestly speaking, while I have worked on ML Projects before, Computer vision isn't exactly my strong point. Still , I truly believe in the concept and my ability to iterate often and learn quickly.

I quickly went through the very basics of the relevant libraries, brushing up on the concepts I'd learnt a semester before along with reading up on new essential topics, before deciding on a plan.

Step 1 - Model Detection

The first step was to look at the detection by the pre-trained model. The provided model is a classification model with the following class names:

```
Model class names: {0: 'ball',
                    1: 'goalkeeper',
                    2: 'player',
                    3: 'referee'}
```

CV2 goes through the video frame by frame and annotates the detection with a bounding box, along with a confidence score.

Step 2 - Object Tracking

As we need to consistently track and map players, tracking the players was the most obvious next step. Here, the goal was to assign a temporary, unique ID to each player *within a single video* and track them from one frame to the next. This ensures that a player maintains the same ID as long as they are visible in that specific video feed.

I encountered a challenge here: **Occlusions**.

Using standard trackers like **SORT or** DeepSORT** wasn't enough, as the occlusion caused by players tackling for possession led to the tracker mislabeling the players after they emerged from the occlusion. This led to constant ID flickering and new IDs being assigned.

I researched more about different trackers and decided to use a simple yet robust implementation of **BoT-SORT with Re-ID**.

- **BoT-SORT** is an advanced object tracker. Like its predecessors, it uses a Kalman filter to predict where a player will be in the next frame. However, its key strengths are better camera motion compensation and, most importantly, its has a strong appearance-based model for Re-ID.
- **Re-ID (Re-identification)** was the real game-changer here. The idea is to create a unique embedding for each detected player based on features like their jersey color, build, etc. When a player is hidden behind another and then reappears, the tracker doesn't just guess based on position. It generates a new "fingerprint" and compares it to the ones it has stored for players who've just disappeared. By finding the closest match, it can confidently re-assign the *correct* ID. This directly solved the ID flickering problem I was facing with occlusions.

I then stored the tracking data in the following structure for both videos:

```
TrackingDataType = Dict[
    int,  # frame number
    Dict[
        int,  # class_id
        Dict[
            int,  # tracking_id
            List[Union[List[float], Tuple[float, float, float, float]]]  # bounding boxes
            ]
        ]
    ]
```

Step 3 - Calculating Homography

The main challenge here was figuring out how do players actually relate. I thought of many methods but none that were exactly applicable or consistent. Some of them were:

```
footballers postiton are generally fixed, CB LB LWF etc;
they run alsong certain lines and have certain behaviours
possesion of the ball is a big factor and players positions relative to the ball;
gestures, breaking away for a pass through;
audience support of player in a particular jersey, aurdience jersey colors, etc;
distances from each other;
```

I kept researching methods like mapping players based on jersey numbers or facial features to relate the two videos but found them unsuccessful due to motion blur, low resolution, different camera angles, and player similarity.

Finally, looking at *distances from each other* somehow calculated over two related frames, I looked at a homography explanation for panorama photos from a Columbia University professor, and everything clicked! If I could find a matrix that can transform the perspective of one video feed to match the other, the problem is essentially solved. I only need to frame match the videos and relate the points from one video to the other.

Initially, I thought of matching players based on the **center distance** of their bounding boxes after the transforming them. However, I later discovered that **IoU (Intersection over Union)** is a much better alternative.

- Using center distance is simple, but it only compares the center points of two boxes and ignores their size and shape. A large bounding box for a close-up player could have the same centroid as a small box for a distant player, leading to a false match. IoU, on the other hand, calculates the ratio of the overlapping area between two boxes to their total combined area. A high IoU score (close to 1.0) means the boxes are very similar in size, shape, and location, making it a much more reliable metric for confirming that we're looking at the same player from two different angles.

Reading upon common implementations to calculate the homography matrix, I found that I needed to find common reference points in both video feeds. I realized manually picking points like goalpost etc, was not feasible for a dynamic video. I researched for something like a 'common points finder' and stumbled upon SIFT.

- It is an algorithm used to detect and describe distinctive keypoints in an image. The incredible part is that it's "scale-invariant," meaning it can find the same keypoint (like the corner of the penalty box or a specific marking on the pitch) regardless of the camera's zoom, angle, or rotation. This was perfect for automatically finding reliable corresponding points between the two different camera views.
- Looking at notebooks and code samples, I noticed everybody using something called a FLANN matcher. Upon seeking explanation I realised that trying to compare every feature from video 1 with every feature from video 2 (a brute-force approach) would be incredibly slow. This is where came in. FLANN quickly finds the best or "nearest" match for a feature from a large dataset. It's "approximate," but the performance tradeoff is worth it. Using FLANN made the feature-matching step faster, allowing me to work on the code, that might've taken too long for my laptop.

Now, homography could be applied between the two tracking data dictionaries for the videos, and a consistent mapping could be found based on the most frequently identified pairs!

Future Scope : If I had more time

Given more time, I would focus on these areas for improvement:

- It is quickly apparent on looking at the classifications by the model that the results are somewhat unpredictable. While they are mostly consistent for players, there are random detections for goalkeepers, players, etc. I would apply a **mask based on pitch keypoints** so entities outside the field aren't marked.
- I would **retrain the detection model** on a custom dataset of player snaps from these videos to improve its accuracy and reduce misclassifications.
- The default Re-ID model used is quite simple. I would implement a more powerful model like **ResNet** to create more robust and better embeddings for the players, further improving tracking through occlusions.

I've learned a lot, thanks to this project. I would love to learn more about any approach or something I've missed.

Thank you, and I sincerely hope you consider me for this exciting role.