Soccer Game State Reconstruction

Emily Anne Wang
eaw2233@columbia.edu

Nelson Lin
nl2873@columbia.edu

Yiu Chung Yau
yy3223@columbia.edu

***Abstract*—In this project, we extend upon a method for defining and reconstructing the game state of a soccer match given its live video footage. The main objective is to identify the key aspects of each game, and highlight them on various planes for a more cohesive visual experience. To achieve this, we leverage YOLOv5 and Roboflow, to classify and track multiple objects in a given video frame, as well as simple heuristics, such as clustering and linear interpolation, to add supplementary team and position information. Furthermore, we utilize YOLOv11-pose to detect a pitch's keypoints and generate a homography matrix. Combining everything together, we are able to map each object detection from the original video onto a 2-dimensional pitch. Finally, we encapsulate the entire process within an Airflow pipeline for simple depolyment. This project helps offer a fresh perspective in enriching televised soccer matches, and has potential for numerous applications. For instance: improving the viewing experience for casual fans, as well as providing more in-depth analysis for professionals. Our model and pipeline are available in https://github.com/xynlophyl/soccertracking/.**

***Index Terms*—Big Data, Soccer Analytics, Machine Learning, Deep Learning, Computer Vision, Object Tracking, Sports, Visualization, Homography**

I. Introduction

As one of the most liked sports in the world, soccer draws a lot of attention, not just from normal fans, but also sports companies, analysts, and researchers. In recent years, there has been a lot of research around athlete-centric data, aiding analysts and researchers in understanding game states such as player positions, ball trajectories, goals, etc. Yet, there exists a lack of continuous tracking data, which hinders comprehensive analysis due to incomplete information for every timestep. In this project, we introduce a system of Soccer Game State Redconstruction (SGSR), which applies techniques such as athlete tracking and identification to allow for such analysis using raw videos of match footage. We believe that such analysis could bring invaluable insights and applications, such as improving athlete training and coaching, discovering new talents, providing medical insights, as well as increasing fan engagement [1].

Our approach combines computer vision and deep learning to solve the aforementioned limitations. This system accepts a single camera view video input, and aims to detect and track soccer players on the field, as well as the ball and referees. After that, the game state will be visualized on a minimap, and combined with the original clip. Additionally, statistics such as player's running speed will be displayed under the players. As such, a more engaging gameplay experience is offered.

This is a novel and challenging task because it is a combination of many different computer vision and classification tasks, and while these steps have been tackled as challenges in their past in isolation, the difficulty of combining them altogether for good overall performance means that this is the newest frontier for computer vision in soccer broadcasting. Since this is still a relatively new trend, techniques for reconstructing the game state are still not mature yet, and thus we face multiple challenges. For example, players often overlap with each other within a small space on the pitch, which can make distinguishing different players very difficult.

SGSR is a key innovation for sports and the sports broadcasting industry, as it allows automated, objective insights on player performances, strengths, weaknesses, and strategy from a data-driven perspective. For sports broadcasting, this is helpful for viewer engagement by visualizing tactics and making them more accessible to a general audience, and using statistics to explain the nuances of the game.

In the following sections, we will discuss related work in soccer data analytics, the data we use, our methodologies, an overview of our system's architecture, the experiments we conducted and their results, as well as some potential future improvements.

II. Related Works

A. Game State Reconstruction

In [1], Somers et al. listed the limitations of modern Multi-Object Tracking (MOT). They introduced and defined the concept of Game State Reconstruction (GSR), a novel sports analytics task [1]. A GSR system aims to 1. detect and track all players on the pitch; 2. visualize game state data on a 2D minimap [1]. Key components include tracking, re-identification, team affiliation, jersey number recognition, pitch localization, and camera calibration [1]. As the outcomes of the paper, they published 1. SoccerNet-GSR (video dataset), 2. GS-HOTA (a new metric to evaluate game state), and 3. GSR-Baseline (end-to-end pipeline for GSR) [1]. In our paper, we built a new GSR pipeline based on GSR-Baseline, as an attempt to improve its performance and address some of its limitations.

Reconstructing a soccer game's game state requires a lot of complex computer vision sub-tasks, e.g. multi-object tracking, team affiliation, jersey number recognition, pitch localization, camera calibration, etc. [1] [2]. Numerous research has been done for each of these tasks.

B. Multi-Object Tracking

ByteTrack [2] is a popular method in MOT, assigning tracklets to object detections by associating their bounding

boxes across frames. This is achieved using Kalman filters and IoU-matching, applied to an object's detection bounding box in a given frame to predict it's location in the next frame and then calculate its similarity with a corresponding bounding box. These tracklets are then matched based off this similarity score, starting with high-confidence detections, and then with ones lower confidence.

C. Team Affiliation

In [3], Istasse et al. built a Convolutional Neural Network (CNN) that can differentiate players from different teams. It separates pixels of different teams in an embedding space, and groups them into common teams [3]. As a result, we do not need to learn or recognize the features of a specific team, but rather learn how each player's pixels are mapped to the feature space [3]. Although their experiments were performed on basketball datasets, we can apply similar concepts to soccer.

III. Data Description

A. Source

The datasets used in this paper were obtained from two main sources, SoccerNet and Roboflow. Both datasets were used in training and finetuning of our models. The SoccerNet dataset comprises of 200 30-second videos of soccer match footage, as well as providing ground truth labels for each step in our modeling process to help with benchmarking and evaluating performance. The Roboflow dataset is split into two sections, one for object detection and classification, including 372 individual frames of matches and their respective labels, and another for pitch keypoint detection, comprising of 317 frames and labels. On top of the training data, an external dataset with 6 30-second clips of match footages were also used for visualizations and evaluation.

For this task, all of the match footage used are continuous shots filmed from the main camera, which is oriented above ground-level from the center-line on the pitch, and it pans left and right to keep the ball in frame, while keeping the goals/corners on the edges of the camera frame. This perspective is critical for the step of transforming the camera image from the camera plane to a 2D plane, and there are no cuts, close-ups, or moving cameras in this footage.

B. Preprocessing

The main data preprocessing step was to capture each 30-second video into 750 individual frames for the model to process using cv2's VideoCapture module.

IV. Methods

A. Object Detection and Tracking

The first step for our reconstruction pipeline was to detect each key object in a given frame, and sort them into their respective classes (i.e. player, goalkeeper, referee, ball). To perform this, we finetuned YOLOv5-detect using the Roboflow dataset. These updated model weights were then used for the final model's inference, which outputs detections in the format: (cls_id, bbox, confidence). Only detections with confidence ¿ 0.1 were kept to improve the model's accuracy.

Using Roboflow's Supervision framework, we were able to associate each player and referee across the frames of the match video. Supervision utilizes ByteTrack [2], which allowed us to assign tracklet ids to object detections. This enables more informative visualizations, especially in the 2-dimensional case, where each player is represented as a point on the pitch plane. Tracking can also be beneficial in future steps to allow easier association between detections and practical identifications such as jersey numbers and player names.

B. Team Assignment

Using the updated tracking data, we want to identify their respective teams. This is achieved using K means clustering. For each player detection in a given frame, we use its bounding box to extract the player's image, clipping the upper half. This allows us to focus only on the player's jersey. Then, we will perform clustering to segment the jersey's color, and the background, to isolate the jersey's color. After repeating such steps for all players, we perform another 2-means cluster to group similar team colors together, one cluster representing the jersey color of team 1, and one of team 2. As such, we are able to identify the team of each player. We then assign 2 hard-coded colors for consistent visualization.

For goalkeepers, however, we cannot just simply apply the same method to assign a team. This is due to the fact that goalkeepers, mostly, have different jersey colors than the rest of their team, which makes it difficult to differentiate which team the goalkeeper belongs to. Therefore, besides using K means, a heuristic is utilized to estimate the teams for each goalkeeper. The idea is simple: given a frame with players who are identified and assigned a team, we calculate the positional centroids of both teams:

$$C_t = (\frac{1}{n_t}\sum_{i}^{n_t} x_i, \frac{1}{n_t}\sum_{i}^{n_t} y_i),$$

where n_t is the number of players of a team t in a frame, C_t is the centroid (average positions) of the team.

Then we calculate the distance between the goalkeeper and the centroids:

$$d(C_{goalkeeper}, C_t) = \left|\left|C_{goalkeeper} - C_t\right|\right|_2$$

The team that has a closer centroid to the goalkeeper, we assign the goalkeeper to it.

C. Ball Interpolation

One of the key aspects of capturing the game state of a soccer match, is knowing the location of the soccer ball. Subsequently, the ball should always be in view of the capturing camera. Yet, it is not guaranteed to be detected by our model in each frame due to factors such as motion blur, player obstruction and other noise. Therefore, we included a

processing step to predict the ball's movement between frames where its detection is missing, using simple linear interpolation. This step takes the ball's last detection coordinates before the missing detection, and the first detection after, plots a straight line between the two points and segments it based off how many frames the ball was not detected for. This solution is not the most accurate, but works well enough for our project needs.

D. Pitch Localization

After obtaining the bounding boxes and coordinates for various objects in the camera images, the next step is to orient them to the 2D plane, and to do so, we need reference points for which we have ground truth distances/coordinates for. Pitch localization is the process of detecting such reference points and matching them with the keypoints we know. On a soccer pitch, these keypoints are the white lines drawn on the pitch forming the boundaries and structures, such as the center circle, left/right penalty boxes, goal lines, and side lines. These are important, standardized values in the sport, and thus by measuring the angles and distances of these structures on the camera image, we are able to estimate the camera parameters and orient ourselves from the camera perspective to a real-world bird's eye view.

The first step is keypoint detection, and this is done with Ultralytic's YOLO11 model, pretrained on the task of detecting points along the white lines on the pitch, especially corners, and then these are matched to a csv containing the known coordinates of the keypoints on our 2D pitch image. In order to obtain a homography matrix that will allow us to transform between the camera plane and the 2D plane, we need at least 3 keypoints (particularly corners, center points, or endpoints of lines) visible in every single frame. This is why the format of the data is important - it is critical that the data is filmed from the main camera, which is sufficiently zoomed out to keep more than enough keypoints in frame.

E. Camera Calibration/Perspective Transformation

Finally, having obtained the coordinates of pitch keypoints within the camera image, we can then find a matrix that will map the camera image coordinates of keypoints to the known 2D-plane coordinates, and having multiple keypoints will allow us to find this matrix with higher confidence. This allows us to conduct camera calibration, as from this step, we can find the camera parameters that led to this image. For instance, if at one point, the penalty spot of the left goal is oriented at the center of the camera image, but we know that in real life (and proportionally on our 2D pitch png), the penalty spot is exactly 12 yards from the left goal. Thus, we take the two points of the penalty spot and the goal on the image, and find a way to transform them into the distance we know is ground truth, as well as undistort the angle such as the tilt and pan of the camera.

Then, we can take that transforming matrix, called a homography matrix, and apply it to the center coordinates for the bounding boxes of each object on the pitch. Since the image keypoints were mapped correctly to the known dimensions of the pitch, the players and ball would thus be correctly placed relative to these points. This allows us to plot the objects (players, goalies, refs, ball) on our 2D minimap frame-by-frame. In our implementation, we use cv2's function to find a homography matrix by providing it with our detected keypoints, the ground truth coordinates, and then applying the homography matrix to our objects.

F. Jersey Identification

A subtask that we wanted to take on alongside the rest of the project, was jersey identification using digit and character recognition. Being able to identify players based off the names and numbers on the back of their jerseys, could aid in matching detections with the actual player, using external team and player information. This, in turn, could also be used in further analysis beyond the scope of the input video.

In this project, OpenMMLab's MMOCR was used to identify jersey numbers [4]. We chose to identify jersey numbers over player names, since the former tends to be larger and more identifiable. To perform the stage, we used images of each player, using their detected bounding box to crop the frame, and performed inference with the model.

G. Visualization

To put everything together, we generated visualizations that can be used for analysis and tooling. First, in order to visualize our tracking module, we added annotations to the original input match footage. Using the detections and tracklets from each frame, we added markers to indicate and identify each detection, their respective classes, teams, as well as their jersey/tracking number. Then, we stitched the annotated frames back into the full-length video.

Next, a minimap was also generated to view the match as a 2D representation. The transformed coordinates were mapped onto the 2D pitch, and each annotated frame was, again, stitched together to form the full-length output.

V. System Overview

A. Tech Stack

We use Python throughout the entire system. We used YOLOv5 as the baseline object detection model, and YOLOv11 as the baseline camera calibration model. We utilized Google Colab and the GPUs to fine-tune the models with Roboflow and SoccerNet data. We used OpenCV for video and image processing.

B. Software Architecture

We set up a Virtual Machine (VM) on Google Cloud Platform (GCP), and built a workflow pipeline with Airflow, running on the VM. The workflow includes the entire SGSR system. VM specs: c2-standard-4 machine, 16GB RAM, 100 GB hard-drive

On a high level, we can think of the SGSR system as a black box, users just need to input a 30-second video and pass it to the system, it will process it and output 3 videos. The SGSR

system can be further broken down into small procedures. We modularized the system into 11 components, so it can be easily maintained, and easier to process on a data pipeline platform, like Airflow.

Fig. 1. Airflow DAG

An example workflow: the user first uploads a 30-second video clip, "example_match.mp4" to the GCP bucket's input folder, and then manually triggers the Airflow workflow, and pass the file name in the config. The system will start processing the video, and eventually output 3 videos: an annotated version of the original video (with object tracking information on it), a minimap representation of the players in the original video, and a combination of the previous 2 videos.

C. DAG pipeline

1) Transfer Input File: A component for transferring the input video clip from the Google Cloud bucket to the VM's local directory. After the user has triggered the Airflow DAG and passed the file name in the config, this component uses Google Cloud CLI to copy that file from then bucket to the VM's local, so the next components can start processing.

2) Detection Tracking: A component for handling detection and tracking of the objects in the video, e.g. players, goalkeepers, referees, and the ball. It reads the input video clip and feed it to the YOLO model. The model predict detections frame by frame, and store the information into a "tracks" JSON, in the following format:

```
if goalkeeper:
  tracks["players"][frame\_num][track\_id] = {"bbox": bbox, "cls_name": "goalkeeper"}
if player:
  tracks["players"][frame_num][track_id] = {"bbox": bbox, "cls_name": "player"}
if referee:
  tracks["referees"][frame_num][track_id] = {"bbox": bbox}
if ball:
  tracks["ball"][frame_num][1] = {"bbox": bbox}
```

Fig. 2. Tracking data format

This provides all the information we need for further analysis of the objects. After done predicting, it stores the JSON into a Pickle (.pkl) file so later components can utilize it.

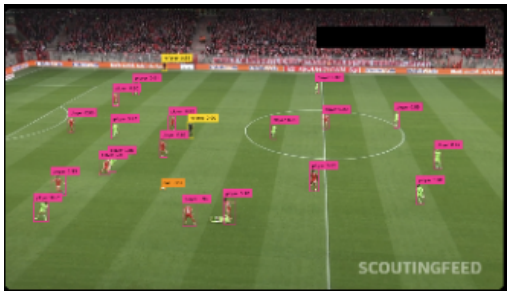

Fig. 3. Sample YOLO object detections

3) Ball Interpolation: A component for estimating untracked ball positions. It reads the tracks JSON from Detection Tracking, and uses linear interpolation and backfilling to estimate untracked ball positions, so the tracking of the ball's location will look more accurate. It also checks which player is controlling the ball, by comparing the distance between each player and the ball. If a distance is less than the threshold (we set it to 70 pixels), that player controls the ball, and we highlight that player. After done analyzing, it also saves the JSON into a Pickle file so later components can utilize it.

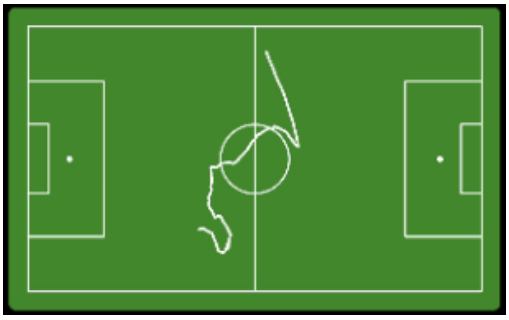

Fig. 4. Ball interpolation visualization

4) Keypoints Detection: A component for detecting key points of the pitch. It reads the input video and feed it to the camera calibration model. It predicts the positions of the key points and stores the results JSON into a Pickle file, so later components can utilize it.

Fig. 5. Sample YOLO keypoint detections

5) Perspective Transformation: A component for transforming the key points of the pitch to 2D. It reads the tracks JSON from Keypoints Detection, then takes source (with confidence values) and target vectors, filters by minimum confidence threshold, then calculates the homography transformation, to transform the players and the ball's position to 2D plane. Then it saves the JSON into a Pickle file.

6) Team Assignment: A component for determining the team color of each player, including the goalkeepers. It reads the tracks JSON from Detection Tracking/Ball Interpolation, and starts a K means clustering. It detect each players'jersey by using his bounding box from the tracks JSON, cuts it by half and feed the upper part (very likely to include the jersey) and feed it to the clustering model. This should group all players into 2 teams. We used 2 hardcoded colors for better

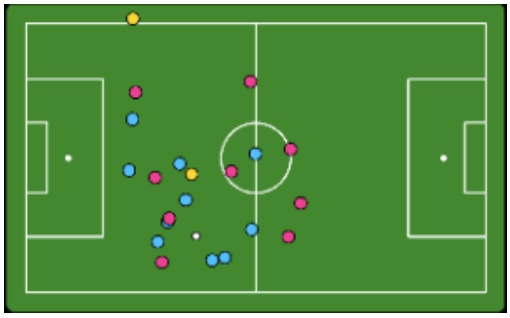

Fig. 6. Sample perspective transformation

visualizations. Team 1 players should be assigned with team 1's color, and so for team 2. For goalkeepers, it is hard to tell his team just from his jersey because it usually looks different from normal players' jersey. Instead, we utilize a heuristic, to calculate the average position of the players from both team, whichever is closer will be the goalkeeper's team. Of course, this is not always 100% accurate, we will discuss some improvements made in the Improvements and Optimizations section. After done analyzing, it saves the tracks JSON into a Pickle file.

Fig. 7. Team assignment sample

7) Merge Tracking Data: From most of the previous components, each of them reads a JSON pickle file before processing anything, and output the intermediate JSON at the end. This components merge all previous tracking JSON pickle files, and output a finalized pickle for the output components. There shall be no conflicting between the previous pickle files because each component will only add new key-value objects, but would not update anything existing objects, so they files are safe to be merged.

8) Output Annotated Video: This component reads the finalized tracking JSON from the previous component and draws notations, like tracking id, player's team color, an triangle that indicates the ball, etc. on the input clip frame by frame, and combined them to an output video.

9) Output: Minimap Video: This component reads the finalized tracking and key points JSON from the previous component and draws notations on a 2D mini-map. Each player is presented as a dot on the 2D map, colored in his assigned team color, annotated by his tracking id. Then the frames will be combined into an output video.

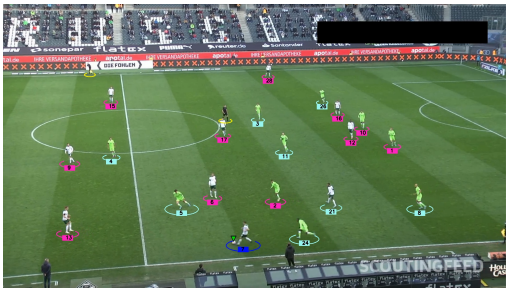

Fig. 8. Annotated match footage

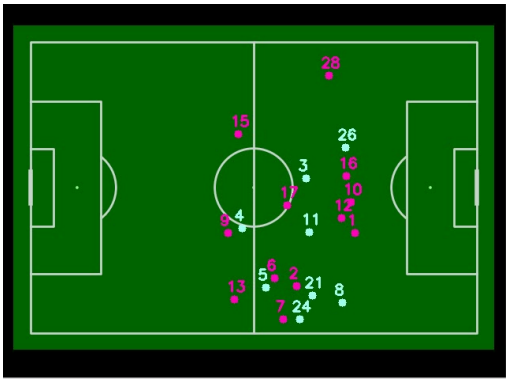

Fig. 9. Annotated minimap

10) Output: Combined Video: This components read the annotated video and mini-map video generated from the previous 2 steps, and combine them into 1 video. It uses the annotated video as the main video, and uses cv2 to resize the mini-map video, places it at the bottom.

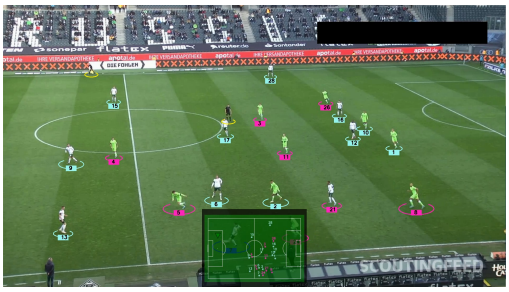

Fig. 10. Combined video

11) Transfer Output Files: Finally, after every tasks has been run, we should have 3 output videos and a couple of pickle files in the local directory. This components copy all output videos and pickle files to the Google Cloud bucket using Google Cloud CLI. The pickle files are used for further analysis and evaluation.

D. Bottlenecks

During the development and experiment process, we noticed a few bottlenecks:

1) Overall Running Time: We observed that the system takes about 25 minutes to finish from reading input to transferring outputs to the Google Cloud bucket. The main bottleneck is at the Detection Tracking component and the Keypoints Detection component. Each takes about 10-12 minutes to generate the JSON objects. As the length or the resolution of the video grows, the process will take more time to finish.

2) Object Detection Accuracy: We noticed that the detections from our model is not 100% accurate. Sometimes in some frames, a player may not be detected, that you will see no annotation under that player. Sometimes, for the same player, he may be labeled with different tracking id in different frames. We noticed this happens more frequently on goalkeepers.

3) Team Color Assignment Accuracy: We noticed that the team color assignment is also not 100% accurate. Sometimes, some players may be assigned to the wrong team. This could happen throughout the entire clip, on just a few frames. This could due to the 2 teams' color are close, or the players overlap with other players in some frames, so both team colors are detected when passing the jersey part to the K means cluster.

4) Camera Calibration: We noticed that in the 2D map output, most points are shaking and volatile, instead of having a smooth movement on the map. This is due to the homographic 2D points are not smooth enough.

5) Jersey Detection: An issue with this method was that identifications were sparse, since each image had a low resolution, due to its cropped nature, and players had varying orientations across frames, only occasionally having the back of the jersey clearly facing towards the camera. Thus, only a small portion of jersey numbers could be identified across all the detections.

E. Improvements and Optimizations

1) Overall Running Time: To stabilize the over running time, we requires the video to be in resolution 640*480, 25 FPS, 30 seconds long. We did not have enough time to build a complete backend in this project, but in the future once we have a backend, the tasks on Airflow can be ran in parallel so the workflow will take less time to finish.

2) Object Detection: Our current fine-tuning method did not see a big improvement in model accuracy. In the future we need to train the model with more high quality data.

3) Team Color Assignment: We implemented an enhanced version of K means clustering: every time after a clustering is done, we checked the size of both cluster group. We know each soccer team should have exactly 11 players, so each cluster group should have at most 11 points. So if the smaller group has at least 40% of the players on the ground, it is a valid clustering. Otherwise, redo clustering again, until we found a good one or it reaches the maximum number of trial (100). This enhanced version significantly boosted the accuracy of the team color assignment task.

4) Camera Calibration: To address the noisy transformations in homography mappings, utilizing smoothing techniques between frames can help improve robustness in our visualizations. One such group of methods is temporal smoothing, such as Kalman filtering, similar to the method described in ByteTrack [2]. Another method is to utilize a small window of neighboring frames to compute the homography instead of just the current one, which could help increase the accuracy of keypoint detections as well as variance in each transformed point between frames.

5) Jersey Detection: In order to improve detection accuracy across all frames, we utilized a minor work-around, using tracklets and majority voting to assign a constant jersey number for a particular player. This meant that for a particular player (or tracklet) and their possible jersey numbers, a weighted sum using each identification's confidence was calculated to determine the final assignment across all frames. This helped improve overall accuracy and robustness of the outputs, but there are still occasions which can not be handled, such as when players never show their numbers to the capturing camera.

Overall, due to these issues, we decided to not include jersey detections in our final implementation, using tracklet id's instead to identify players.

VI. Experiments

A. Evaluation Metric

For evaluation, we designed a custom metric to measure the difference between the ground truth coordinates for players on the 2D mapping and our predicted coordinates, using the SoccerNet data which comes with the ground truth information. This custom metric first uses the Hungarian Algorithm to match our points to the ground truth data's coordinates, which uses the Euclidean distance to find the distances between each possible pair of ground truth and predicted points, and then matching the closest pairs. Our loss metric is the sum of the Euclidean distances between each of these pairs, for all the objects in an image, divided by the number of objects. Dividing by number of objects is necessary because some frames have more objects than others. This is done efficiently using scipy.optimize.linear_sum_assignment.

The reason why we had to use this custom loss metric is because without the complete classification information for each point (jersey number, team, role), we are unable to match our points to the ground truth based on those values. Thus, we use Euclidean distance to match our points first (the matching may be incorrect if there are multiple players that are close together, then our matching algorithm might compare our predicted player location with the ground truth location of a different player instead).

B. Setup

This metric calculates the total loss (sum of Euclidean distances) for each frame of a video from SoccerNet, and can also be applied across all frames of a video to obtain a total video loss.

While working on hyperparameter tuning and smaller model improvements, using the complete video loss was unnecessary

due to the runtime, so we compared the loss within a few testing frames to determine model improvement.

The loss is the average Euclidean distance per object on the image frame. To run this loss metric, is is necessary to use a SoccerNet video, and first convert the SoccerNet coordinates (which are on a custom scale) to the image-level coordinates using their function draw_radar_pitch to normalize the two coordinate systems.

C. Metrics

Video	Frame	Loss
test/SNGS-194	000001.jpg	2.953
test/SNGS-194	000050.jpg	3.476
test/SNGS-194	000100.jpg	1.587
test/SNGS-194	000150.jpg	4.014

TABLE I
CUSTOM LOSS METRIC

VII. CONCLUSION

This paper outlines an infrastructure for reconstructing game states using live soccer match footage. The approach utilizes a multimodal pipeline, utilizing airflow, to integrate various computer vision tasks, such as object detection, tracking and keypoint detection. Using YOLOv5-detect, we were able to finetune a model capable of identifying all players, referees and balls across several frames of a 30 second video. With Roboflow's supervision toolkit, we were able to perform reidentification across each frame to allow for constant tracking of each unique object in the video. Team assignment and position estimation were then integrated using various heuristics, such as linear interpolation and k-means clustering to complete the tracking submodule. Next, YOLOv11-pose was used to identify pitch keypoints in each frame, before calculating a homography transformation. Using the homography, a perspective transformation was applied to each player's detection coordinates. Then, each transformed coordinates was applied onto the corresponding 2D pitch frame, and each frame was stitched back to together to form the final reconstructed game state visualization. The final model is versatile and robust, capable of handling many forms of match footage to generate insightful visualizations that can be utilized for further analysis.

For future work, our priority would be to improve jersey detection and incorporate this into our pipeline. Other improvements that we would make including post-perspective transformation position smoothing, which would lower the jitter that can be observed on the points because of slight location jumps from our frame-to-frame coordinate plots. Another way to smoothen our video output is by applying temporal smoothing after generating coordinates for every frame - for instance, if the distance traveled within two frames is too large, then this is likely to be due to error in either our player tracking (image-level) or perspective transformation having high loss (error on the 2D coordinate side), and lowering the distance traveled between frames of the video would be more realistic and look better, likely also lowering loss.

We would also produce visualizations such as a Voronoi plot, heatmap/arrows showing where and how far a single player traveled over the 30s clip, or speed estimators by scaling the 2D image to a real-life pitch dimension to detect what their running speed is.

REFERENCES

[1] V. Somers et al., SoccerNet Game State Reconstruction: End-to-End Athlete Tracking and Identification on a Minimap. 2024. [Online]. Available: https://arxiv.org/abs/2404.11335

[2] Y. Zhang et al., ByteTrack: Multi-Object Tracking by Associating Every Detection Box. 2022. [Online]. Available: https://arxiv.org/abs/2110.06864

[3] M. Istasse, J. Moreau, and C. De Vleeschouwer, "Associative Embedding for Team Discrimination," Jun. 2019. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*

[4] Z. Kuang et al., MMOCR: A Comprehensive Toolbox for Text Detection, Recognition and Understanding. 2021. [Online]. Available: https://arxiv.org/abs/2108.06543