ONTARIO TECH UNIVERSITY
FACULTY OF SCIENCE, COMPUTER SCIENCE

Project Group: 1 April 6, 2024

# Computer Vision Course Project: Football Player and Ball Tracking System

## by

Odunaiya Israel

**Abstract:**

This project focuses on the design and implementation of a football (soccer) analytics pipeline using object detection and tracking. Using YOLOv8, DeepSORT (via ByteTrack), and custom computer vision modules, the system can detect, track, compute, and analyze players, referees, and the ball from video footage. The final product includes advanced features such as ball possession statistics, team color assignment, ball trajectory interpolation, speed and distance estimation, and perspective transformation. The system provides a complete foundation for real-time or post-game analysis and highlights the power of modern AI-assisted sports analytics.

# Introduction

Football is the world's most popular sport, my favorite sport, with analytics playing an increasingly important role in player development, team strategy, and sometimes fan participation after games. Although professional-level analytics typically rely on expensive camera systems and sensors, computer vision provides a cost-effective alternative that can provide access to detailed game insights.

This project was created with the goal of building a comprehensive system that can process video of football matches to extract meaningful statistics and visuals. The system supports player and ball tracking, referee identification, team classification, and real-time possession tracking, all built from the ground up using Python and a set of modern CV libraries.

# Methodology

## Tools and Libraries

- **YOLOv8**: Used for real-time object detection of players, ball, referees, and goalkeepers.
- **Supervision**
  **ByteTrack**: For tracking object IDs consistently across frames.
- **OpenCV**: Core image processing and drawing library.
- **Scikit-learn**
  **KMeans**: Used for team clustering based on color.
- **NumPy**
  **Pandas**: Essential for numerical operations and interpolation.
- **Matplotlib**: For additional visualization on my end.

## Overview

1. Read frames from input video.

2. Detect objects in each frame using YOLO.
3. Track objects using ByteTrack (via supervision).
4. Separate ball, players, and referees using class IDs.
5. Assign team colors using K-Means clustering.
6. Interpolate missing ball positions.
7. Estimate player speed and distance traveled.
8. Detect possession based on proximity to the ball.
9. Track and visualize ball movement using trails.
10. Apply perspective transforms to normalize the field view.

## Development and Thought Process

The development process began with identifying a suitable match for analysis. I sourced a clip of the Euro 2024 semi-final between Spain and France from Footballia [1]. This match was chosen because of its high quality, dynamic play, high camera altitude, and clear visual conditions.

Initial experiments with YOLOv8 pretrained weights showed that the model was unable to consistently detect the ball or differentiate referees from players. This was primarily due to class imbalance and insufficient ball representation in the model's general training. To address this, I sourced a football-specific dataset with over 7,700 annotated images containing players, referees, goalkeepers, and balls from Roboflow (Football Analyst v2) [2].

I used Google Colab for training, writing the pipeline in a Jupyter notebook. YOLOv8m was the initial model of choice, which has around 10 million parameters. However, due to hardware limitations and frequent runtime disconnections in Colab, training was extremely time-consuming, which had some effects on the final product. A single epoch could take over 30 minutes, and the session often broke midway. Ultimately, I switched to YOLOv8n, a lighter model, to make training feasible. Despite this, I was only able to complete 30 epochs over the course of three days

due to these interruptions, which was a lot smaller than the 100 epochs I wanted. Still, the trained model showed significantly improved detection, particularly for the ball.

After training, I integrated the model into the pipeline. Detection was handled by YOLOv8, and tracking was implemented using ByteTrack. This allowed for consistent object ID assignment across frames. Custom logic was written to merge goalkeepers into the player class and exclude referees from tracking and possession analysis.

Team classification was achieved using KMeans clustering on player jersey colors in the first frame. Once assigned, colors were maintained throughout the rest of the video. Edge cases like referees and ambiguous colors were manually assigned to maintain accuracy.

Possession logic was refined to be more stable. Instead of assigning ball possession to the closest player in each frame, a smoothing algorithm was implemented. A player had to be nearest to the ball for three consecutive frames before being considered in possession. This greatly reduced flickering and frequent switching of ball ownership.

The rest of the pipeline followed naturally: interpolating missing ball detections, estimating player speed and movement, applying camera compensation, and transforming player positions to a bird's eye view for consistent spatial analysis.

## Learning Process and Key Takeaways

Throughout the course of this project, I encountered a wide range of technical concepts and practical challenges that contributed greatly to my understanding of computer vision systems.

- Gained a deep understanding of object detection pipelines using YOLOv8. I learned how to interpret detection outputs, modify confidence thresholds, and adapt to class imbalances by augmenting datasets.
- Developed the ability to use object tracking techniques such as DeepSORT and ByteTrack via the Supervision library, enabling me to maintain consistent object IDs across frames.

- Learned how to apply KMeans clustering (via scikit-learn) to jersey colors in the first frame of a match video to assign teams. I discovered how to sample pixel regions, apply clustering, and handle edge cases like referees using manual ID assignment.

- Discovered the use of interpolation and temporal smoothing to manage noisy detections—especially for the ball, which is often missed in fast-paced scenes. This included using linear interpolation via Pandas and smoothing possession changes over a number of frames.

- Implemented perspective transform logic to normalize player movement across the field. This involved calibrating top-down coordinates and transforming bounding boxes accordingly.

- Learned how to replace standard rectangular bounding boxes with more informative visuals. Specifically, I drew ellipses under players' feet to represent grounding and added floating number annotations for tracking IDs. For the ball, I designed a triangle marker to distinguish it visually, and later added a trail for recent positions to show motion direction.

- Discovered how to remove a specific classifier—"goalkeeper"—and reassign it to the "player" class, since goalkeepers were not treated differently for analytics.

- Used optical flow to estimate how the camera moves between consecutive frames. This was helpful for adjusting player and ball positions over time, compensating for camera shifts (panning, slight zooms, etc.).

- Understood the importance of structuring code into reusable, well-encapsulated modules. I used Object-Oriented Programming (OOP) principles to create modules such as 'Tracker', 'PlayerBallAssigner', 'TeamAssigner', 'CameraMovementEstimator', and more.

- Faced practical challenges with Google Colab: I learned how to prevent session timeouts, how to save and restore model weights using 'last.pt' and 'best.pt', and how to resume training when disconnections occurred. This was crucial in a training process that spanned days.

# Challenges and Solutions

Throughout the development of this project, several key challenges were encountered that required creative and technical solutions. Some challenges done, some not, but still provided opportunities to deepen understanding of computer vision techniques.

- **Inconsistent ball detection**: One of the major difficulties was maintaining consistent detection of the football across frames. The ball is small, moves rapidly, and is often occluded by players. To address this, a linear interpolation strategy was employed whenever ball detections were missing. The trajectory was estimated using previously known positions and velocities, and virtual positions were rendered with visual indicators like trails and translucent markers to help communicate their inferred nature. Although it helped, the implementation is far from perfect.

- **Referees mistaken as players**: During detection, referees often shared similar poses and positions with players, causing YOLO to misclassify them. To prevent this, specific referee IDs which were showing up (such as 25, 33, and 762) were manually flagged and excluded from logic involving team color assignment and possession. This is not a flexible solution, but because of my lack of model training, it was the best solution.

- **ID switching between players**: DeepSORT (ByteTrack) is powerful but not immune to ID switches, especially when players overlap or leave and re-enter the frame. This led to incorrect possession assignments. To solve this, the player only gains possession being the closest to the ball for 3 consecutive frames, thus ignoring abrupt and short-lived switches.

- **Perspective issues**: A moving camera introduces perspective distortion, making it hard to judge distances or draw consistent positional statistics. A perspective transformation (homography) was introduced to map bounding box coordinates from the video space into a top-down view of the field. This allowed normalized distance and speed calculations, and created a consistent visual baseline for further metrics.

- **Model training difficulties**: Training YOLOv8 on the football dataset from Roboflow introduced computational challenges. The original model (YOLOv8l) had over 26 million parameters and was extremely slow to train on Google

Colab. Training frequently disconnected or crashed, requiring multiple restarts and checkpoints .A single epoch could take over 30 minutes, and the session often broke midway. Despite this, I was only able to complete 30 epochs over the course of three days due to these interruptions, which was a lot smaller than the 100 epochs I wanted. Still, the trained model showed significantly improved detection, particularly for the ball. The training was done over several days and saved both the best and last model checkpoints to resume progress when needed.

# Results and Conclusion

The system successfully processes full-length football videos and outputs annotated footage with bounding boxes, player IDs, team colors, ball trails, and possession stats in real-time. It also estimates speed and total distance covered by each player.

Despite some occasional errors in ball detection or team assignment, the project still does the bare minimum for object detection and tracking. There is still work to be done outside of the assignment time-frame like heatmaps, formations and better confidence levels overall for objects.

Overall This project significantly enhanced my understanding of how AI and CV can be used outside of traditional image tasks.

Tech Report writing is an art.

# Acknowledgements

I would like to thank ”Code in a Jiffy” for guidance and support throughout this project and matrial from the course work.

To the developers of open-source libraries such as Ultralytics YOLO, Supervision, and OpenCV and to Roboflow for providing access to the Football-Analyst-v2 dataset.

Google Colab for offering accessible GPU support, despite occasional challenges, contributed a lot in aiding my learning process.

# References

[1] Footballia. Spain vs france - euro 2024. https://footballia.net/matches/spain-france-euro-2024, 2024. Used for extracting match footage.

[2] Roboflow Universe. Football-analyst-v2 dataset. https://universe.roboflow.com/football-analyst/football-analyst-v2/dataset/4, 2024. Dataset used to fine-tune YOLOv8 for football object detection.

# A YOLOv8 Training Configuration

The model was trained using the following configuration: To run, Run main.py, for convenience in VSCode

```
!yolo task=detect mode=train model=yolov8n.pt \
data=/content/Football-Analyst-v2/data.yaml \
epochs=25 imgsz=512 batch=32 device=cpu
```