Football Player Tracking System: Key Code Explanations

Ammar Ali

January 16, 2025

1 Key Code Segments and Explanations

1.1 1. YOLO Model Initialization and Video Processing

This section initializes the YOLO model for detecting players in the video and sets up video capture and output.

```
model = YOLO("yolo11n.pt") # Load YOLOv8 model
video_path = "Test_2.mp4"
cap = cv2.VideoCapture(video_path) # Open the video file

# Video output setup
output_path = "output.mp4"
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
fps = int(cap.get(cv2.CAP_PROP_FPS))
out = cv2.VideoWriter(output_path, fourcc, fps, (640, 480))
```

Explanation: - The `YOLO` class loads the trained YOLOv8 model for player detection. - `cv2.VideoCapture` opens the video file for processing. - The output video is set up with parameters such as frame rate, codec, and resolution.

—

1.2 2. Player Detection and Tracking

Detect players in each frame, track their movements, and draw bounding boxes.

```
results = model(frame) # Perform detection using YOLO
detections = np.empty((0, 5))

for result in results:
    for box in result.boxes:
        if int(box.cls) == 0: # Class 0 for players
            x1, y1, x2, y2 = map(int, box.xyxy[0][:4])
            conf = float(box.conf[0])
            if conf > 0.5:
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 4)
                current_array = np.array([[x1, y1, x2, y2, conf]])
                detections = np.vstack((detections, current_array))
```

Explanation: - Detects objects in the frame and filters out non-player detections (`class 0` represents players). - Confidence thresholds are applied to ensure only high-confidence detections are considered. - Bounding boxes are drawn around detected players.

—

1.3 3. Distance and Speed Calculation

Calculate player movement metrics, including distance and speed.

```
for result in resultsTracker:
    x1, y1, x2, y2, track_id = map(int, result)
    center = ((x1 + x2) // 2, (y1 + y2) // 2)

    if player_positions[track_id]:
        prev_center = player_positions[track_id][-1]
        distance = euclidean(prev_center, center) * pixel_to_meter
        player_distances[track_id] += distance

        current_velocity = distance / time_interval
        player_velocities[track_id].append(current_velocity)

player_positions[track_id].append(center)
```

Explanation: - For each detected player, the center of the bounding box is calculated. - The Euclidean distance between the current and previous centers gives the distance traveled. - Velocity is derived by dividing distance by the time interval between frames.

—

1.4 4. Heatmap Generation

Generate heatmaps for individual players or the entire team based on tracking data.

```
heatmap_gen = FootballHeatmap()
heatmap_gen.generate_heatmap(df_tracking, 'team_heatmap.png')

for player_id in df_tracking['player_id'].unique():
    heatmap_gen.generate_heatmap(
        df_tracking,
        f'player_{player_id}_heatmap.png',
        player_id=player_id
    )
```

Explanation: - `FootballHeatmap` generates heatmaps showing player movements. - Separate heatmaps are created for the team and individual players using tracking data.

—

1.5 5. Adjusting the Pixel-to-Meter Conversion Factor

The `pixel_to_meter` value represents the conversion factor between pixels in your video frames and real-world distances in meters. To set it accurately, you need to calibrate this based on your video setup. Here's how you can determine and adjust it:

1.5.1 Steps to Adjust `pixel_to_meter`

1. **Identify a Reference Object in the Video:** Find an object in your video with a known real-world size (e.g., a soccer field line, a player's height, or a marked distance on the ground).
2. **Measure the Object's Pixel Size in the Video:** Pause the video at a frame where the reference object is visible. Use a tool like OpenCV or an image editor (e.g., Photoshop, GIMP) to measure the length or width of the object in pixels.

3. **Calculate the Conversion Factor:** Use the formula:

$$\text{pixel\_to\_meter} = \frac{\text{Real-World Size (m)}}{\text{Pixel Size}}$$

For example:

$$\text{pixel\_to\_meter} = \frac{5}{250} = 0.02 \,\text{meters per pixel}$$

4. **Update the Value in Your Code:** Replace the placeholder `pixel_to_meter = 0.05` with your calculated value.

Tips for Accuracy: - **Camera Perspective:** If your camera is not perpendicular to the ground, the pixel-to-meter ratio will vary across the frame. For higher accuracy: - Use objects near the center of the frame for calibration. - If precise measurements are required, use camera calibration techniques to account for perspective distortion. - **Multiple References:** If possible, measure multiple objects and average the results to reduce error. - **Recalibration for Different Videos:** If you use a different video or change the camera position, repeat the calibration process, as the pixel-to-meter ratio may differ.

Example: Suppose you have a video where: - A player's average height is 1.8 meters. - In the video frame, the player's height appears to be 90 pixels. The conversion factor would be:

$$\text{pixel\_to\_meter} = \frac{1.8}{90} = 0.02 \,\text{meters per pixel}$$

Set `pixel_to_meter = 0.02` in your code.

—

1.6 6. Tuning Object Tracking Parameters

When using object tracking algorithms, it is essential to tune the following parameters for optimal performance:

1.6.1 max_age

- Use a higher value in scenarios with frequent occlusions or missed detections (e.g., when players move behind each other).
- Use a lower value for faster removal of objects (e.g., fast-moving scenes with minimal occlusions).

1.6.2 min_hits

- Use a higher value if you want to avoid tracking spurious or false-positive detections (e.g., static objects or noise).
- Use a lower value if the scene requires immediate tracking (e.g., fast-paced sports events with quick player movement).

1.6.3 iou_threshold

- Use a higher value for strict object association, especially if objects are spaced apart (e.g., players spread out on the field).
- Use a lower value in crowded scenes where bounding boxes may overlap more frequently (e.g., players close to each other).

1.6.4 Example Use Case:

For tracking players in a football game:

- `max_age = 25`: Allows the tracker to maintain player IDs despite temporary occlusions (e.g., when players overlap or move behind others).
- `min_hits = 3`: Ensures that only consistently detected players are tracked, reducing false positives.
- `iou_threshold = 0.3`: Ensures that player IDs are matched even with slight variations in bounding box positions.

—

2 Conclusion

These code segments represent the core functionalities of the system: 1. Loading the YOLO model and processing the video. 2. Detecting players and tracking their movements. 3. Calculating metrics like distance and speed. 4. Generating visualizations (heatmaps) for analysis. 5. Adjusting the pixel-to-meter conversion factor for accurate measurements. 6. Tuning tracking parameters to optimize object detection and tracking performance.

This structure allows clients to understand and adapt the system for various applications.