

Cricket Ball Detection & Tracking

Author: Akash Kumar Singh (2023AIY7582) – IIT Delhi

Date: 30/12/2025

1. Summary

This project aims to detect and track a single cricket ball in short video clips recorded from a fixed camera. For each frame, the pipeline outputs the ball centroid (if visible) and a visibility flag, and produces processed videos with centroid and trajectory overlays.

Included in the submission:

- Processed videos (*results/*) with centroid + trajectory overlays.
- Per-frame CSV annotations (*annotations/*) with *frame*, *x*, *y*, *visible*.
- Inference and utility code in *code/*.
- Reproduction script (*bash.sh*) and *requirements.txt*.

This report summarizes the approach, implementation changes, and practical limitations faced during the work.

2. Problem Definition

Detect the cricket ball centroid in every frame where it appears, mark frames where it is not visible, and render processed videos showing the centroid and trajectory. The camera is static, which simplifies the tracking problem.

3. Data & Inputs

- **Videos:** 15 short clips located in *25_nov_2025/* (mix of .mp4 and .mov).
 - No manual annotations were required for inference.
-

4. Pipeline Overview

Steps in the pipeline:

1. Run YOLO (*yolo11l*) inference per frame to obtain candidate detections for the *sports-ball* class. This model version is chosen by keeping in mind the tradeoff between accuracy and params rather than choosing version *m* or *x*.
2. Filter detections by confidence.

3. Use the *BallDetector* logic to select the most likely ball candidate for this frame.
4. If no candidate passes filters record: $x=-1$, $y=-1$, $visible=0$.
5. Save CSV rows and assemble processed output videos with overlays.

This simple pipeline is robust for short clips with a single ball and static camera.

5. BallDetector

For tracking more stably and reduce false positives:

- The BallDetector runs YOLO restricted to the sports ball class (COCO id 32) and gathers all candidates per frame.
- Each candidate is scored using a ***combination of YOLO confidence (weighted higher) and spatial proximity to the previously detected centroid path.***
 - If a previous centroid exists, the detector computes a combined score (higher weight to confidence, plus a proximity (distance) term) and selects the candidate with the best score.
 - If there is no previous centroid (start of video), the detector chooses the candidate with the highest confidence.
- The detector returns (cx , cy , $visible$) and stores the chosen centroid as the previous position for the next frame.

This change helps reject obvious false positives (e.g., gloves, shoes, advertisements) and yields smoother trajectories in practice.

6. Tracking & Visualization

- Multiple detections: choose the one nearest to prior centroid using the combined score described above.
- Maintain a short history (default 30 frames) to draw a smooth trajectory polyline with small circles marking detected centroids.
- Fallback frames are recorded as $-1, -1, 0$ when no valid detection is available.

The static camera and single-ball assumption make this centroid-based approach effective for short continuous ball motion.

Result: of 1.mp4

Frame: 4144



7. Training Attempt & Notes

I attempted to fine-tune a YOLO model using the [public dataset](#) in order to improve the performance. Code is given in `code/train.ipynb`

Summary of the attempt and true outcome:

- I started training with **yolo11s** at an image size of **1920** on Google Colab (free tier). The reason to choose these is because 1. To be able to be within compute 2. Image size is chosen this so that it is near to the test datasets image size whereas original yolo expects ~640 size which can cause the small objects like balls to be diminished and be missed by the model or less confident. 3. All these values I reached after some experimentations.
- Training was limited to **25 epochs** because of GPU time limits and resource constraints.
- From the epoch-wise logs, there were signs the model was *learning* (metrics changed across epochs), but the **overall performance remained poor**.

While the logs hinted at learning, the final model did not reach satisfactory accuracy for reliable ball detection on this dataset and test set. Because of that, I relied on a pre-trained yolo11l model for the inference pipeline, accepting its limitations.

Root causes and likely required fixes for pre-trained approach:

- Need a larger, more diverse set of high-resolution training images focused on cricket scenes.
- More training time on a GPU with careful augmentation to handle motion blur, etc.

- Hard negative mining or additional filters to reduce false positives from round advertisements or logos.

I tried to solve these limitations by attempting to train on diverse dataset but unable to do so, which I already mentioned above. I think, for fine-tuning we need around 100 epochs.

8. Outputs & Examples

Produced files (examples):

- annotations/1.csv — 1 row per-frame in1.mp4.
- results/1.mp4 — processed video showing centroid + trajectory.

Sample CSV rows:

frame,x,y,visible

0,512.3,298.1,1

1,518.7,305.4,1

2,-1,-1,0

9. Final Remarks

I enjoyed working on this assignment and it was a good mix of practical engineering and experimentation. The BallDetector made the pipeline noticeably more stable, but resource limits prevented producing a well-finetuned model. Despite that, the project is in a reproducible state and provides a base to continue improving detection quality when compute and data are available.

Thanks !