CRICKET BALL TRACKING USING YOLOv8

EdgeFleet.Ai AI/ML Assignment Submission

Submitted by:
Himanshu Kumar

Project Description:
This report presents an end-to-end system for automatic cricket ball detection and tracking from video footage. The solution leverages a fine-tuned YOLOv8 model for ball detection and a lightweight tracking pipeline to generate smooth centroid-based trajectories and per-frame annotations.

Technology Stack:

- YOLOv8 (Ultralytics)
- Python
- OpenCV
- PyTorch (CUDA enabled)

Test Environment (Used / Minimum):

- GPU: NVIDIA GeForce MX450 (2GB)
- OS: Windows 11
- Python: 3.12
- CUDA: 12.1

Repository:
https://github.com/13himanshukumar/edgefleet_ball_tracking

Date of Submission:
25-12-2025

1. Introduction

This project focuses on automatic cricket ball detection and tracking from broadcast-style video footage. The goal is to detect the ball in each frame, compute its centroid, and visualize its trajectory over time. The final system produces both a processed video with trajectory overlay and a per-frame CSV annotation file.

The solution combines a fine-tuned YOLOv8 object detection model with lightweight tracking logic to ensure robustness under motion blur, occlusions, and missed detections.

2. Problem Understanding

Cricket ball tracking is a challenging task due to:

- Small object size relative to frame resolution
- High-speed motion causing motion blur
- Frequent occlusions by players and bat
- Background clutter and camera motion

The system must reliably detect the ball frame-by-frame and maintain a smooth trajectory, even when detections are temporarily lost.

3. Dataset Preparation & Annotation

Training data was prepared from cricket match videos. Frames were extracted at regular intervals and manually annotated using Roboflow.

Annotations were created in YOLO format with a single class:

- ball

The dataset was split into training and validation sets and exported in a YOLO-compatible structure. Care was taken to annotate small, blurred balls accurately to improve generalization.

Training videos are available in *data/train/* , and extracted frames are stored in *data/train/frames_raw/* and *data/train/frames_selected/*

After Annotations from Roboflow, dataset is stored in *dataset/* in Yolo-format.

4. Model Architecture & Training

Model Choice

YOLOv8 was selected due to:

- Strong small-object detection performance
- Real-time inference capability

- Easy fine-tuning support

Training Details

• Base model: YOLOv8n

• Input size: 640 × 640

• Epochs: 50

• Batch size: 1 (limited by 2GB GPU memory)

• Optimizer: Default YOLOv8 optimizer

• Framework: Ultralytics YOLO

Due to GPU memory constraints (2GB), training was performed with a reduced input resolution and batch size.
Training was performed on an NVIDIA GeForce MX450 GPU (2GB). The final trained model is saved as yolov8_ball.pt, available in *models/yolov8_ball.pt*

5. Tracking & Post-Processing Logic

Detection-only output often results in flickering and discontinuous trajectories. To address this, a lightweight tracking layer was added on top of YOLO detections.

Tracking steps:

1. YOLO detects the ball in each frame
2. Highest-confidence detection is selected
3. Ball centroid is computed from bounding box
4. A fixed-length deque stores recent centroids
5. Trajectory is drawn by connecting valid points

If the ball is temporarily missed, the tracker maintains continuity using recent motion history, reducing abrupt trajectory breaks.
When multiple detections are present in a frame, the detection with the highest confidence score is selected as the true ball candidate.

6. Inference Pipeline

The inference pipeline performs the following:

- Reads the input video frame-by-frame
- Runs YOLO inference on each frame
- Extracts centroid coordinates
- Updates trajectory buffer
- Draws centroid and trajectory overlay

- Writes per-frame CSV annotations

Outputs:

- Annotated video (.mp4, saved in *results/*)
- CSV file with columns: frame, x, y, visible (saved in *annotations/*)

7. Performance Improvements & Debugging

Several issues were encountered during development:

Issue 1: No detections on test videos

- Cause: High confidence threshold
- Fix: Reduced confidence threshold to 0.01

Issue 2: Flickering trajectory

- Cause: Frame-by-frame detection noise
- Fix: Trajectory smoothing using deque-based buffering

Issue 3: GPU not utilized

- Cause: CPU-only PyTorch installation
- Fix: Installed CUDA-enabled PyTorch (cu121)

These fixes significantly improved detection stability and runtime performance.

8. Assumptions & Limitations

Assumptions

• The primary cricket ball is the object of interest in the scene

• The ball is approximately circular and visually distinguishable from the background

• Camera motion generally follows the ball trajectory

• Lighting and video quality are broadly similar to the training data

• Among multiple ball-like detections, the highest-confidence detection corresponds to the actual ball

Limitations

• Other round or ball-like objects (e.g., logos, player equipment, highlights) may produce false positive detections

• Multiple detections may appear in a single frame, requiring heuristic selection

• Extreme occlusions or fast motion blur can still lead to missed detections

• Very small, partially visible, or out-of-focus balls may not be detected reliably

• The model is trained on a limited dataset and may not generalize to all camera angles or match conditions

9. Qualitative Results

The final system successfully detects and tracks the cricket ball across various test videos (available in *data/test/*). The trajectory overlay clearly visualizes ball motion, including bounces and aerial movement.

Sample annotated frames and videos are provided in:

- *examples/*
- *results/*

These demonstrate accurate centroid placement and smooth trajectory formation.

10. Conclusion

This project demonstrates an end-to-end cricket ball tracking pipeline using deep learning and lightweight tracking logic. By combining YOLOv8 detection with post-processing and trajectory smoothing, the system achieves robust performance under challenging conditions.

Future improvements could include multi-ball handling, advanced Kalman filtering, and training with a larger, more diverse dataset.