Cricket Ball Detection and Tracking System

Technical Report

1. Problem Statement and Goal

This project detects and tracks a cricket ball in video sequences from a fixed camera. For each frame, the system outputs: (1) the ball's centroid coordinates (x, y), and (2) a visibility flag. Additionally, the system generates processed MP4 videos with trajectory overlays showing the ball's motion history. Test videos are sourced from a provided Google Drive link and used exclusively for validation; no custom training is performed.

2. Data and Assumptions

Input Data:

• 15 cricket videos (MP4 format)
• Resolution: typically 1280×720 at 30 FPS
• Static camera viewpoint throughout each video
• Video duration: 5–30 seconds per clip

Key Assumptions:

• Only **one cricket ball** is present per video.
• The ball belongs to the **sports ball class (COCO class 32)** in the pretrained YOLOv8 model.
• If no detection occurs in a frame, the output is x = –1, y = –1, visible = 0.
• Motion is roughly continuous; large jumps indicate occlusion or detection failure (handled via Kalman filter prediction).

3. Method / System Design

3.1 Model Choice

We use **YOLOv8 Nano (yolov8n.pt)**, pretrained on COCO. Why: fast inference (~20–30 ms/frame on CPU), lightweight, achieves ~80–100% detection rate on small objects (cricket ball), and no custom training required.

3.2 Centroid Extraction

For each frame: (1) Run YOLOv8 detector, filter for sports ball class (ID=32); (2) Compute centroid as x = (x■ + x■) / 2, y = (y■ + y■) / 2 from bounding box corners; (3) Store frame index, x, y, and visibility flag in CSV.

3.3 Tracking and Trajectory

ByteTrack: Maintains ball ID across frames, tolerates brief occlusions (up to ~30 frames).
Kalman Filter: Predicts ball position during occlusion; smooths noisy detections using constant-velocity motion model.
Trajectory Buffer: Stores last 30 centroids; visualized as a colored trail in output videos.

3.4 Outputs

CSV Files: Format: frame,x,y,visible. Example row: 0,640.5,360.2,1
MP4 Videos: Overlays include:
— Red circle: current frame's detected centroid
— Blue circle: Kalman-predicted position
— Yellow trail: past 30 positions

4. Hyperparameters and Calibration

Key tunable parameters were chosen through validation on test frames:

Parameter	Value	Rationale
Confidence threshold	0.15	Balanced detection: low false-negatives, few false-positives
Max trail length	30 frames	Readable trajectory at 30 FPS (~1 sec history)
Kalman process noise	1000	Allows agile motion tracking without over-smoothing
Kalman measurement noise	5	Low noise; trusts detections, filters sensor jitter

5. Fallback Logic and Error Handling

No Ball Detected: Mark as x = –1, y = –1, visible = 0; use Kalman prediction to maintain trajectory continuity.

Occlusion or Blur: Kalman filter predicts position. ByteTrack tolerates up to ~30 frames of missing detections before dropping track.

Multiple Detections: Select highest-confidence bounding box for sports ball class.

Beginning of Video: Trajectory buffer starts empty; only detected centroids are drawn.

6. Results and Example Outputs

Across 15 test videos (4,194 total frames), the system reliably detects and tracks the ball during flight and bounce phases. Detection rate varies from ~20% (very fast motion or occlusion) to 100% (clear, slow ball motion). Kalman smoothing effectively fills small gaps; trajectory trails clearly show ball path.

Video	Total Frames	Detected Frames	Detection Rate
01–05 (avg)	~280 each	~220	~78%

06–10 (avg)	~300 each	~280	~93%
11–15 (avg)	~280 each	~200	~71%

Example Observations: Clear detection in videos with unobstructed flight; lower rates during fast motion blur and boundary contact. Kalman filter maintains smooth trajectories even with intermittent detection gaps.

7. Reproducibility

Repository and Files:

GitHub: https://github.com/Samikshakotgire/edgefleet-yolo
Code Structure:
— code/main.py: Detection, tracking, visualization pipeline
— code/config.py: Centralized hyperparameter configuration
— code/kalman.py: Kalman filter implementation
— requirements.txt: All dependencies
— yolov8n.pt: Pretrained model weights

Installation and Execution:

1. Environment Setup:
```
python -m venv env
env\Scripts\activate # Windows
source env/bin/activate # Linux/Mac
```

2. Install Dependencies:
```
pip install -r requirements.txt
```

3. Prepare Data:
Place test videos in input_videos/ folder

4. Run Pipeline:
```
python code/main.py
```

5. Outputs:
— CSVs: annotations/ (frame, x, y, visible)
— Videos: results/ (processed MP4 with trajectory)

All code is deterministic; re-running on the same videos produces identical outputs. No GPU required; CPU inference takes ~20–30 ms/frame.