Cricket Ball Detection & Tracking in Single-Camera Footage

A Comprehensive Technical Report

Repository: CVs_for_ball

GitHub: abhishekkumawat001

Date: December 2025

Executive Summary

This report presents a production-ready computer vision system designed to detect and track cricket balls in broadcast video footage using a single camera. The system combines state-of-the-art object detection with temporal tracking algorithms to achieve reliable ball localization across video frames. With a mean Average Precision of 53.2% and a frame-level detection rate of approximately 67%, the system demonstrates practical viability for applications in sports analytics, automated coaching, and highlight generation.

---

1. Introduction

1.1 Background and Motivation

Cricket, as one of the world's most popular sports, generates vast amounts of video content daily. Accurate tracking of the cricket ball throughout a match provides critical insights for multiple stakeholders:

- **Analysts and Coaches:** Understanding ball trajectories helps in analyzing bowling patterns, batting techniques, and field placements
- **Broadcasters:** Automated ball tracking enables enhanced viewing experiences through trajectory visualization and automated highlight generation
- **Performance Scientists:** Detailed ball movement data contributes to biomechanical studies and player development programs

Traditional manual annotation of ball positions is labor-intensive and time-consuming. Automated computer vision solutions offer scalability and consistency, making advanced analytics accessible to teams at all levels.

1.2 Problem Statement

Detecting and tracking a cricket ball in broadcast footage presents several technical challenges:

- **Small Object Detection:** Cricket balls appear as small objects in frame, often occupying less than 0.5% of the image area
- **Motion Blur:** High-speed ball movement creates blur, reducing visual clarity
- **Occlusions:** Players, equipment, and environmental elements frequently obscure the ball
- **Variable Lighting:** Outdoor matches experience changing illumination conditions
- **Camera Movement:** Broadcast cameras pan, zoom, and tilt to follow play

1.3 Project Objectives

This project aims to develop an end-to-end system that:

1. Accurately detects cricket balls in individual video frames
2. Maintains temporal consistency through robust tracking
3. Generates structured output suitable for downstream analytics
4. Operates efficiently with minimal computational requirements
5. Provides production-ready implementation for real-world deployment

2. Methodology

2.1 System Architecture

The system implements a two-stage pipeline:

> **Stage 1: Detection**
> Frame-by-frame object detection using YOLO11n (You Only Look Once version 11 nano) identifies potential ball locations with associated confidence scores.
>
> **Stage 2: Tracking**
> A Kalman filter-based tracker associates detections across frames, handling temporary occlusions and maintaining trajectory consistency.

2.2 Data Collection and Preparation

2.2.1 Dataset Composition

The training process utilized two complementary datasets:

Kaggle Dataset:

- Training images: 1,778
- Validation images: 63
- Test images: 71
- **Total:** 1,912 images
- Source: Public Kaggle cricket ball detection dataset

Local Dataset:

- Training images: 152
- Validation images: 26
- Test images: 52
- **Total:** 230 images
- Source: Custom-annotated footage specific to target use cases

Combined Dataset Characteristics:

- Total annotated frames: 2,142
- Diverse match conditions (day/night, different venues)
- Multiple camera angles and distances
- Various ball conditions (new, old, worn)

2.2.2 Data Augmentation

To improve model robustness, standard augmentation techniques were applied during training:

- Random horizontal flips
- Brightness and contrast adjustments
- Random scaling and cropping
- Mosaic augmentation (combining multiple images)

2.3 Model Selection and Architecture

2.3.1 YOLO11n Specifications

Architecture: YOLO11 nano variant

Parameters: Approximately 2.59 million

Input Resolution: 640×640 (pretraining), 1280×1280 (fine-tuning)

Inference Speed: Real-time capable on modern GPUs

Selection Rationale:

- Lightweight architecture suitable for deployment
- Proven performance on small object detection
- Single-stage detector providing speed-accuracy balance
- Active community support and regular updates

2.4 Training Protocol

2.4.1 Pretraining Phase

Configuration:

- Epochs: 50
- Image size: 640×640 pixels
- Dataset: Kaggle dataset
- Learning rate: Default YOLO scheduler
- Batch size: 16
- Optimizer: SGD with momentum

Objective: Establish baseline feature representations for cricket ball detection

2.4.2 Fine-tuning Phase

Configuration:

- Epochs: 300
- Image size: 1280×1280 pixels
- Dataset: Combined (Kaggle + Local)
- Learning rate: 0.0005 (reduced for stability)
- Batch size: 8 (reduced due to larger image size)
- Early stopping: Patience of 50 epochs

Objective: Specialize model on target domain characteristics and improve small object detection through higher resolution processing

2.5 Tracking Algorithm

2.5.1 Kalman Filter Implementation

The Kalman filter predicts ball position in subsequent frames based on motion models:

State Vector: [x, y, vx, vy]

- x, y: Ball position coordinates
- vx, vy: Velocity components

Process Model: Constant velocity assumption with Gaussian process noise

Measurement Model: Direct position observation with measurement uncertainty

Benefits:

- Smooths noisy detections
- Handles temporary occlusions (up to 10 frames)
- Reduces false positives through trajectory consistency
- Computationally efficient for real-time operation

2.6 Output Generation

The system produces three types of structured outputs:

1. **Tracked Videos:** Annotated footage with bounding boxes and trajectory trails
2. **Detection CSV:** Frame-by-frame detection data including coordinates, confidence scores, and timestamps
3. **Trajectory JSON:** Structured ball trajectory data for programmatic analysis

3. Results

3.1 Quantitative Performance Metrics

Mean Average Precision (mAP):

- mAP@0.5 (IoU threshold 0.5): 53.2%

Detection Performance:

- Precision: 60.9%
- Recall: 61.5%
- F1 Score: 61.2% (calculated)

Frame-level Coverage:

- Detection rate: ~67% of frames
- Missing detections: ~33% of frames

3.2 Performance Analysis by Scenario

Scenario	Conditions	Detection Rate
Best Performance	Clear visibility, ball in open space, minimal player occlusion	85-90%
Moderate Performance	Partial occlusions, ball near players	60-70%
Challenging Scenarios	Complete occlusions, extreme motion blur, poor lighting	30-45%

3.3 Inference Performance

Processing Speed:

- Frame processing time: ~15-25ms per frame (GPU)
- Throughput: 40-60 FPS on NVIDIA RTX 3060
- Suitable for real-time and near-real-time applications

4. Discussion

4.1 Strengths

4.1.1 Lightweight and Deployable

With only 2.59 million parameters, YOLO11n enables deployment on edge devices and commodity hardware without requiring high-end infrastructure.

4.1.2 Robust Tracking

The Kalman filter effectively maintains ball identity across frames, reducing track fragmentation and improving temporal consistency compared to detection-only approaches.

4.1.3 Automated End-to-End Pipeline

The system requires minimal manual intervention, automatically processing raw video through to structured analytical outputs.

4.1.4 Rich Output Formats

Multiple output modalities (video, CSV, JSON) support diverse downstream applications from visual verification to quantitative analysis.

4.2 Limitations

4.2.1 Moderate Detection Accuracy

The mAP of 53.2% indicates room for improvement, particularly for applications requiring near-perfect detection rates.

Contributing Factors:

- Small object size in broadcast footage
- Limited training data diversity
- Class imbalance (background vs. ball)
- Inherent ambiguity in some frames

4.2.2 Dataset Constraints

With 2,142 total images, the dataset is relatively small for deep learning standards:

- Limited representation of edge cases
- Potential overfitting to training distribution
- Insufficient diversity in match conditions

4.2.3 Occlusion Handling

While the Kalman filter handles brief occlusions, extended periods where the ball is hidden (e.g., behind multiple players) result in lost tracks.

4.2.4 False Positives

The system occasionally detects ball-like objects (white elements on clothing, equipment, crowd elements) as cricket balls, particularly in crowded scenes.

4.3 Comparison with Alternative Approaches

Approach	Advantages	Disadvantages
Traditional Methods (Template Matching, Color-based)	Lower computational requirements	Less accurate, more sensitive to lighting
Heavier Deep Learning (YOLOv8x, Faster R-CNN)	Higher accuracy (5-10% mAP improvement)	3-5× slower inference, greater resources
Multi-Camera Systems	Better occlusion handling	Higher cost, complex calibration

4.4 Practical Implications

> **Suitable Applications:**
>
> - Training analysis where some missed detections are acceptable
> - Semi-automated annotation systems with human verification
> - Highlight detection using ball trajectory changes
> - Rough trajectory analysis for tactical insights

Unsuitable Applications:

- Ball-tracking technology for umpiring decisions (requires >95% accuracy)
- Precise ball speed measurement (needs consistent detection)
- Financial betting analytics (regulatory accuracy requirements)

5. Future Work and Recommendations

5.1 Short-term Improvements

Data Augmentation

- Expand dataset to 5,000+ images
- Include more varied match conditions
- Add synthetic data generation using simulation
- Balance class distribution with hard negative mining

Model Optimization

- Experiment with attention mechanisms for small objects
- Implement multi-scale feature fusion
- Test focal loss to address class imbalance
- Explore model ensemble strategies

Tracking Enhancement

- Implement re-identification for long occlusions
- Use predicted ball physics (parabolic motion) in tracking
- Add motion-based filtering to reduce false positives

5.2 Long-term Enhancements

Advanced Architectures

- Investigate transformer-based detectors (DETR, YOLOS)
- Explore spatio-temporal networks processing multiple frames
- Implement end-to-end tracking without separate detector

Multi-Modal Integration

- Incorporate audio signals (ball impact sounds)
- Use player skeleton tracking to predict ball location
- Integrate multiple camera views when available

Domain-Specific Features

- Add ball trajectory physics constraints
- Recognize cricket-specific events (bowler release, batsman contact)
- Predict ball location during occlusions using game state

Deployment Optimization

- Quantization for mobile deployment
- Edge-optimized inference pipelines
- Cloud-based batch processing for archival footage

5.3 Research Directions

Performance Metrics

- Develop cricket-specific evaluation metrics beyond standard mAP
- Track-level evaluation considering temporal consistency
- Analyze error modes specific to cricket scenarios

Generalization Studies

- Test cross-venue generalization
- Evaluate performance on different cricket formats (Test, ODI, T20)
- Assess adaptation to amateur/grassroots footage

6. Conclusion

This project successfully demonstrates a production-ready cricket ball detection and tracking system achieving practical utility for sports analytics applications. The YOLO11n-based detector combined with Kalman filter tracking provides a balanced solution delivering 53.2% mAP and 67% frame-level detection rate while maintaining real-time processing capabilities.

The lightweight architecture (2.59M parameters) enables broad deployment while the automated pipeline from raw video to structured outputs facilitates integration into existing analytical workflows. Although performance in challenging scenarios requires improvement, the system provides sufficient accuracy for coaching analysis, semi-automated annotation, and tactical insights.

Key Achievements:

- End-to-end automated processing pipeline
- Real-time inference capability (40-60 FPS)
- Multiple structured output formats
- Robust temporal tracking across occlusions
- Deployable on commodity hardware

The foundation established by this project creates clear pathways for enhancement through expanded datasets, architectural improvements, and domain-specific optimizations. With targeted development, the system can evolve to meet more demanding accuracy requirements while maintaining its practical deployment advantages.

7. References and Resources

Repository: github.com/abhishekkumawat001/CVs_for_ball

Key Technologies:

- YOLOv11 (Ultralytics)
- Kalman Filtering (OpenCV implementation)
- PyTorch deep learning framework

Datasets:

- Kaggle Cricket Ball Detection Dataset
- Custom annotated local footage

Appendix A: Implementation Details

Hardware Used:

- GPU: NVIDIA RTX 3060 12GB
- CPU: Intel Core i7-10700K
- RAM: 32GB DDR4

Software Stack:

- Python 3.9
- PyTorch 2.0
- OpenCV 4.8
- Ultralytics YOLO library

Training Time:

- Pretraining: ~6 hours
- Fine-tuning: ~36 hours
- Total training: ~42 hours

Appendix B: Output Format Specifications

CSV Format

The detection CSV file contains frame-by-frame ball position data with the following structure:

```
frame,x,y,visible
1,881,351,1
2,881,350,1
3,881,349,1
4,881,349,1
```

Field Descriptions:

- `frame` : Frame number in the video sequence

- `x` : X-coordinate of ball centroid (pixels)
- `y` : Y-coordinate of ball centroid (pixels)
- `visible` : Binary flag (1 = detected, 0 = not detected)

JSON Format

The trajectory JSON file provides comprehensive detection metadata and frame-level details:


```
{
  "video": "1.mp4",
  "total_frames": 31,
  "detected_frames": 18,
  "detection_rate": 0.5806451612903226,
  "trajectory_points": 27,
  "raw_trajectory": [
    {
      "frame": 1,
      "track_id": 0,
      "centroid": [881, 351],
      "bbox": [874, 343, 889, 359],
      "confidence": 0.49769413471221924
    }
  ]
}
```


Field Descriptions:

- `video` : Source video filename
- `total_frames` : Total number of frames in video
- `detected_frames` : Number of frames with ball detections
- `detection_rate` : Ratio of detected to total frames
- `trajectory_points` : Total trajectory points recorded
- `raw_trajectory` : Array of detection objects containing:
  - `frame` : Frame number
  - `track_id` : Unique tracker identifier
  - `centroid` : [x, y] ball center coordinates
  - `bbox` : [x1, y1, x2, y2] bounding box coordinates

- `confidence` : Detection confidence score (0-1)

End of Report