Cricket Ball Detection and Tracking in Cricket Videos

1. Problem Statement

The goal is to build a complete computer-vision system that detects and tracks a cricket ball in videos recorded from a single, fixed camera[1].

The system must:

- Detect the cricket ball centroid in every frame where it is visible.
- Output a per-frame annotation file (frame, x, y, visible).
- Generate a processed video with the ball centroid and trajectory overlaid.
- Provide fully reproducible code for inference, tracking, and evaluation.

2. Dataset and Constraints

Input characteristics:

- Single static camera, night conditions with a **white ball**.
- Testing dataset: provided; allowed **only for testing**, not for training.
- Ball is small, fast, and often blurred.
- Ball color is close to other white objects (non-striker's gloves, bats, boundary lines).
- Lower half of the frame contains umpires, batters, and pitch markings that cause false positives.

Implementation approach:

Due to time constraints and the prohibition on training with test data, a pure classical computer-vision pipeline was developed instead of training a deep neural network detector[2].

3. Methodology

3.1 Overall Pipeline

The pipeline consists of five main stages:

1. Frame preprocessing and HSV conversion.
2. Color-based masking for the white ball.
3. Contour-based centroid detection with shape filtering.
4. Geometric region filtering (central pitch region only).
5. Temporal filtering (jump filter) and trajectory accumulation.
6. Optional trajectory interpolation to extend the path from bowler's hand to stumps-region detections.

This is implemented in `utils.py`, `detect_ball.py`, `track_and_annotate.py`, and `interpolate_trajectory.py` under `code/` as required.

3.2 Preprocessing

Frame resizing and blurring:

- Each frame is resized to a fixed width (960 px) to standardise scaling and reduce computation.
- A Gaussian blur (5×5 kernel) is applied to suppress sensor noise and texture artifacts.

Color space conversion:

- Frames are converted from BGR to HSV color space, because HSV separates chromaticity and brightness, making it more robust for color thresholding under varying illumination[3][4].

3.3 Ball Detection (Color + Contour)

Color masking:

For a white ball under floodlights, the ball appears as a low-saturation, high-value region in HSV:

- $H \in [0, 180]$
- $S \in [0, 110]$
- $V \in [180, 255]$

A binary mask is constructed using `cv2.inRange`, followed by morphological opening (to remove small noise) and dilation (to slightly grow the ball region)[3][4].

Contour and centroid:

- Connected components are extracted with `cv2.findContours`.
- For each contour, **area** and **perimeter** are computed.
- **Circularity** is measured as: *circularity* = $4\pi \times \text{area} / \text{perimeter}^2$
- A relaxed filter allows small and blurred balls:
  - min_area ≈ 5
  - max_area ≈ 12000
  - min_circularity ≈ 0.2

Among remaining contours, the one with the highest **area × circularity** score is chosen as the ball candidate. Its centroid is computed via image moments and reported as (x, y) with visible = 1[2][3].

If no contour passes the filter, the frame is marked as x = y = -1, visible = 0.

3.4 Geometric Region Filtering

Because multiple white objects exist (non-striker's gloves, pads, boundary lines), additional camera-specific geometric priors are applied:

Vertical constraint:

- Only detections in the **upper 75% of frame height** are allowed, ignoring the lower half where feet, gloves, and umpires are located.

Horizontal constraint:

- Horizontally, detections are constrained to a central lane: $x \in [0.30W, 0.70W]$, where W is frame width.
- This roughly corresponds to the pitch region where the ball travels.

Non-striker glove exclusion:

- A fixed "no-go box" is defined around the non-striker's gloves and bat on the right side of the frame:
  - $x \in [0.60W, 0.90W]$
  - $y \in [0.35H, 0.80H]$
- Any candidate centroid inside this box is rejected, even if color and shape match.

These priors come from visual inspection of sample frames and are deliberately simple but effective for this fixed camera setup.

3.5 Temporal Filtering and Trajectory Building

Jump filter:

After region filtering, candidate centroids are mapped back from resized to original coordinates. A **jump filter** is applied:

- If the Euclidean distance from the previous accepted point exceeds a threshold ($max\_jump \approx 150$ pixels), the new point is treated as invalid (visible = 0).
- This prevents sudden jumps to wrong objects when the detector has a spurious hit[2][3].

Trajectory accumulation:

- Valid centroids are appended to a list of trajectory_points.
- For visualization:
  - A **red circle** is drawn at the current centroid.
  - A **green polyline** shows all stored trajectory_points, giving continuous trajectory overlay on each frame.

3.6 Trajectory Interpolation from Bowler to Stumps

The detector is most reliable near the stumps, where several consecutive frames have visible = 1 with consistent centroids.

Approach:

1. All frames with visible = 1 are treated as reliable anchor points.
2. Linear interpolation in time fills small gaps between detected frames.
3. For frames **before** the first detection, a parametric quadratic model extrapolates a **downward-facing parabolic path**:
   - Time index t = 0 at first detection; t < 0 for earlier frames
   - x(t) is linear in t: $x(t) = x_0 - v_x \times t$ (moving left as t decreases)
   - y(t) is quadratic: $y(t) = y_0 + v_y \times t + a \times t^2$ (with a > 0 for downward curve)

4. The interpolated centroids are written to a new CSV with visible = 1, then re-rendered onto the original video, producing a visually plausible full trajectory from bowler's hand to impact.

Note: Only the stumps-region points are actual detections; earlier frames are approximated.

4. Hyperparameter Tuning and Fallback Logic

4.1 Key Hyperparameters

Parameter	Value	Rationale
Frame resize width	960 px	Balance speed and detection accuracy
Gaussian blur kernel	5×5	Smooth noise while preserving edges
HSV H range	[0, 180]	Full hue spectrum (white is achromatic)
HSV S range	[0, 110]	Include slightly saturated off-whites
HSV V range	[180, 255]	Focus on bright pixels under floodlights
Min contour area	5 px^2	Detect small, distant balls
Max contour area	12000 px^2	Ignore large objects (players, shadows)
Min circularity	0.2	Allow blurred, slightly elongated balls
Jump threshold	150 px	Reject sudden jumps to wrong objects
Region height cutoff	0.75H	Exclude lower half (feet, umpire)
Region width bounds	[0.30W, 0.70W]	Central pitch lane
Glove box	$x \in [0.60W, 0.90W]$, $y \in [0.35H, 0.80H]$	Exclude non-striker region

4.2 Tuning Process

- **HSV thresholds** were tuned iteratively by checking masked frames and adjusting until the ball appeared consistently while gloves and boundary lines remained mostly suppressed[3][4].
- **Contour filters** (min_area, max_area, min_circularity) were relaxed to reduce missed detections when the ball is blurred or distant.
- **Region thresholds** (height cutoff and width bounds) were calibrated by checking the visual ball travel path across multiple overs.
- **Jump threshold** was empirically set to drop sudden jumps while allowing frame-to-frame motion.

4.3 Fallback Logic

- If no valid candidate is found in a frame, the system outputs (-1, -1, 0) but preserves the previously drawn trajectory.
- For visual continuity, interpolated points are added in post-processing to fill the path between reliable detections.
- If the interpolation extends beyond frame boundaries, it is silently clipped.

5. Evaluation and Results

5.1 Qualitative Evaluation

Because no ground-truth labels are provided, evaluation is qualitative:

- **Visual inspection** of processed videos shows the red circle tracking the ball correctly for most frames near the stumps.
- A smooth interpolated trajectory extends from bowler's hand to batsman's end.
- Per-frame CSVs confirm that visible = 1 occurs primarily when the ball is in the central pitch region, and (-1, -1, 0) is reported otherwise.

5.2 Example Output Format

Annotation CSV (first 10 frames of sample):

```
frame,x,y,visible
0,-1.0,-1.0,0
1,-1.0,-1.0,0
...
15,942.0,338.0,1
16,-1.0,-1.0,0
17,946.0,340.0,1
18,946.0,340.0,1
19,948.0,344.0,1
20,-1.0,-1.0,0
```

Processed video: MP4 with red circles on detected ball positions and green trajectory overlay.

5.3 Known Failure Modes

- **Color overlap:** When the ball overlaps with white pads or bat, color-based detection sometimes misses or briefly tracks the wrong object.
- **Motion blur:** Under extreme motion blur or glare from floodlights, the ball may fall outside the chosen HSV range and be marked invisible.
- **Extrapolation:** The extrapolated part of the trajectory (from bowler hand to first detection) is modelled, not directly detected, and may deviate from the true path.
- **Camera-specific:** Region filters are hard-coded for one camera angle and do not generalize to other viewpoints.

6. Repository Structure and Reproducibility

```
cricket_ball_tracking/
├── code/
│   ├── extract_frames.py # Frame extraction utility
│   ├── detect_ball.py # Main detection + tracking pipeline
│   ├── track_and_annotate.py # Re-rendering trajectories from CSV
│   ├── interpolate_trajectory.py # Parabolic trajectory fitting
│   └── utils.py # Preprocessing, masking, centroid
├── annotations/
│   └── 1_compromise_parabola.csv # Per-frame ball coordinates
├── results/
│   └── 1_traj_parabola.mp4 # Processed video with overlay
├── README.md # Setup and usage instructions
├── requirements.txt # Python dependencies
└── report.pdf # This document
```

All code is self-contained and requires no external training or pre-trained models.

7. Assumptions Made

1. **Single ball:** Only one cricket ball is present at a time in the video.
2. **Fixed camera:** The camera remains stationary; no camera motion or zoom.
3. **Consistent lighting:** Lighting conditions are stable (floodlights); no major shadows or reflections.
4. **White ball:** The ball color is white or light cream; red/pink balls would require different HSV ranges.
5. **Pitch visibility:** Most of the pitch and flight corridor are visible in the frame.
6. **No occlusion:** The ball is not occluded by players or other large objects for extended periods.

8. Future Work

8.1 Improved Detection

- Replace the HSV+contour detector with a learned object detector (e.g., YOLO or Faster R-CNN) trained on a dedicated cricket-ball dataset[2][5].
- Leverage domain-specific data augmentation (motion blur, occlusion, scale variation) to handle real-world conditions.

8.2 Better Trajectory Modeling

- Learn a Kalman filter or ballistic model for trajectory smoothing and speed estimation.
- Use physics-based priors (gravity, air resistance) to constrain interpolation between detections.

8.3 Multi-Camera and Generalization

- Extend to multiple camera views for 3D ball tracking.
- Develop automatic camera-agnostic region priors instead of hard-coded coordinates.
- Adapt to different pitch geometries (indoor, outdoor, different grounds).

8.4 Performance Metrics

- If ground-truth labels become available, compute precision, recall, F1-score, and trajectory error metrics.
- Benchmark against competing cricket-ball trackers in the literature.

9. References

[1] EdgeFleet.Ai AI/ML Assessment. Cricket Ball Detection and Tracking Task. Available online.

[2] Harsh Singh. CV_Project: Cricket Ball Tracking. GitHub. Retrieved from https://github.com/HarshSingh18/CV_Project

[3] Pyimagesearch. Ball Tracking with OpenCV. (2015). Available from https://pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

[4] OpenCV. Thresholding Operations using inRange. Documentation. Available from https://docs.opencv.org/4.x/da/d97/tutorial_threshold_inRange.html

[5] Kushagra. Cricket Ball Dataset for YOLO. Kaggle. (2024). Available from https://www.kaggle.com/datasets/kushagra3204/cricket-ball-dataset-for-yolo

---

Document Information

- **Author:** Cricket Ball Tracking System
- **Date:** December 2025
- **Assessment:** EdgeFleet.Ai AI/ML Assessment (NIT Trichy)
- **Submission Deadline:** 26-12-2025, 8:00 P.M.