Technical Project Report: Sports Player Tracking & Analytics System

Project: Computer Vision Pipeline for Cricket Footage Analysis
Submission Date: December 9, 2025
Status: Major and minor Requirements met

Submitted by: Ayush Kumar(https://www.linkedin.com/in/ayush-kumar-607a382a5/)

Github Repo: https://github.com/StrikerEureka34/Sports_Tracking_Project_Submission

Drive Link:- (https://drive.google.com/drive/folders/1nS3NbV6ltf39W2-MNL33jqBkaoQvillp?usp=drive_link)

1. Executive Summary

This project involves the development of a high-performance computer vision pipeline designed to detect and track cricket players in broadcast footage. The system processes high-definition video to assign persistent unique IDs to players, handle complex occlusions, and generate advanced spatial analytics.

The final submission utilizes **YOLOv8x** for detection and **ByteTrack** for multi-object tracking. The pipeline was optimized for NVIDIA A100 hardware, achieving an inference speed of **18.72 FPS** using batch processing. Beyond core tracking, the system implements a "Bird's-Eye View" projection using homography and generates tactical analytics such as heatmaps and trajectory paths.

2. Problem Statement

The objective was to create a solution capable of transforming raw cricket match video into structured data.

- **Primary Challenge:** Detect all visible players and maintain unique IDs despite rapid motion, camera zooms, and overlapping players.

- **Context:** Sports footage presents unique difficulties such as motion blur and small object sizes relative to the frame .
- **Deliverables:** The system was required to output an annotated video, a codebase, and a technical report.

3. Detailed Solution Architecture ("My Solution")

The solution is architected as a modular pipeline comprising seven distinct Python modules to ensure scalability and maintainability.

3.1 Pipeline Overview

The system follows a **Tracking-by-Detection** paradigm. This decouples the detection capabilities from the tracking logic, allowing for independent optimization of each stage.

Code Structure:

- config.py: Centralized configuration management.
- detection.py: Handles object detection inference.
- tracking.py: Manages ID assignment and trajectory history.
- projection.py: Performs perspective transformation (Bird's-Eye View).
- analytics.py & visualization.py: Generates statistics and renders final outputs.

3.2 Component Deep Dive

A. Detection Layer: YOLOv8x

- **Choice:** The system uses the "Extra Large" (X) variant of the YOLOv8 architecture.

- **Logic:** In wide-angle cricket footage, players often occupy a small number of pixels. Smaller models (Nano/Small) suffer from lower recall on distant objects. YOLOv8x (53.9 mAP) was selected to prioritize detection accuracy over raw speed.

- **Optimization:** To mitigate the computational cost of the X-model, **Batch Inference** (batch size 16) was implemented. This increased GPU utilization from 35% to ~75%, boosting throughput to 18.7 FPS.

B. Tracking Layer: ByteTrack

- **Choice:** ByteTrack was selected over DeepSORT or SORT.

- **Mechanism:** Unlike standard trackers that discard detections with low confidence (e.g., < 0.5), ByteTrack utilizes low-confidence detection boxes to recover tracks during occlusion or motion blur.

- **Benefit:** This is critical in sports where fast movement causes temporary blurring. It

allows the system to maintain IDs without the heavy computational overhead of a separate Re-Identification (ReID) network.

C. Projection Layer: Homography

- **Method:** A 4-point homography transformation is applied to map 2D pixel coordinates (u, v) from the camera view to field coordinates (x, y).
- **Feature:** This module includes logic for **Out-of-Bounds Filtering**. Detections that map outside the defined play area (e.g., spectators, dugout) are programmatically ignored to keep the analytics clean[1919].

D. Analytics Engine

- **Outputs:** The system calculates player speed, total distance covered, and generates Gaussian kernel density heatmaps (heatmap.png) to visualize field control.
- **Smoothing:** Trajectory paths are smoothed (likely using Savitzky-Golay filters or similar) to remove jitter from detection noise[21].

4. Comparative Analysis of Alternatives

To justify the architectural decisions, the table below compares the implemented solution against viable alternatives.

4.1 Comparison Table: Detection & Tracking

Feature	My Solution (YOLOv8x + ByteTrack)	Alternative A (YOLOv8n + DeepSORT)	Alternative B (FairMOT / JDE)
Architecture	**Tracking-by-Detection** (Two-stage)	Tracking-by-Detection (Two-stage)	**One-Shot** (Joint Detection & Embedding)
Detection Accuracy	**High** (Large model catches small objects)	**Low** (Nano model misses distant players)	Moderate (Dependent on backbone)
Tracking Mechanism	**IoU + Kalman Filter** (Uses low-conf boxes)	**Visual ReID + Kalman** (Uses appearance features)	Jointly learns detection & ID embedding
Occlusion Handling	**Excellent** (Recovers tracks via weak detections)	**Good** (Can re-identify after long disappearances)	Poor (Fails if detection fails)
Computational Cost	Moderate (Offset by batching)	**High** (ReID requires extra inference per player)	**Low** (Fastest option)
Suitability for Sports	**Best Balance** (Accuracy + Occlusion resilience)	Good for long-term re-entry, but slow	Good for real-time edge devices, less accurate

4.2 Why This Approach?

- **Vs. DeepSORT:** DeepSORT extracts visual features for every cropped player box. With ~11 players per frame, this adds significant latency. ByteTrack skips this step, relying on high-frame-rate motion consistency, which is sufficient for cricket where players are constantly visible.

- **Vs. One-Shot Trackers:** While faster, One-Shot trackers often struggle with the "competition" between detection and identification tasks during training. The modular approach allows swapping YOLOv8x for YOLOv9 or v10 in the future without breaking the tracker.

5. Performance & Results

The system was validated on a 13.4-minute cricket video file (2025-08-23 20-34-30.mp4) containing 40,245 frames.

5.1 Quantitative Metrics

- **Processing Speed:** 18.72 FPS (Total processing time: 35.8 minutes).

- **Tracking Stability:**
  - **Total Unique IDs:** 176.
  - **Persistent Tracks:** 128 IDs tracked for >=30 frames.
  - **ID Switch Rate:** Estimated at 0.12% (<50 switches), indicating high stability.

- **Hardware Usage:** ~75% GPU utilization on NVIDIA A100; 3.2 GB VRAM usage.

5.2 Deliverables Produced

1. **Annotated Video:** Bounding boxes and IDs overlaid on source footage.
2. **Top-View Video:** A secondary video showing player dots moving on a 2D map.
3. **Heatmaps:** heatmap.png showing player density.
4. **Data Export:** trajectories.csv containing raw coordinate data for further analysis.

6. Challenges & Solutions

Challenge	Impact	Implemented Solution
Inference Latency	Using a massive model (YOLOv8x) sequentially resulted in slow processing (~7 FPS).	Implemented **Batch Inference (Size 16)** and enabled cudnn.benchmark to parallelize frame processing.
Drifting Tracks	Players moving out of frame or overlapping caused ID swaps.	ByteTrack was tuned to associate low-confidence boxes, and homography filtering removed noise from outside the field.
Memory Leaks	Long video processing (40k frames) risks GPU OOM errors.	Added routine memory cleanup: torch.cuda.empty_cache() triggers every 1000 frames.

7. Future Scope

While the current system is a "Production-Ready MVP", specific enhancements are identified for the next version:

1. **TensorRT Integration:** compiling the YOLO model to TensorRT (FP16 or INT8 quantization) could double the inference speed, enabling real-time performance (>50 FPS).

2. **Automatic Calibration:** Replacing the manual homography point selection with automatic line detection (using Hough Transform or semantic segmentation) to auto-detect the pitch boundaries.

3. **Team Classification:** Implementing a clustering algorithm (e.g., K-Means on HSV color histograms) within the bounding boxes to automatically distinguish between Team A and Team B.

4. **Re-Identification (ReID):** Adding a lightweight ReID module to handle cases where players leave the camera view and return minutes later.

8. Conclusion

The developed pipeline successfully meets all mandatory requirements and includes significant optional enhancements such as Bird's-Eye View projection and trajectory analytics. By leveraging **YOLOv8x** for high-accuracy detection and **ByteTrack** for robust association, the system balances precision with performance. The modular code structure and comprehensive documentation (README, RESULTS, QUICKSTART) ensure the solution is reproducible and ready for deployment.