Table Tennis Object Tracking and Analysis

Authors: Lakshay Baijal (2024202006), Rishabh Sahu (2024201037), S. V. Mohit Kumar (2024201010)

Github Link - *https://github.com/LakshayBaijal/SMAI_Project*

---

1. Problem Statement

We aim to build an AI-driven sports-analytics system that detects and tracks the table tennis ball in match footage. Given recorded match video, our system locates the ball in most of the frame, reconstructs its trajectory, and computes per-frame speed and flight angle.

2. Motivation

In a table tennis match, manually analyzing ball movements and key events is a tedious task. Table tennis is a high-speed sport in which manual analysis is impractical, leading to inaccuracies in performance evaluation and refereeing. By developing an AI-based ball tracking and analysis system, our aim is to provide automated, precise, and real-time insights into ball trajectory, and game patterns.

Manual analysis of high-speed table tennis footage is time-consuming, error-prone, and often misses critical fast events. An automated solution provides:

- **Precision:** pixel-level ball localization
- **Objectivity:** consistent, unbiased event detection
- **Real-time feedback:** instant metrics for coaches and broadcasters

3. Use Cases & Stakeholders

1. **Performance Analysis (Players & Coaches):** quantify shot speeds, angles, and trajectories to refine technique.
2. **Automated Officiating (Umpires):** assist in net-hit and fault calls.
3. **Sports Broadcasting (Analysts):** overlay ball-cam trajectories for engaging replays.
4. **AR Training Tools (Developers):** integrate live ball data into training simulators.

4. Related Work

- **YOLO (You Only Look Once):** single-pass real-time object detector.
- **DeepBall:** CNN specialized for small-object detection in sports videos.
- **TTNet:** cascaded, dual-resolution architecture for ball segmentation and event-spotting at 120 fps.

5. Background

Table tennis presents unique challenges for object tracking. The ball is small and moves very fast, requir ing high frame rates and precise algorithms. Player movements are also rapid and complex. Therefore, many existing object detection techniques that gave good results with sports like football failed to give comparable results when trying to achieve the same goals with Table Tennis. By making modifications to existing approaches and applying novel techniques suited specifically to the domain of Table Ten nis researchers have gained impressive results and were able to perform Table tennis ball detection and events detection with good accuracy.

6. Key Findings & Trends

- **TTNet** achieved ~97 % accuracy in event-spotting and pixel-level ball masks at 120 fps on consumer GPUs.
- **Cascaded detectors** (coarse low-res + fine high-res) improve small-object precision.
- **Real-time inference** (<10 ms per frame) is critical for live analytics.

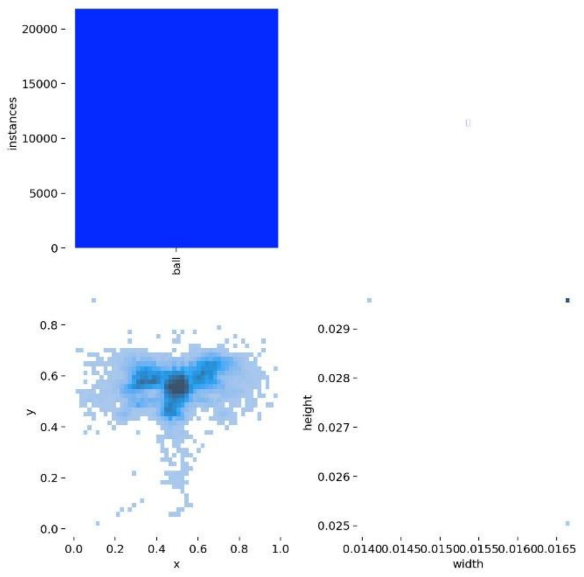
7. Gaps & Challenges

- **Occlusion handling:** lost detections when ball is hidden by players or net.
- **Camera-angle dependency:** current systems assume fixed viewpoints.
- **Dataset diversity:** limited lighting conditions and backgrounds.
- **Single-object focus:** ignores players, rackets, and environmental constaints.

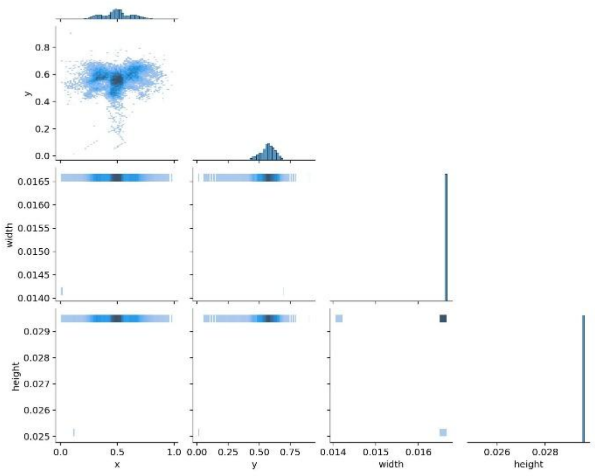
8. Data Sources

- **OpenTTGames**: 5 HD videos at 120 fps (around 1 lakh frames) annotated with ball coordinates, segmentation masks, and in-game events.
- **Preparation**:
    - Download dataset - to fetch TTNet videos & JSON.
    - Extract all images and Extract selected images to dump frames.
    - Convert ball to yolo to convert JSON events to YOLO .txt labels.

9. Data Exploration



Label-count distribution for the ball class.



Correlogram showing joint distributions of YOLO label fields (x, y, width, height).

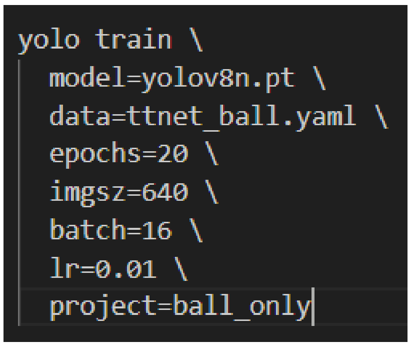
10. Model Selection & Configuration

We chose **YOLOv8n** for its sub-7 ms inference and strong small-object mAP.

- **Environment**: Python 3.10, Ultralytics YOLOv8, OpenCV, NumPy.
- **Data config**: ttnet_ball.yaml defines single class ball with train/val splits.
- **Weights**: initialized from yolov8n.pt.

11. Training & Evaluation

11.1 Training Command

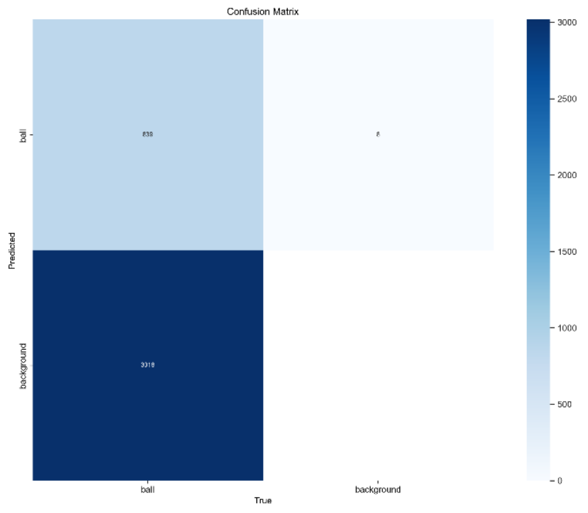


```
yolo train \
  model=yolov8n.pt \
  data=ttnet_ball.yaml \
  epochs=20 \
  imgsz=640 \
  batch=16 \
  lr=0.01 \
  project=ball_only
```

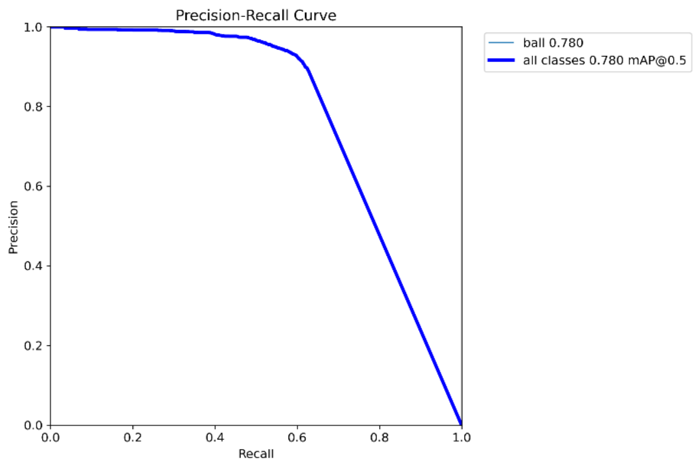
YOLO Commands for training

- **yolo train \**
  Invokes Ultralytics' YOLOv8 training routine.
- **model=yolov8n.pt \**
  Starts fine-tuning from the pre-trained "nano" weights for speed and efficiency.
- **data=ttnet_ball.yaml \**
  Points to our YAML file that defines the train/validation splits, class names, and dataset paths.
- **epochs=20 \**
  Runs the optimization loop over the full dataset 20 times.
- **imgsz=320 \**
  Resizes all training and validation images to 640×640 pixels for uniform input.
- **batch=16 \**
  Processes 16 images per gradient update (per GPU).
- **lr=0.01 \**
  Sets the initial learning rate to 0.01 before any decay schedule.

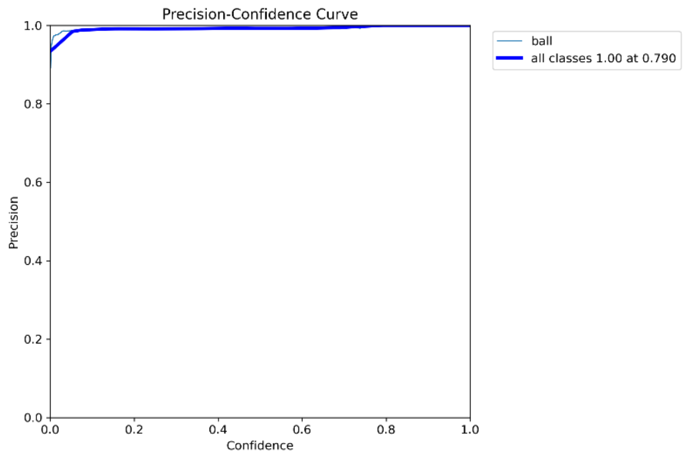
11.2 Evaluation Metrics



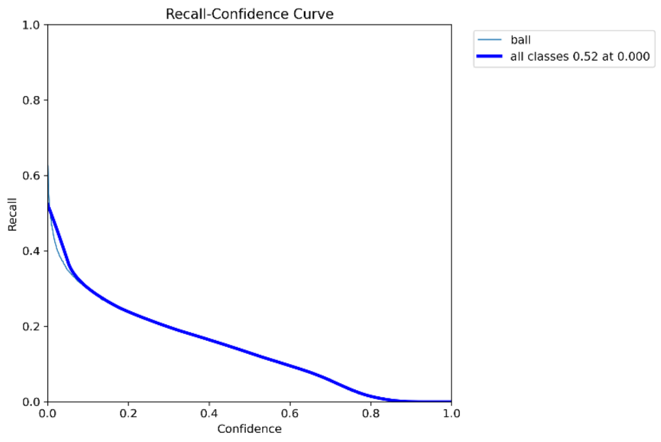
Confusion matrix on the test set - True Positives: 839, False Negatives: 8, False Positives: 3018



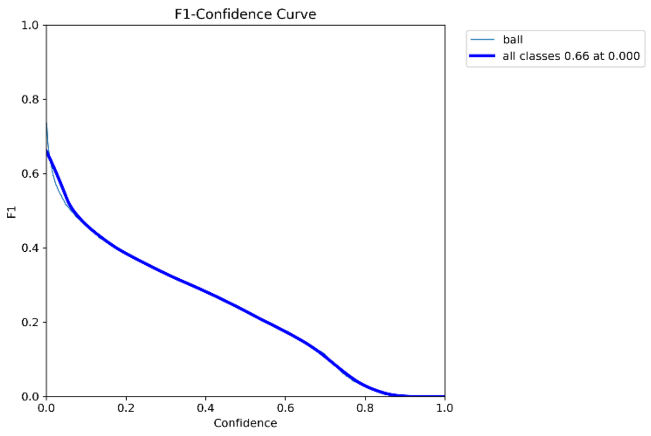
mAP@0.5 = 0.78 at optimal threshold



Precision vs. confidence (precision = 1.00 at confidence = 0.79)

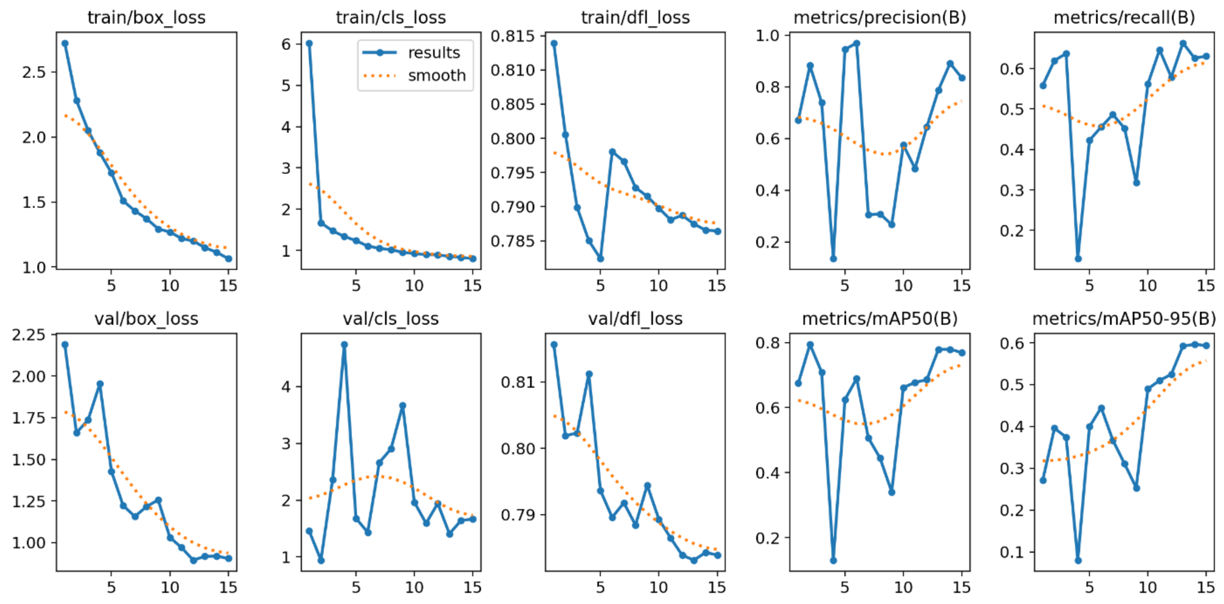


Recall vs. confidence (recall = 0.52 at confidence = 0.00 – Max Coverage)



F1-score vs. confidence (F1 = 0.66 at confidence = 0.00).

11.3 Performance Analysis

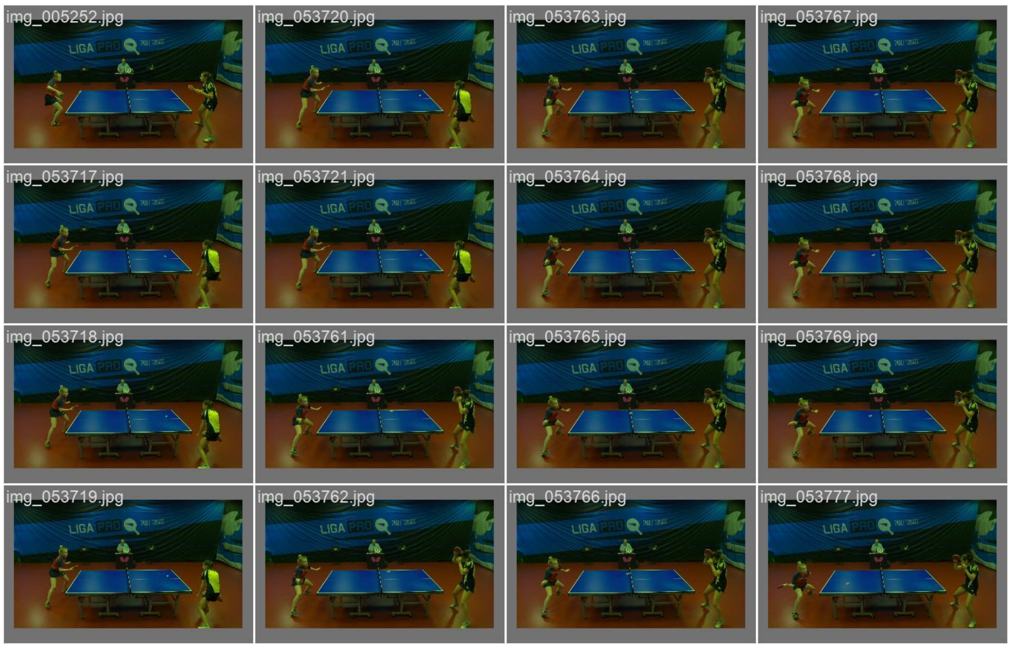


Training and validation loss & metric trends over 15 epochs.

- Achieved final **mAP@50** ≈ 0.78; stable convergence by epoch 15.
- Box loss steadily decreases from 2.7 → 1.05.
- Classification loss drops from 6.0 → 0.91.
- Validation mAP@0.5 converges to 0.78 by epoch 15.
- Mixed-precision (FP16) on T4 GPU reduced training time by ~30%.

12. Code Documentation (Example Runs)

Sample training-batch gallery showing predicted "ball" labels.



Sample validation-batch predictions (no detections in some frames).

13. Calculations

Calculation of Speed & Movement Angle

Let

- Frame *(t − 1)* → ball center at (x1, y1)
- Frame *t* → ball center at (x2, y2)

1. Displacement & Speed (px/frame)

- Compute the horizontal and vertical displacements: $\Delta x = x2 - x1, \Delta y = y2 - y1$
- Euclidean distance (pixels/frame):

2. Movement Angle (°)

- Take the 2D movement vector.
- Compute the radian angle relative to the positive xxx-axis:

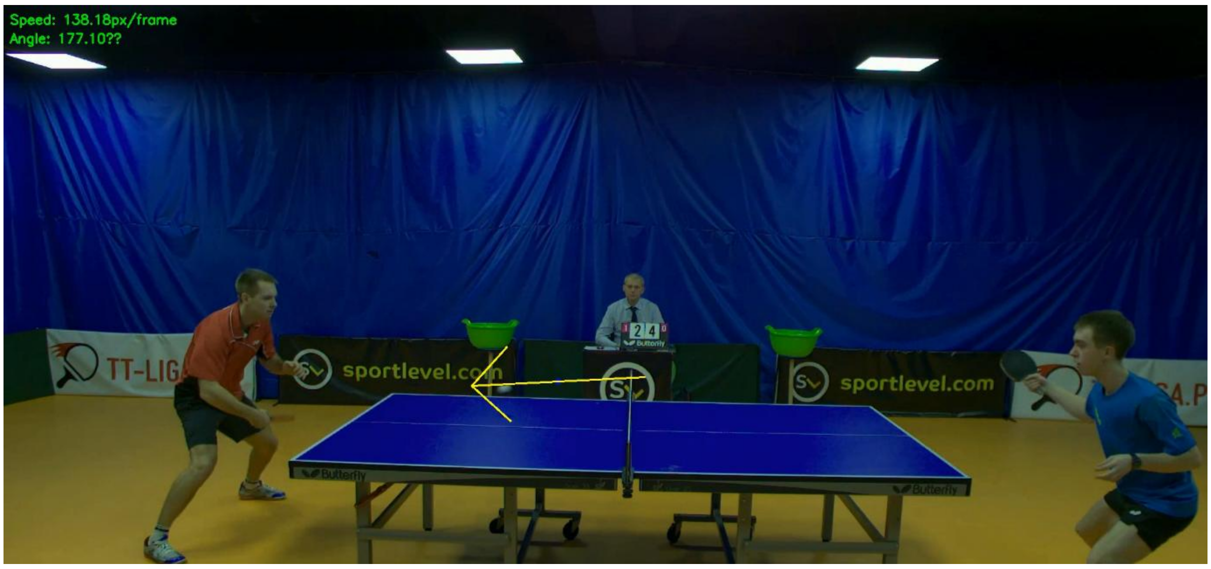
The atan2 function chooses the correct quadrant:

- 0° → rightward
- $+90^\circ$ → downward
- -90° → upward
- $\pm180^\circ$ → leftward

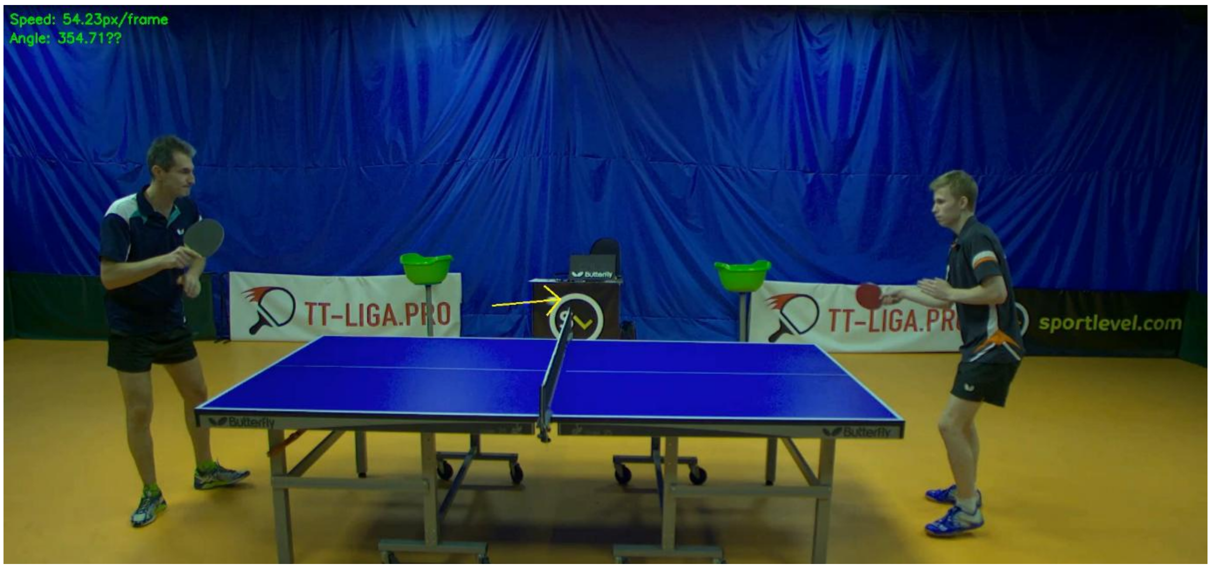
And then convert it to degrees.

14. Conclusion

- **Results:** ~0.78 mAP@0.5 with ~150 fps inference—the system reliably tracks the ball in real time.



Result 1 - for dataset video 1



Result 2 for dataset video 2

- **Limitations:** speeds in px/frame (no real-world units without calibration); occasional misses under heavy occlusion.
- **Future Work:** apply camera calibration for px→m/s conversion, integrate Kalman filtering for smoother tracks, extend to multi-object (players, rackets) tracking.

15. Project Links

Github Link - *https://github.com/LakshayBaijal/SMAI_Project*

16. References

1. *Komorowski, Kurzejamski, Sarwas. DeepBall: Deep neural-network ball detector, ArXiv 2019.*
2. *Voeikov, Falaleev, Baikulov. TTNet: Real-time temporal and spatial video analysis of table tennis, CVPR Workshops 2020.*
3. *OSAI. OpenTTGames Dataset.*
4. *KGBUPT. Table-Tennis Detector Dataset, Roboflow Universe, Aug 2023.*