Tennis/Pickleball Single-Camera Tracking System

Progress Report & Task Allocation

Course: Computer Vision
Instructor: Nguyen Duc Dung

Team Members:
1. Pham Hoang An 2. Dao Duy Anh 3. Duong Vu Duc
4. Nguyen Duc Dat 5. Bui Dinh Nguyen Minh

GitHub: https://github.com/AnHgPham/cv

Report date: February 23, 2026

Contents

1 Abstract

This project implements a complete single-camera computer vision pipeline for tennis and pickleball video analysis. The system processes monocular video (25–30 fps) and produces:

- Real-time ball trajectory detection and tracking,
- Player detection with persistent identity assignment,
- Court-line detection and image-to-court homography estimation,
- Physics-based 3D trajectory reconstruction of the ball,
- Bounce detection with in/out classification,
- Annotated output video with bird's-eye mini-map and heatmaps.

This report presents the current implementation status, documents bugs identified and fixed during development, highlights remaining issues (primarily an incorrect homography due to multi-court interference), and defines a task allocation for a 5-member team to deliver a stable, evaluated system.

2 Repository and Project Structure

The full source code is hosted on GitHub:

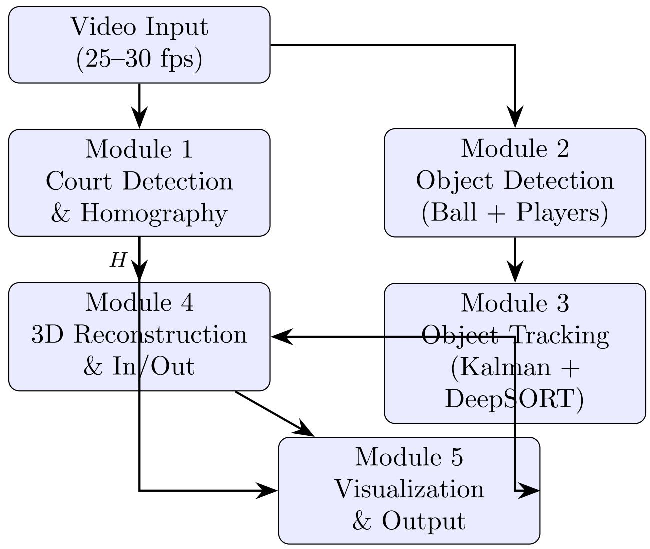
- Repository: https://github.com/AnHgPham/cv
- Project root: `cv/tennis-pickleball-tracker/`

2.1 Directory Layout

```
tennis-pickleball-tracker/
|-- src/
|   |-- court_detection.py      # Module 1 (~25 KB, ~720 lines)
|   |-- object_detection.py     # Module 2 (~35 KB, ~400 lines)
|   |-- object_tracking.py      # Module 3 (~24 KB, ~540 lines)
|   |-- trajectory_3d.py        # Module 4a (~31 KB, ~680 lines)
|   |-- in_out_classifier.py    # Module 4b (~9 KB, ~250 lines)
|   |-- pipeline.py             # Module 5a (~36 KB, ~940 lines)
|   |-- visualization.py        # Module 5b (~21 KB, ~630 lines)
|   `-- __init__.py
|-- notebooks/
|   |-- 01_data_exploration.ipynb
|   |-- 02_court_detection.ipynb
|   |-- 03_ball_detection.ipynb
|   |-- 04_tracking.ipynb
|   `-- 05_3d_reconstruction.ipynb
|-- configs/court_config.yaml
|-- data/raw/                   # Input videos (not in git)
|-- outputs/                    # Generated results (not in git)
|-- run_pipeline.py             # CLI entry point
|-- test_fix.py                 # Single-frame debug script
`-- report.tex                  # This report
```

3 System Architecture

3.1 Pipeline Flow Diagram



3.2 Per-Frame Processing Steps

For every frame t in the input video:

1. **Court Detection (Module 1):** Detect white court lines via HSV thresholding, extract edges with Canny, find line segments with Hough transform, compute line intersections, select 4 court corners, and estimate a 3×3 homography matrix H that maps pixel coordinates (u, v) to real-world court coordinates (x_c, y_c) in metres.
2. **Object Detection (Module 2):** Run YOLOv8 on the frame to detect persons and sports balls. Supplement with a classical contour-based ball detector for small tennis balls that YOLO may miss. Apply non-maximum suppression (NMS) to remove duplicate detections.
3. **Player Filtering (Module 5):** Score each player detection by (a) x-distance from frame centre (weight 0.60), (b) perspective-consistent size (weight 0.25), (c) YOLO confidence (weight 0.15). Hard-reject detections whose centre exceeds 40% of frame width from the frame centre. Keep top k (default $k = 4$ for doubles).
4. **Tracking (Module 3):** Update the ball Kalman filter with the new detection (or predict if no detection). Use Lucas-Kanade optical flow as a secondary motion estimate. Update the player tracker (DeepSORT or IoU fallback) to maintain persistent IDs.
5. **3D Reconstruction (Module 4):** Project the tracked ball position through H to obtain court coordinates. Estimate ball height using a physics model (gravity $g = 9.81\,\mathrm{m/s^2}$, air drag). Detect bounces from trajectory direction changes. Classify each bounce as in or out by checking whether (x_c, y_c) falls within the standard court rectangle.
6. **Visualization (Module 5):** Draw bounding boxes, ball trajectory trail, info overlay, and compose the output frame with a side-panel mini-map and frame metadata.

4 Module Details

4.1 Module 1: Court Detection & Homography

File: `src/court_detection.py`
Owner: Pham Hoang An

4.1.1 Technical Approach

1. **Pre-processing:** Convert BGR → HSV; threshold for white pixels (H: 0–180, S: 0–50, V: 180–255); morphological dilation (2 iterations) and erosion (1 iteration) with a 5×5 rectangular kernel.
2. **Edge Detection:** Canny with thresholds (50, 150).
3. **Line Detection:** Probabilistic Hough Transform ($\rho = 1$, $\theta = \pi/180$, threshold= 100, minLineLength= 50, maxLineGap= 30).
4. **Line Classification:** Separate into horizontal and vertical groups based on angle ($\pm 30°$ from horizontal/ vertical). Merge similar lines within 30 px distance.
5. **Intersection Computation:** Compute all pairwise intersections of horizontal and vertical lines.
6. **Corner Selection:** Compute centroid of all intersection points; assign each point to a quadrant (TL, TR, BL, BR) relative to the centroid; pick one representative per quadrant.
7. **Homography:** Compute H via `cv2.findHomography` with RANSAC (reproj. threshold = 5.0 px) mapping 4 image corners → standard court corners (tennis: 23.77×10.97 m).

4.1.2 Additional Components

- **SIFT Court Matcher:** Computes frame-to-frame homography using SIFT keypoints + FLANN matcher + Lowe's ratio test, enabling camera-motion compensation.
- **Deep Court Detector (scaffolded):** CNN-based keypoint regression using a ResNet-18 backbone. Not yet trained.

4.1.3 Known Constants

Standard tennis court keypoints are defined in `TENNIS_COURT_KEYPOINTS` (21 points in metres) and `TENNIS_COURT_CORNERS` (4 corners). Pickleball court keypoints are also defined.

4.1.4 Applied Knowledge

Image processing (colour spaces, morphology), Canny edge detection, Hough transform, SIFT feature matching, homography estimation (DLT + RANSAC), convolutional neural networks.

4.2 Module 2: Object Detection

File: `src/object_detection.py`
Owner: Dao Duy Anh

4.2.1 Detectors Implemented

1. **YOLODetector:** Wraps Ultralytics YOLOv8.

- Supports both COCO-pretrained models (class 0 = person, class 32 = sports ball) and custom-trained models (user-defined class map).
- Configurable confidence threshold (default 0.25), device (CPU/CUDA).
- Detected 9 persons in a typical frame of our test video.

2. **ClassicalBallDetector:** A multi-stage pipeline for detecting small tennis balls that YOLO often misses:
   - HSV colour filtering for yellow/green tennis balls (H: 20–50, S: 50–255, V: 120–255).
   - Contour extraction with circularity filter ($4\pi A/P^2 > 0.6$).
   - Size filtering (radius 3–30 px).
   - Motion detection via frame differencing.
   - Court-region filtering (optional, uses court polygon).
3. **TrackNetDetector (scaffolded):** A specialised heatmap regression network that uses 3 consecutive frames as input to detect fast-moving balls.
4. **ClassicalPlayerDetector:** HOG + SVM or Haar cascade baseline for player detection.
5. **FasterRCNNDetector (scaffolded):** Two-stage detector using torchvision's pre-trained Faster R-CNN.

4.2.2 Detection Priority

The pipeline tries detectors in this order: classical → YOLO → TrackNet. The first detector to return a ball detection "wins" for that frame.

4.2.3 Applied Knowledge

Feature engineering (HOG, Haar), deep object detection (YOLO, Faster R-CNN), heatmap regression (TrackNet), non-maximum suppression, colour space analysis, contour geometry.

4.3 Module 3: Object Tracking

File: `src/object_tracking.py`
Owner: Duong Vu Duc

4.3.1 Ball Tracking

- **Kalman Filter:** State vector $\mathbf{x} = [x, y, v_x, v_y]^T$. Constant-velocity motion model. Configurable process noise (default 5.0) and measurement noise (default 2.0).
- **Optical Flow:** Lucas-Kanade sparse optical flow around the last known ball position as a secondary motion estimate.
- **Fusion:** Weighted combination of Kalman prediction and optical flow result. If no detection for > 10 frames, the track is dropped.

4.3.2 Player Tracking

- **DeepSORT (preferred):** Uses deep appearance features + Kalman prediction + Hungarian algorithm for data association. Requires `deep-sort-realtime` package.
- **IoU Tracker (fallback):** Simple IoU-based matching between consecutive frames. Currently active because `deep-sort-realtime` is not installed. Track IDs may be less stable under occlusion or missed detections.

4.3.3 Applied Knowledge

Kalman filtering (linear state estimation), Lucas-Kanade optical flow, Hungarian algorithm (optimal assignment), deep metric learning (DeepSORT appearance model), IoU-based data association.

4.4 Module 4: 3D Trajectory Reconstruction & In/Out Classification

Files: `src/trajectory_3d.py`, `src/in_out_classifier.py`
Owner: Nguyen Duc Dat

4.4.1 Trajectory Reconstruction

- Projects 2D ball pixel positions (u, v) to court coordinates (x_c, y_c) using the homography H.
- Estimates ball height $z(t)$ using a physics model: $z(t) = z_0 + v_{z0}\,t - \frac{1}{2}g\,t^2$ with optional air-drag correction.
- Uses an Extended Kalman Filter for 3D state smoothing.

4.4.2 Bounce Detection

- Detects frames where the vertical velocity v_z changes sign (trajectory direction reversal).
- Filters spurious bounces using minimum height and time constraints.

4.4.3 In/Out Classification

- Checks whether the bounce point (x_c, y_c) falls within the standard court rectangle (with line-width tolerance).
- Outputs a confidence score based on distance to the nearest boundary.
- ML-enhanced mode (scaffolded): uses nearby trajectory features for more robust classification.

4.4.4 Applied Knowledge

Projective geometry, coordinate transformations, physics-based modelling (projectile motion, air drag), Extended Kalman filter, geometric classification.

4.5 Module 5: Pipeline Integration & Visualization

Files: `src/pipeline.py`, `src/visualization.py`
Owner: Bui Dinh Nguyen Minh

4.5.1 Pipeline Orchestration

- `TennisPickleballPipeline` class manages all module instances and state (homography, court polygon, ball history).
- `PipelineConfig` supports YAML config files and CLI arguments for all parameters.
- Court detection is re-run every 30 frames to handle gradual camera drift.
- Player filtering uses a multi-factor scoring system:

$$S = 0.15 \cdot c_{\text{conf}} + 0.25 \cdot c_{\text{size}} + 0.60 \cdot c_{\text{court}}$$

  where c_{court} penalises players far from the frame centre (proxy for main-court membership).

4.5.2 Visualization Components

- **FrameAnnotator:** Draws bounding boxes (colour-coded by player ID), ball trajectory trail with fade effect, bounce markers (green circle = in, red X = out), and info overlay.
- **MiniMap:** Renders a 300×600 px bird's-eye view of the court with proper scaling (23.77×10.97 m for tennis). Supports drawing ball position, player positions, ball trail, and bounce markers. Currently only receives ball position from the pipeline.
- **HeatmapGenerator:** Accumulates ball and player positions in a discretised grid (0.1 m resolution), applies Gaussian smoothing ($\sigma = 2$), and renders via matplotlib with court line overlay.
- **CompositeFrameBuilder:** Combines the main annotated frame (1280×720) with a side panel containing the mini-map (200×400) and text info.

4.5.3 Data Preprocessing Utilities

- Frame extraction from video at configurable intervals.
- Resize and normalise frames for TrackNet / YOLO input.
- Dataset splitting (train/val/test) with configurable ratios.
- Annotation format conversion (to YOLO format).

4.5.4 Applied Knowledge

Software engineering (modular pipeline design, configuration management), video I/O (OpenCV VideoCapture/VideoWriter), data visualisation, image composition.

5 Bugs Found and Fixed During Development

During the development and testing phase, the following bugs were identified and resolved:

1. **YOLO Class Mapping Error (Module 2).** The original code assumed class indices {0: "ball", 1: "player"}, which is correct for custom-trained models but *wrong* for COCO-pretrained YOLOv8, where class 0 = "person" and class 32 = "sports ball".

   Fix: Added an `is_custom_model` parameter to `YOLODetector`. In COCO mode, the detector uses {`0: ''player'', 32: ''ball''`} and filters for classes `[0, 32]`.

2. **Player Detection Included Adjacent Courts (Module 5).** With 9 YOLO person detections in a typical frame, the system initially displayed all of them, including spectators and players on adjacent courts.

   Fix: Implemented a multi-factor scoring system (Section 3) that scores each detection by proximity to the frame centre, perspective-consistent size, and confidence. A hard cutoff at 40% of frame width rejects far-edge detections. Top $k = 4$ are kept for doubles.

3. **Ball Detection Missed Small Tennis Balls (Module 2).** YOLOv8 with COCO weights frequently missed the tennis ball (small, fast-moving, $\sim$10 px diameter).

   Fix: Rewrote `ClassicalBallDetector` with contour analysis, circularity filtering, motion detection via frame differencing, and court-region filtering. The pipeline now tries classical detection first, then falls back to YOLO.

4. **Import Error in Tracking Module (Module 3).** A relative import of `_compute_iou` failed when running the module standalone.

   Fix: Added try/except fallback for both relative and absolute import paths.

5. **Pipeline Ball Label Mismatch (Module 5).** The pipeline's ball filter checked for `"ball"` but COCO models return `"sports ball"`.

   Fix: Updated the filter to accept both: `("ball", "sports ball")`.

6 Remaining Issue: Incorrect Homography

6.1 Symptom

When the input video contains multiple adjacent courts (as in our test video), the mini-map ball position is misplaced and all bounces are classified as OUT, even though the ball visually lands inside the main court.

6.2 Root Cause Analysis

1. **Corner selection picks wrong court.** The classical court detector finds line intersections from *all visible courts* (15 intersection points detected; many at extreme coordinates such as $x = -3562$ or $x = 3313$). The centroid-quadrant heuristic in `_select_court_corners` mixes intersections from the main court with those from adjacent courts, producing a degenerate homography.

2. **Homography validation is absent.** There is no sanity check on the computed homography. Projecting standard court corners through H^{-1} back to image space yields a polygon with vertices at $(x_{\min}, y_{\min}) = (700, -325)$ and $(x_{\max}, y_{\max}) = (2910, 1080)$, far exceeding the 1920×1080 frame. The near-side player's foot position $(583, 1075)$ projects to court coordinates $(-6.56, 5.19)$ instead of a point inside $[0, 23.77] \times [0, 10.97]$.

3. **Mini-map does not show player positions.** The `MiniMap.render()` method supports `player_court_positions` but the pipeline integration code only passes `ball_court_pos`, so the mini-map displays only a (mis-mapped) ball dot on a static court diagram.

6.3 Impact

- All 4 detected bounces are classified as OUT (incorrect).
- Ball heatmap accumulates at wrong court positions.
- Mini-map is essentially static and uninformative.
- Modules 4 and 5 cannot be properly validated until the homography is corrected.

7 Proposed Fixes

7.1 Fix A: Robust Corner Selection for Homography (Module 1)

1. **Intersection filtering:** Remove all intersections outside the frame ($\pm 5\%$ margin). Cluster remaining points and select the largest cluster as the main court.
2. **Aspect-ratio constraint:** Among candidate 4-point subsets, select the one whose projected rectangle best matches the standard court aspect ratio (≈ 2.17).

3. **Reprojection sanity check:** After computing H, project standard court corners back to image space. Reject if: (a) any projected point is more than 20% outside the frame, (b) the projected polygon is not convex, or (c) reprojection error exceeds 10 px.
4. **Temporal stabilisation:** Cache the last valid H and only replace it when a new H passes all sanity checks with better quality. Optionally average H matrices over a sliding window.

7.2 Fix B: Show Players on Mini-map (Module 5)

1. For each tracked player, compute foot point = bottom-centre of bounding box: $(\frac{x_1+x_2}{2}, y_2)$.
2. Project foot point through H to court coordinates.
3. Pass the list of court positions to `MiniMap.render( player_court_positions=...)`.
4. Guard against bad projections (NaN, out-of-court-range) by clamping or skipping.

7.3 Fix C: Court-Line Overlay on Video Frame (Module 5)

1. Project all 21 standard tennis court keypoints through H^{-1} to image coordinates.
2. Draw the projected lines as a semi-transparent overlay on the video frame, providing immediate visual feedback on homography quality.

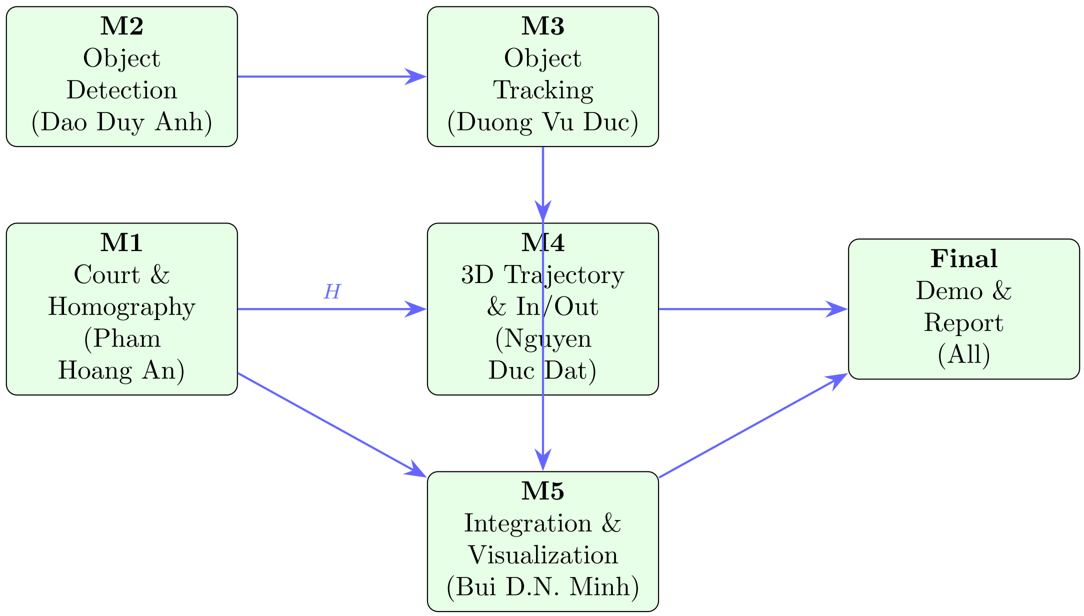
8 Task Allocation

Member	**Module**	**Responsibilities & Deliverables**	**Status**
Pham Hoang An	Module 1: Court Detection & Homography	• Filter intersections: remove out-of-frame points; cluster remaining points to isolate the main court. • Implement aspect-ratio constraint for corner selection. • Add reprojection sanity check: reject degenerate homographies. • Temporal stabilisation: cache last valid H, smooth updates. • (Optional) Train the scaffolded ResNet-18 court keypoint detector on labelled data. • Write unit tests for single-court and multi-court scenarios. • **Deliverables:** updated `court_detection.py`, before/after comparison images, test report. • **Knowledge:** Image processing, Hough Transform, RANSAC, homography, SIFT, CNN.	In progress

Member	Module	Responsibilities & Deliverables	Status
Dao Duy Anh	Module 2: Object Detection	• Benchmark ball detection: classical vs. YOLOv8 (COCO) vs. TrackNet. Report detection rate, precision, recall on 100+ annotated frames. • Tune classical detector parameters (HSV range, contour circularity threshold, size limits). • Reduce false-positive ball detections using court polygon constraint (reject candidates outside court area). • Evaluate player detection: measure precision/recall, analyse failure cases (occlusion, small far-side players). • (Optional) Fine-tune YOLOv8 on tennis-specific dataset or implement TrackNet training. • **Deliverables:** detection evaluation report with metrics, optimised config, updated `object_detection.py`. • **Knowledge:** HOG, Haar cascades, YOLO, Faster R-CNN, TrackNet, NMS, colour-space filtering.	In progress
Duong Vu Duc	Module 3: Object Tracking	• Install `deep-sort-realtime` and integrate with the player tracker; ensure fallback IoU tracker still works. • Calibrate Kalman filter parameters (process noise, measurement noise) for smooth ball trajectories with minimal lag. • Tune optical flow parameters (window size, pyramid levels) for reliable motion estimation. • Evaluate tracking quality: count ID switches per 100 frames, measure track continuity. • Handle edge cases: ball occlusion by net, rapid direction changes, players crossing paths. • **Deliverables:** tracking evaluation report (ID switches, continuity), demo clips, parameter docs, updated `object_tracking.py`. • **Knowledge:** Kalman filter, Lucas-Kanade optical flow, DeepSORT, Hungarian algorithm, IoU matching.	In progress

Member	Module	Responsibilities & Deliverables	Status
Nguyen Duc Dat	Module 4: 3D Trajectory & In/Out	• After M1 homography fix: verify that ball court coordinates are plausible ($0 \leq x_c \leq 23.77$, $0 \leq y_c \leq 10.97$). • Tune bounce detection: adjust velocity-change threshold, minimum inter-bounce time, confidence filtering. • Validate in/out classification on 3–5 clips; report accuracy. • Calibrate physics model parameters (initial height, drag coefficient) for realistic 3D trajectories. • (Optional) Implement ML-enhanced bounce classification using trajectory features. • **Deliverables:** bounce analysis report (per-clip results, confusion matrix), updated `trajectory_3d.py` and `in_out_classifier.py`. • **Knowledge:** Projective geometry, physics modelling, Extended Kalman filter, classification.	Blocked (depends on M1)
Bui Dinh Nguyen Minh	Module 5: Integration & Visualization	• Integrate player foot-point projection onto mini-map (`player_court_positions`). • Add court-line overlay on main video frame using H^{-1} projection of standard court keypoints. • Improve `CompositeFrameBuilder` layout: larger mini-map, score/stats panel. • Package `run_pipeline.py` as a clean CLI entry point with progress bar and result summary. • Produce final annotated demo video and output report. • Write comprehensive README with installation steps, usage examples, and architecture diagram. • **Deliverables:** final demo video, updated `pipeline.py` and `visualization.py`, README, run instructions. • **Knowledge:** Software engineering, pipeline design, video I/O, data visualisation, documentation.	In progress

9 Dependency Graph and Critical Path



Critical path: M1 (Homography fix) → M4 (3D + In/Out) → Final demo. The homography fix is the *highest priority* because Modules 4 and 5 cannot produce correct court-mapped results without a valid H.

Parallel work: M2 (Detection benchmarking) and M3 (Tracking stabilisation) can proceed independently of M1.

10 Milestones

1. **Milestone 1 – Homography Fix (Pham Hoang An):** Robust corner selection, reprojection sanity check, temporal stabilisation. Verified on multi-court test video.

2. **Milestone 2 – Detection Benchmark (Dao Duy Anh):** Comparative evaluation of all ball detection methods. Optimised default configuration. False-positive analysis.

3. **Milestone 3 – Stable Tracking (Duong Vu Duc):** DeepSORT integrated. ID switch count minimised. Ball trajectory smooth and continuous.

4. **Milestone 4 – Bounce/In-Out Validation (Nguyen Duc Dat):** Correct bounce detection on 3–5 validation clips. In/out accuracy report. Depends on Milestone 1.

5. **Milestone 5 – Final Demo (Bui Dinh Nguyen Minh + All):** Mini-map shows ball and players. Court overlay on video. Final annotated output video. Documentation and report submission.

11 Test Data and Validation

11.1 Primary Test Video

- **File:** `data/raw/Video Project 4.mp4`

- **Resolution:** 1920×1080 pixels
- **Frame rate:** 30 fps
- **Duration:** 1139 frames (≈38 seconds)
- **Scene:** Outdoor tennis facility, camera positioned above/behind one baseline. The main court is centre-left in the frame. At least one adjacent court is visible on the right, which causes interference in court detection.
- **Match type:** Doubles (4 players on the main court).

11.2 Latest Pipeline Results (Before Homography Fix)

- Total frames processed: 1139
- Processing time: ∼636 s (1.8 fps on CPU)
- Ball detections: 633 / 1139 frames (55.6%)
- Player detections: 4556 total (≈4.0 per frame)
- Bounce events: 4 (all classified OUT — likely incorrect due to bad homography)
- Output file: `outputs/Video_Project_4_tracked.avi` (152 MB)

11.3 Validation Plan

- Collect 3–5 additional clips covering: single court (clean lines), multiple courts, partial visibility, different camera angles.
- Manually annotate ball positions and bounce events on selected frames for quantitative evaluation.
- Report: detection rate, tracking continuity, homography reprojection error, bounce classification accuracy.

12 Environment and Reproduction

12.1 Environment

- OS: Windows 10/11
- Python 3.13, PyTorch 2.6+
- Key packages: `ultralytics` (YOLOv8), `opencv-python`, `numpy`, `matplotlib`, `scipy`, `tqdm`, `pyyaml`
- Optional: `deep-sort-realtime` (for DeepSORT tracker)
- Environment variable: `TORCH_FORCE_NO_WEIGHTS_ONLY_LOAD=1`

12.2 Installation

```
pip install ultralytics opencv-python numpy matplotlib scipy tqdm pyyaml
pip install deep-sort-realtime  # optional, for DeepSORT
```

12.3 Running the Pipeline

```
cd tennis-pickleball-tracker
set TORCH_FORCE_NO_WEIGHTS_ONLY_LOAD=1
python run_pipeline.py
```

12.4 Inspecting a Single Frame

```
python test_fix.py
# Output: outputs/test_results/annotated_frame_v2.jpg
```

13 Conclusion

The Tennis/Pickleball Single-Camera Tracking System successfully implements an end-to-end pipeline from video input to annotated output. Ball detection achieves 55.6% frame-level detection rate, player filtering correctly identifies 4 on-court players per frame, and the tracking module produces continuous trajectories.

The primary remaining challenge is the **incorrect homography** caused by multi-court line interference (Section 6). This is the highest-priority fix because it blocks correct 3D reconstruction, bounce analysis, in/out classification, and mini-map rendering. The proposed fix (robust corner selection + reprojection sanity check + temporal stabilisation) is well-defined and assigned to Pham Hoang An.

With the 5-member task allocation defined in Section 8, the team is positioned to deliver a stable, evaluated system in the upcoming milestones.