

PitchTrack: the coach report, and the video behind it

Kam Labs, Jos, Nigeria · OpenCV AI Competition 2026, powered by AWS · Overall Awards track · 26 October 2026

1. Problem

Below the very top of African football, clubs play without match analysis. Plateau United are in the Nigeria Premier Football League (NPFL), the country's top flight, and like most clubs here they have no camera crew and no data provider. The tools that exist elsewhere are priced in dollars per pitch per month, and they assume a broadcast feed or a gantry the stadium doesn't have. So the post-match conversation runs on memory and a phone clip.

PitchTrack gives a club a **coach report after each home game it films**, from a fixed rig of three 4K lenses on the stand with no operator. It shows territory, team shape, who pushed up in each half, possession and pressing estimates, and the official's event record. A report only changes what a coach does if they can go and *see* what it describes, though. A three-lens panorama isn't something a coach can scrub through to find "the minute Kano pushed us back". This entry closes that gap with a **virtual camera**: OpenCV 5 on AWS turns any moment the report points at into a broadcast-style follow shot cut from the same fixed rig.

2. Users, and what they told us

- **Club coaching staff.** Plateau United's analyst and head coach have had a report for each of the last three home games (20 Sep, 11 Oct, 18 Oct 2026). These are live at kamhq.com/plateau/. PitchTrack is also on pilot with four academies.
 - **Their feedback changed the product.** In a review call on 20 Oct, the analyst could not tell which end each team attacked in each half. We added a half-by-half section with the attacking direction read off the kick-off frames. They read "no tackles" as zero tackles, so absences in the official's record are now labelled "not recorded". And they asked what the man-of-the-match pick meant, so it is now labelled a data pick, not a panel's verdict.
 - **What the report already asks for.** Each report has a section titled "Worth a look on the video". The data flags the moments, and the coach has to see them before trusting them. The virtual camera is that step.
 - **Leagues.** Kam Labs is technical partner to a national community tournament launching in November 2026. There, a fixed rig with no operator is the only coverage that scales to hundreds of matches.
-

3. The coach report (what the club receives)

From the 11 October match, Plateau United 4-0 Kano Pillars, which is also the source of this entry's footage:

- **Two labelled sources.** "Recorded by match official" is our Matchday app, filled in live. The score and all four scorers agree with the published result. "Measured by cameras" is the rig. Every section says which source it uses.
- **Territory and zone control.** Each 4 m square is tinted toward whichever side spent more of *its own* time there, so the side the cameras happen to pick up more often gets no head start. Plateau were the more present side in 128 squares, Kano in 122.
- **Half by half: who pushed up.** In the first half the sides were level (49% vs 48% of time in the other's half). In the second half Kano Pillars pushed Plateau back: 63% vs 49%, a 15-point gap with a 95% interval of 4–26. That is in a game Plateau won 4-0, the kind of finding a coach needs to watch before acting on.
- **Possession and pressing (estimates).** Plateau had about 58% of the ball (95% interval 54–62): 65% in the first half and 45% in the second. Pressing was level, at 12% and 13% of nearby players closing down. Both come from the rig's own view of the ball and are labelled as estimates.
- **Honest where it can't settle.** Position along the pitch and team width are reported as "not settled". A minute-by-minute test finds a gap, but no second camera view confirms it, so the page says so instead of naming a winner.

The report is built from player positions with **no player identity**. It measures the team, so the tracker's identity breaks don't touch it. In a five-minute test, a name lasted about 18 s. Names on the page come only from the official's event record.

4. Architecture

Four stages. The first three existed before this competition. The virtual camera's AWS service, and its link to the report, were built for this entry.

1. **Capture (rig).** Three lenses at 3840×2160 capture at a variable 17–18 fps, recorded as H.264 `.mkv` on a Pi 5. A JSON manifest stores each lens's start offset (`offset_ms`) and its pitch homography. The homography is fitted to the painted lines on a median frame with no players in it.
2. **Tracking (laptop GPU).** YOLO detection plus ByteTrack per lens produces `tracking_log.csv` : one row per player per tick, in pitch metres, on a shared session clock. For this match it holds 103,142 ticks over 6.9 h of compute.
3. **Coach report (laptop).** Kit split per lens, then territory, shape, half-by-half, possession and pressing estimates, each gated by a minute-by-minute test. The result is a static HTML report published to the club.
4. **Virtual camera (AWS, OpenCV 5).** `ptz.py` reads the three videos and the tracking log and renders a 1920×1080 follow shot at 25 fps of any window of the match. `server.py` exposes it as a job

queue behind nginx on EC2. The endpoint shows a 3-minute render around Plateau's second goal, and a form lets anyone render a 10–30 s window live.

The architecture diagram is on the endpoint's home page and in the submission images.

5. OpenCV 5 implementation

Everything that touches pixels in stage 3 is OpenCV 5.0.0 (`opencv-python-headless==5.0.0.93`), running unchanged on Windows and on the Ubuntu EC2 instance.

5.1 Time: three variable-rate decoders on one clock

The rig's container declares 20 fps, but the true rate varies with USB load. Frames are therefore located by time, never by index. Each lens has a `cv2.VideoCapture` (FFmpeg backend) that we walk with `grab()` , and `CAP_PROP_POS_MSEC` is read *after* each grab. Session time maps to local time through `offset_ms` . The output timeline is resampled onto a fixed 25 fps by advancing each decoder to the first frame at or after the wanted instant. Only the lens on air calls `retrieve()` ; the others only `grab()` , which keeps decode cost near one stream. For the cloud clip, each lens was stream-copied from a keyframe, and its `offset_ms` was shifted by that keyframe's exact presentation time, so no re-encode was needed and no drift was introduced.

5.2 Where to aim: a density peak, not a centroid

The obvious aim point, the mean position of every tracked player, follows *which lens happened to see how many people*, because per-lens detection counts swing from 0 to 19 between ticks. We aim instead at the point with the most players within 8 m. That is a Gaussian-weighted density on a 2 m grid, followed by a weighted-mean refinement off the argmax. Three gates keep a bad tick from moving the camera: at least 12 detections, a peak score of at least 2.5, and no implied movement above 22 m/s. Over the full match those gates kept 61,528 of 102,290 ticks. Movement is critically damped, with a 3 m dead zone and a speed cap.

5.3 Which lens, how much zoom, which way up

- The aim point is projected into each lens through its homography. The camera cuts to the lens with the greatest margin from the frame edge, with hysteresis (250 px margin, 0.6 s hold) so it does not flicker at the seams.
- Crop width comes from projecting a 30 m span of pitch at the aim point, clamped between 1280 and 3400 source pixels. Far-side play gets a tighter crop than near-side play.
- Roll is measured from the **vertical vanishing point**, not from the touchline, which slopes mostly because of perspective. Since $H \propto K[r_1 \ r_2 \ t]$, we take $r_3 = r_1 \times r_2$, and the vertical vanishing point is $K \cdot r_3$, with K built from the lens's 51.1° horizontal field of view. On an earlier ground with a tilted mount, this code measured +6.0° and –6.6° on the outer lenses, and the levelled output was checked against a goalpost. **For this clip levelling is off.** The venue's calibration was fitted to the paint

without recording the lens field of view, so K is unknown, and the code applies no rotation rather than guessing one.

5.4 Resampling

The crop's axis-aligned footprint is downscaled first with `cv2.resize(INTER_AREA)`, because `warpAffine` has no area filter and aliases grass texture into shimmer. A 1:1 `warpAffine` then cuts the final rotated window. Output frames are piped to `ffmpeg /libx264`.

5.5 Set pieces: a director, not a wider lens

Reviewing the goal, our own team spotted that the first version **never showed the corner being taken, or the shot from outside the box**. The camera held on the crowd in the goalmouth. The data showed why. The taker stood on the near corner arc at (100.0, -1.2), **33–37 m from the aim point**, which the 30 m frame can never contain. The shooter also stood apart from the crowd.

- **Making the frame contain everyone failed.** We tried widening the frame to hold every final-third player within 38 m of the aim. It zoomed to the whole sensor on 79% of frames, and the picture was mostly sky. The near flag is close to the right-hand lens and the box is 50–60 m away, so no 16:9 crop holds both at a watchable size.
- **What shipped: a capped widening plus a corner director.** In the final third the frame may widen by up to 1.6×, to take in players within 25 m of the aim. That brings the 18-yard line into shot. For corners, we render after the match, so the delivery time is known in advance. It is the moment the player standing still on a corner arc moves off it, with at least 6 players in that penalty area. The camera stays on the box while the corner is set, **cuts** to the taker 4 s before the delivery, and cuts back to the box 0.8 s after it. A figure standing 3.6 m behind the goal line for the whole sequence (a ball boy or photographer) is excluded by an on-pitch test.
- For the Kano Pillars goal the director put the delivery at 2,783.8 s. The official logged "corner taken" at 2,782 s, the taker's speed went from 0.7 to 5.3 m/s, and the box's median speed doubled.

6. AWS deployment

Component	Choice	Why
Compute	EC2 <code>m7i-flex.large</code> (2 vCPU, 8 GB), eu-north-1	Three 4K decoders need memory headroom. It is the largest free-plan instance type, and our project's Region is fixed to eu-north-1.
OS / image	Ubuntu 24.04, Canonical AMI via SSM parameter, 30 GB gp3	Reproducible AMI lookup.
Service	Python stdlib HTTP server under systemd, behind nginx	No framework dependency. nginx serves the MP4s directly.
Security	Port 80 open. SSH only from one admin IP, with an ED25519 key. IMDSv2 required.	Minimum exposure for a public demo.

Abuse limits	Window bounded to the clip, duration 10–30 s, at most 5 queued jobs, one render at a time, 1 KB request bodies	A public render button must not be a free CPU farm.
Observability	<code>/api/health</code> (OpenCV version, queue length). Each job's timing and log are appended to <code>jobs.jsonl</code> . systemd journal.	

The detector does not run on AWS in this entry. It needs a GPU, and the free plan does not offer GPU instances. The boundary is deliberate: the tracker's output is a small CSV in metres, and everything downstream of it is CPU-only OpenCV work, well suited to cheap cloud instances.

7. Evaluation

7.1 Does the camera point at the ball?

The virtual camera never sees the ball; it aims at players. To test whether that is good enough, we used an independent ball path. Our rig ball extractor, built for possession estimates, found the ball on 4,899 ticks of this match, mostly on the centre lens. For each of the 4,889 ticks where every rule had an aim point, we measured the distance from aim point to ball (`eval/aim_vs_ball.py`):

Aim rule	Median distance	90th percentile	Ball inside the 30 m frame (≤ 15 m)
Density peak (this entry)	14.9 m	34.2 m	50.2%
Centroid of all players	17.3 m	30.2 m	39.3%
Fixed shot at the centre spot	22.0 m	37.9 m	28.3%

These figures measure the base aim rule; the corner director (5.5) overrides it for about 5 s per corner and is evaluated separately in 7.3. The density peak puts the ball inside the frame **28% more often** than the centroid, and it beats the fixed shot by 1.8×. **Its tail is worse than the centroid's** (90th percentile 34.2 m against 30.2 m). When it commits to the wrong cluster, such as a crowded box while the ball is switched to the far wing, it is further wrong than an average would be. That is the main failure mode, discussed below.

This is aim-point distance, not what the viewer finally sees. The rendered camera is damped and its crop is wider than 30 m on near-side play, so the share of frames with the ball visible is higher than these figures. The ball path is itself a model output: it was spot-checked by eye (about 88 of 96 audit crops real) and is biased towards the centre lens.

7.2 Ground truth for the clip

The official match record (the Matchday app, entered live by a match official) logs Plateau's second goal, Ekundayo Ojo (#10) from a corner in first-half stoppage time (45+3'). On the session clock that is about 2,793 s. The render covering 2,740–2,820 s shows the attack in Kano's penalty area and then the

celebration at the bench in that window. The camera cut to the right-hand lens for the goalmouth on its own, cut to the corner taker at the near flag about 4 s before the delivery, and cut back to the box for the finish.

7.3 Corner detection against the official record

We ran the corner director's detector over the whole match and compared it with the official's record, which logs Plateau's corners only:

- **Recall: 5 of 6 Plateau corners found**, each within about 11 s of the official's tap. The one missed was logged at 196 s, during the opening minutes, when entry was disrupted.
- **13 further detections the record cannot confirm.** Some fall at Kano's end, and Kano's corners were not recorded. First-half entry is known to be patchy, so some of the rest may be real corners too. Four come within 48 s at Plateau's end (1,122–1,170 s), which looks more like a scramble near the flag than four corners. We have not labelled these by eye, so precision is **unmeasured**. When the director is wrong, the cost is a 5-second cut to a player standing near a corner flag.

7.4 Speed

Where	Render	Throughput
Laptop (Windows, OpenCV 5)	20 s and 80 s windows	45–53 fps (2–2.1× real time)
EC2 m7i-flex.large (2 vCPU)	3 min (2640–2820 s), 4,500 output frames	13.6 fps (0.54× real time): 5 min 39 s wall clock, 196% CPU, 478 MB peak memory, with the corner director and final-third widening on

The render is CPU-bound on decoding the three 4K streams. Only the lens on air is fully decoded, but every lens must be `grab()`bed to stay in sync. A full 90-minute match on this instance would take about 2 h 40 min; on a 16-vCPU compute instance, or rendering halves in parallel, it would take well under real time. The free plan limits us to 2 vCPUs.

7.5 Failure cases we observed

- **Long switches of play.** The players arrive after the ball does, so the shot lags and sometimes holds on the crowded area the ball has just left. This shows in the 90th-percentile tail above.
- **Rejected ticks.** 31% of ticks are rejected by the jump gate and 9% by the thin gate (fewer than 12 detections, typically when play sits in an outer lens's narrow strip). In both cases the camera holds its last good aim. That is safe, but it looks static.
- **Far-side softness.** From a low stand, far-side play covers only a thin band of each 4K frame, so a tight shot there is an upscale and looks soft.
- **Upstream tracking.** Track identities break often (in a five-minute test an identity lasted about 18 s). The virtual camera only needs positions, not identities, so it is robust to this. Per-player features are not.

8. Limitations

- It follows players, not the ball. Fusing the ball path into the aim is the obvious next step, and the evaluation above is the measure it will be judged by.
- It runs after the match. The laptop renders faster than real time, but capture-to-cloud upload from a stadium in Nigeria is the real constraint: about 1 MB/s from our office, so a full match takes hours.
- Detection and the coach report run on a local GPU laptop, not on AWS, in this entry. Only the video stage, the part that must serve the club on demand, is in the cloud.
- The cloud clip covers one 200-second window (around the second goal), because uploading 4K footage from Jos runs at about 1 MB/s. Rendering any moment the report flags across the full match needs the whole match uploaded, about 66 GB.
- The calibration was fitted to the painted lines once per venue. A rig nudged mid-match is not detected.

9. Responsible use

- **Consent and rights.** The footage is of a professional league match in a public stadium. It was recorded by our own rig with the home club's agreement under the PitchTrack pilot. It is not redistributed in the code repository.
- **No biometrics, no identification.** The pipeline detects "a person in a football kit" and tracks positions. It does no face recognition, and neither the team report nor the virtual camera uses player identity. Player names in the report come only from the match official's event record, never from the cameras.
- **The club sees it first.** Reports go to the club's analyst and coach, who decide what reaches the squad. As of the 20 Oct review, they were still checking their own reading before showing the players.
- **Surveillance risk.** Pointed at a crowd, this is the same technology as a tracking camera. On-pitch gating, which ignores detections more than 3 m outside the lines, keeps the camera on the field of play. The public demo is limited to a fixed clip of a match, not a live feed.
- **Honest reporting.** Every number in this report comes from a script in the repository, or is labelled as measured by eye. The evaluation reports the failure tail, not only the median.

10. Reproducing

The repository's `README.md` has pinned dependencies, the local run, the EC2 deployment steps (with the exact systemd unit and nginx site in `deploy/`), and `curl` tests for each endpoint. The evaluation is

`python eval/aim_vs_ball.py --log ... --ball ... --out eval/aim_vs_ball.json` , and its output is committed.