ARGUS: User Manual

AI-based virtual camera tracking system for coarse alignment of mobile Free Space Optical Communication (FSOC) terminals. Smart India Hackathon problem statement SIH26169, Department of Space / ISRO Space Applications Centre.

1. What the application does

The application simulates the coarse alignment stage of an FSOC link entirely in software.

- It draws a scene (the "screen", at least 2000 x 2000 pixels) with a moving optical beacon.
- A virtual pan-tilt camera looks at a 640 x 480 pixel window of that scene (4 x 3 degrees) and can turn at a limited speed, like a real motorised mount.
- Disturbances can be added: fog, haze, rain, low light, atmospheric turbulence, platform motion, camera vibration, and sensor noise.
- The tracker detects the beacon, measures its centre to a fraction of a pixel, predicts where it is going, and steers the camera window to keep it centred.
- Every run writes a per-frame log, a summary and an automatic performance report into one folder named `FSOC_<sim|video>_<name>_seed<N>_<date-time>` (section 7).
- For Benchmark 2, an `.mp4` video can be opened in place of the simulated scene.

2. Installation

2.1 Standalone executable (no Python needed)

Archives are built for every desktop platform in use:

Platform	Archive	Run
Windows 10 or 11, 64-bit	`ARGUS-windows-x64.zip`	`ARGUS.exe` (window); `ARGUS-cli.exe` for the command line
Linux, 64-bit (Ubuntu 22.04 or newer, Debian 12, BOSS, RHEL 9, Fedora)	`ARGUS-linux-x64.tar.gz`	`./ARGUS`
macOS 13 or newer, Intel	`ARGUS-macos-intel.zip`	`ARGUS`
macOS 13 or newer, Apple silicon	`ARGUS-macos-arm64.zip`	`ARGUS`

A demonstration video (`ARGUS-demo.mp4`, about four minutes, composed from the engine's own frames) and a 10 to 15 minute live demo script (`record/DEMO_SCRIPT.md`) accompany the builds.

1. Download the archive for your platform from the downloads page, or from the repository's Releases page (`latest` is the newest build; repository access needed).

2. Extract it anywhere the user can write to (Desktop, Documents, home folder). Do not run it from inside the archive. The first launch of a new build can take longer than later ones while the operating system checks the files and the plotting library builds its font cache; the window opens first and that work finishes in the background.
3. Run the executable named above. The first start takes a few seconds.

Platform notes:

- Windows: SmartScreen may show "Windows protected your PC" because the executable is not code-signed. Choose More info, then Run anyway. No administrator rights are needed.
- Linux: the archive carries Qt and all Python libraries. The system needs the usual X11 or Wayland client libraries, present on every desktop install. On a minimal server image install `libxcb-cursor0 libxkbcommon-x11-0 libegl1` (Debian and Ubuntu names). If the file is not executable after extraction, run `chmod +x ARGUS`.
- macOS: Gatekeeper may say the application cannot be verified. Open System Settings, Privacy and Security, and choose Open Anyway, or run `xattr -dr com.apple.quarantine ARGUS` on the extracted folder once.

Results are written to a `results` folder next to the executable. Every archive passed the test suite and a smoke test of the packaged executable on its own platform before it was published (Windows and Linux on the GitHub build workflow, macOS on the team's Mac).

2.3 Web application

The same program is also served as a web application, so no installation at all is needed on any platform: the project site opens it directly in a current browser (Chrome, Edge, Firefox or Safari, on Windows, Linux, macOS, or a tablet). The web page and the desktop window are the same interface: the same toolbar (Start, Pause, Step, Stop, Speed, Open video, Truth CSV, Simulator, Save scenario, Screenshot, Results, Manual, About), the same parameter panel with every field, the same six tiles, views, four plots, telemetry column and end-of-run summary, and the same keyboard keys. Both are generated from one definition in the code (`fsoc_tracker/ui_shared.py`) and run the same engine, so a given scenario and seed give the same numbers and the same report.

What differs follows from running on a server, not from the program:

- The views in the browser are sent as snapshots about 15 times a second; the log, the plots and the report still cover every frame.
- One run executes at a time per server; a second visitor watches the run in progress.
- Files are downloaded from the browser instead of opened in a folder (Results lists them).
- The page also offers the desktop application for download (the Desktop app button).

2.2 From source

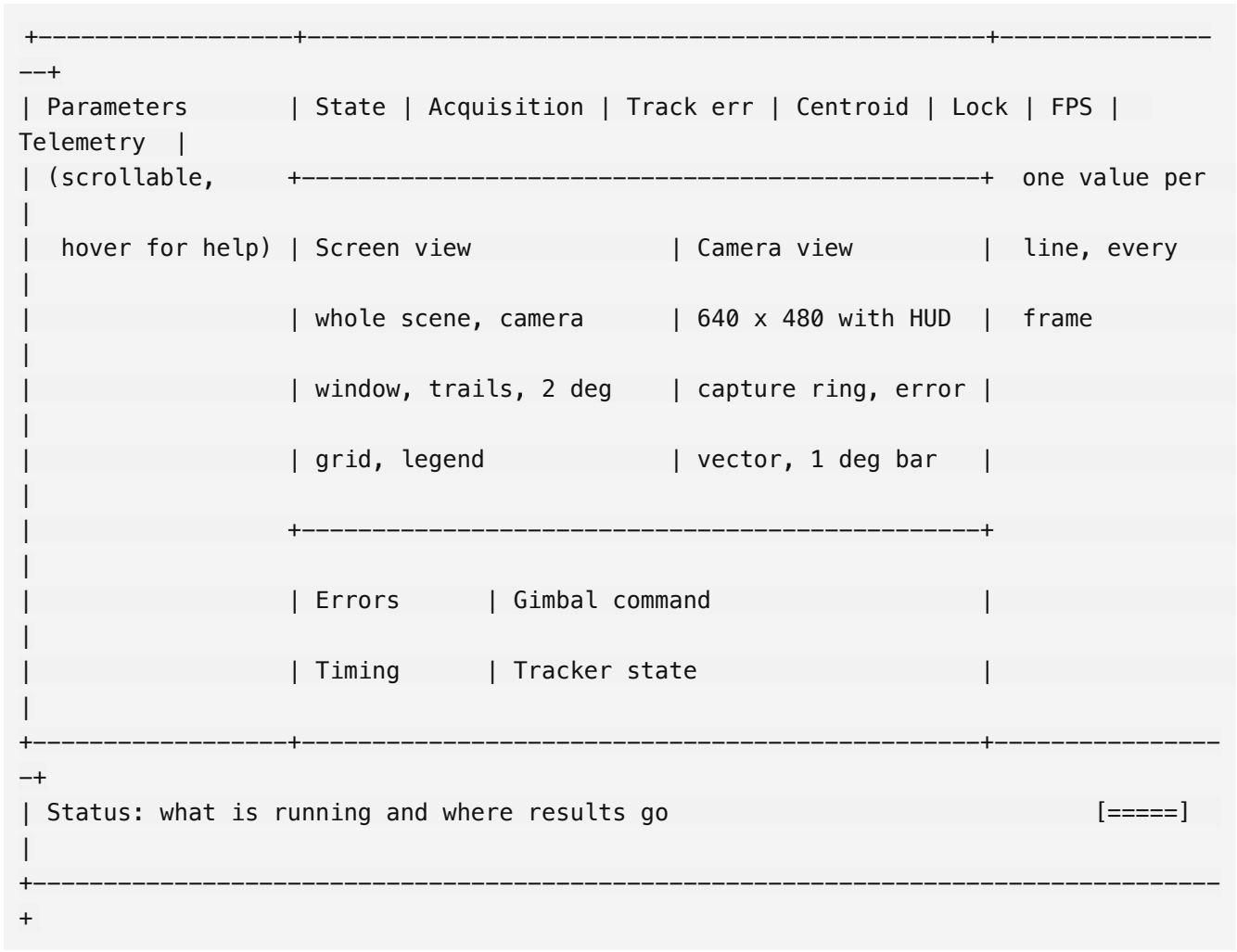
Requires Python 3.11 or 3.12.

```
 git clone <repository>
cd SIH169
python -m venv .venv
```

```
.venv/bin/pip install -e ".[dev]"        # Windows: .venv\Scripts\pip install -e ".
[dev]"
.venv/bin/fsoc-tracker-gui               # start the desktop application
```

Optional, to retrain the AI detector: `pip install -e ".[train]"` then `python training/train_heatmap.py`.

3. The main window



Toolbar, left to right

Control	Action
Scenario	Pick a ready-made test case from `configs/scenarios`. "Custom" keeps whatever is in the panel. A picked scenario runs as written: its seed and paths are kept and "New seed each run" is switched off.
Start / Pause / Step / Stop	Run control. Space starts or pauses, N steps one frame while paused, Esc stops.
Speed	0.25x to 4x real time, or Max speed (shows the true processing rate).
Open video (Benchmark 2)	Choose an `.mp4`; the simulator is bypassed and the video frames become the scene. Ctrl+O.

Control	Action
Truth CSV	In video mode: load the evaluators' beacon positions for the video (section 5), so errors and true lock are computed.
Use simulator	Return to the simulated scene.
Save scenario	Save the panel as a `.yaml`; it appears in the Scenario list. Ctrl+S.
Screenshot	Save a PNG of the window into `results/`. Ctrl+P.
Results folder	Open `results/`.
User manual	Open this manual.
Help	Legend and output summary.

Tiles (live specification check)

Six tiles above the pictures update every frame and turn green when the problem statement specification is met, amber when it is not: tracker state, acquisition time (2 s or less), mean tracking error since acquisition (10 px or less), mean centroiding error, lock retention (target loss under 5 percent), and processing FPS (20 or more). In video mode the two error tiles read "n/a" unless a ground-truth file is loaded (section 5).

Screen view (left picture)

The whole scene, downscaled, with a 2 degree grid. Cyan rectangle: the camera window and its centre. Orange circle: the true beacon (simulator only). Green cross: the tracker's estimate. Grey circles: other targets. Every target carries its name; the designated one is marked "(designated)" in the signal colour, the others are muted. Before a run the view shows a preview of the scene at t = 0 with every target named; click a target to make it the designated one. Faint trails: where the beacon and the window have been. The legend is printed along the bottom edge.

Camera view (right picture)

What the camera window sees, contrast-stretched for display. Cyan cross: boresight. Dashed ring: the capture radius, green when locked. Line from the centre: the pointing error vector. Green box: this frame's detection. Orange dot: the prediction, with a fainter ring for the estimator's uncertainty. Green trail: recent estimates. A 1 degree scale bar sits bottom right. The banner shows the state, tracking error in pixels and arcseconds, confidence and which detector produced the measurement; the bottom line shows pan, tilt, the commanded rates, saturation and processing time.

Plots

- Errors: tracking error (window centre to beacon, cyan) and centroiding error (measured centre to true centre, orange). Dashed line: the 10 pixel specification.
- Gimbal command: pan and tilt rate commands with the rate limit dashed.
- Processing time per frame with the 20 FPS budget (50 ms) dashed.
- Tracker state: SEARCH, VERIFY, TRACK, COAST, REACQUIRE.

Telemetry column

Every internal quantity for the current frame: state, detector, candidates, confidence, SNR, pose, commands, detection, estimate, velocity, uncertainty, motion model probabilities, truth, errors and timing. Before a run it shows a short how-to.

4. Parameters

Every row of the problem statement parameter table has a control. Values outside the suggested range are allowed where the table says "user-defined". Every numeric input is clamped to its accepted range in the engine, for both applications, and the scenario check (section 4.1) says what a value means.

Run

- Name, Seed: the seed makes a run exactly repeatable.
- Duration (s): length of a simulator run. Ignored for video input (the whole video runs).
- Extra targets: number of additional beacons (decoys) added with random paths.
- Identical look: the generated extra targets copy target 1's shape, size and brightness. The tracker is then told the designated one's start position as well (see Target).

Screen (rows 1 to 2)

- Width, Height: scene size in pixels. Default 2000 x 2000.
- Colour camera: renders a three-channel colour frame (PS row 2, optional); the tracker always works on luminance, so results are identical.
- Background: starfield, terrain, gradient or flat. Sky level and star density.

Camera (rows 3 to 6, 13 to 15)

- Width, Height: camera window in pixels. Default 640 x 480.
- FOV width, height: degrees. Default 4 x 3. Degrees per pixel is derived from these.
- Update rate: frames per second. Default 30.
- Max pan, Max tilt: degrees per second. Default 5.
- Max accel, Command latency: gimbal realism.
- Hard mode: the tracker sees only the pixels inside the window and must search.

Target (rows 7 to 12)

- Target: pick the target this section shows and edits; `*` marks the designated one.
- Designated: tick it on the target the tracker must follow (PS: "a designated moving target"). The report scores this one. One target is always designated, so the box cannot be unticked: tick it on another target instead. Before a run, clicking a target on the preview does the same. How the tracker finds it is automatic: by its configured shape, size and brightness, and when another target looks the same, by its start position as well (as an operator or a GPS cue would). A scenario file can force a mode with `designation: appearance | start | cue`.
- Name: shown on the views, in the telemetry and in the report. Empty means "Target N".

- Shape: square (default), circle, gaussian, cross, ring, diamond, custom. Width and Height in pixels, set separately (default 10 x 10; the PS range is 5-20 x 5-20). A square with unequal sides is a rectangle, a circle an ellipse. Peak intensity.
- Custom shape (0/1 rows): for shape `custom`, rows of 0 and 1 separated by `;`, stretched to width x height. `010;111;010` is a plus. A PNG path also works.
- Motion: line, circular, figure8, random, spiral, sinusoidal, waypoints, static. For `waypoints`, the Waypoints field takes screen-pixel points as `x,y; x,y; ...`; the beacon follows them at Speed and loops (the PS row 12 user-defined path).
- Speed, Radius, Period, Heading: path parameters. Start: random, centre, or a typed `x,y` in screen pixels (for example `400,1500`).
- Blink: optional intensity modulation in Hz. 0 is steady.
- Beacon code and Code rate (beyond the PS): a pattern of 0 and 1 (3 to 32 bits, for example `10110`) that the beacon blinks at the code rate (default 10 bits/s, at most half the update rate); a 0 bit dims it to 70%. When the designated target has a code, the tracker accepts a spot only after it has blinked the code for two cycles (1 s for 5 bits at 10 bits/s), turns down spots that do not, and keeps checking while it tracks, so look-alikes are told apart with no start position or click. The end-of-run summary and the report say how many spots it turned down. Codes that differ only in where the cycle starts are the same code; the scenario check warns when a look-alike blinks it. Leave it empty for a faint beacon: the dim bits make a beacon at the detection limit harder to find. Empty is a steady beacon (the PS default).

Disturbances (rows 21 to 25)

- Salt and pepper in percent of pixels (10 = 10 percent; accepted 0 to 50; scenario files store it as a fraction, 0.10), Gaussian sigma (up to 20), Poisson.
- Camera jitter: pixels per frame, up to 20.
- Atmosphere preset: clear, haze, fog, rain, lowlight. Selecting a preset fills contrast, brightness, blur and turbulence; each can then be edited.
- Platform motion: none, linear, circular, random, spiral, figure8; speed in pixels per frame (up to 20) and period.
- Camera effects beyond the PS table (the PS lists disturbances "... noise, etc."), both off by default: **Exposure gain** (1 = normal; below 1 under-exposed, above 1 over-exposed with the bright beacon clipped at white) and **Frame loss** (percent of frames the camera link loses; a lost frame reaches the tracker with no picture, so it coasts). The scenario check notes both.

Changing disturbances during a run. While a run is going, the Disturbances section stays editable and every other section is locked (the other settings are fixed for the run). Change any disturbance, for example switch the atmosphere to fog or raise the camera jitter, and it reaches the camera feed on the next frame, without restarting. The status line says what changed and when; the error and gimbal plots get a dotted line at that moment; the tracking error, centroid error and lock tiles start again from the change ("since 12.3 s") so the effect of the new conditions shows at once. The report adds a page with one row per setting (segment) and its own figures. The platform never jumps when its sway is switched on, changed or off: the picture moves on from where it was and settles into the new pattern within 1.25 times the larger peak speed. Video runs (Benchmark 2) have no live disturbances; the video holds its own.

Tracker

- Tracker algorithm: argus (this project's tracker, the default) or baseline, a deliberately simple brightest-spot tracker with proportional control kept only to compare against (`record/BASELINE_COMPARISON.md`). The scenario check says so when baseline is chosen.
- Detector: hybrid (classical first, AI fills gaps), classical, cnn.
- Use picture-shift estimate: phase correlation as a vibration hint.
- Controller gains, deadband, capture radius, estimator lag, minimum confidence to acquire. Defaults: kp 5, kd 0, ki 0.8. The derivative gain is 0 on purpose: the error reaches the controller one frame late and in whole pixels, and a derivative on it (0.3 before) drove a limit cycle of about +/-10 px, so a still beacon was never settled. With kd 0 a still beacon is held within 4 px.
- Faint path: when nothing reaches the acquisition confidence, weak detections (threshold `faint_threshold_k`, default 3 sigma) are linked across frames and a motion-consistent chain with mean SNR above `faint_snr_min` (default 3.5) is promoted. Used automatically for dim beacons in low light; a static dim beacon is not covered by this path. While a faint track is active, a sub-pixel fit that jumps more than 3 px or balloons in width is discarded, much wider blobs are not followed, and the AI detector is not used.
- Hard mode (Camera section) restricts the tracker to the window; SEARCH then flies an expanding square spiral of window-sized cells at the rate limit. A full sweep of a 2000 px screen at 5 deg/s takes about 12 s, so acquisition in hard mode is 2.83 to 11.97 s (measured) depending on where the beacon is.

4.1 Scenario check

Below the Run section, both applications show notes on the current values. They also appear in the status bar at Start, in the end-of-run dialog, on page 1 of the report and in the summary. Values inside the PS envelope give no note.

Note	Meaning
Corrected	A value was outside the accepted range and was clamped, for example salt and pepper 13 (a fraction) becomes 0.5.
Beyond the PS	A value is outside the PS table, and the row is named: screen below 2000 x 2000 (row 1), update rate below 30 Hz (row 5) or 20 Hz (row 15), pan or tilt outside 5 to 10 deg/s (rows 13, 14), size outside 5-20 x 5-20 (row 10), custom shape without a mask, salt and pepper above about 10% (row 21), Gaussian sigma above 20 (row 22), jitter above 20 px/frame (row 23), platform above 20 px/frame (row 25), turbulence above 0.6, contrast below 0.4. The PS targets are not promised for it.
Near the limit	The designated beacon moves above 70 percent of the camera turn rate. It can be done, with little margin.
Cannot be met	A physical limit: the designated beacon moves faster than the camera turns (800 px/s at 5 deg/s and the default FOV); jitter plus platform motion above the camera's turn per frame (26.7 px/frame at the defaults); salt and pepper at 50%; designation `cue` without a point; designation `start` with a video; a ground-truth file that is not found; other targets that look the same as the designated one in appearance mode.

5. Running Benchmark 2 (video input)

1. Click "Open video (Benchmark 2)" and choose the `.mp4` file. The application reads the file's real facts (displayed size with any rotation tag applied, average frame rate with a variable-rate warning, exact frame count, length) and calibrates the settings to them: the screen becomes the video's size and the update rate its frame rate. Disturbance settings and the target's motion fields are locked, because the video already contains them. The application then scans the first 30 frames with its detector and lists every beacon it finds in a "Beacon" picker that appears in the Run section, under Extra targets, as "Target 1: at x,y, w x h px, peak p", "Target 2: ...", brightest first. The preview labels each one. Choose the beacon you want followed in that picker: the tracker is cued to its first-frame position, Designation switches to `cue`, the Target section shows that target and the status bar names it. Clicking the picture does nothing in video mode. The run follows and scores only the chosen beacon (PS row 8: one designated target, the others are decoys). If the scan finds none, the picker reads "none found"; describe the beacon in the Target section (shape, width, height, intensity). Closing the video hides the picker and restores the simulator's targets.
2. Set the camera window size, FOV and rate limits if the graders specify them; these are not in the file. Degree readouts depend on the FOV you set.
3. Click Start. The video frames are used as the scene; nothing is drawn by the simulator.
4. When the run ends, `frames.csv` contains the measured beacon centre for every frame (`det_x`, `det_y`) and `report.pdf` contains acquisition time, re-acquisition time, lock retention rate and FPS. Without a ground-truth file, tracking and centroiding error read "n/a" and lock is judged from the tracker's own estimate, which can overstate it.

From the command line: `fsoc-tracker video path/to/file.mp4`

Ground-truth file (optional)

If the evaluators give the true beacon positions, put them in a CSV: frame (or `t` in seconds), x, y in video pixels. Header names are matched loosely (`frame` / `idx`, `t` / `time`, `x` / `true_x` / `cx`, `y` / `true_y` / `cy`); without a header the columns are frame, x, y. A row with a blank or NaN position means the beacon is not in the frame. A frame not listed is filled in when it lies in a gap of at most a third of a second between two visible rows (a file sampled every few frames; it is used to judge lock, and no error is scored on it); otherwise it counts as "beacon not visible". With it, tracking error, centroiding error, RMSE and a true lock retention are computed against their positions.

How	Where
Command line	`fsoc-tracker video clip.mp4 --truth truth.csv`
Beside the video	a file named `<video>_truth.csv` (or `<video>.csv`) next to the video is picked up by the command line and the desktop app
Desktop	toolbar button "Truth CSV" (enabled in video mode)
Web	toolbar button "Truth CSV" (uploads the file)
Scenario file	field `video_truth`

The report's source line names the truth file or says "no ground-truth file". Measured on a noisy_line clip rendered to `.mp4` with its truth: tracking error 4.85 px, centroiding error 0.188 px; without the file both read n/a.

6. Scenario files and the command line

A scenario is a YAML file with the same fields as the parameter panel. Example:

```
name: fog_circular
seed: 6
duration_s: 30
targets:
  - {name: Remote terminal, shape: square, size_px: 10, height_px: 10, motion:
circular, radius_px: 400, period_s: 15}
disturbance:
  atmosphere: fog
  gaussian_sigma: 8
```

Target fields: `name`, `shape`, `size_px` (width), `height_px` (0 = same as the width), `mask` (for `custom`), `intensity`, `motion` and its path fields, `start` (`random`, `centre` or `"x,y"`). Run fields for designation: `designated` (index of the target to follow, default 0), `designation` (`appearance`, `start` or `cue`) and `designation_cue` (`"x,y"` in screen pixels). `disturbance.salt_pepper_frac` is a fraction (0.10 = 10 percent).

Commands:

```
fsoc-tracker run --scenario configs/scenarios/fog_circular.yaml [--seed 3] [--
duration 20]
fsoc-tracker video path/to/file.mp4 [--truth truth.csv]
fsoc-tracker batch --scenario configs/scenarios/*.yaml --seeds 0-49
fsoc-tracker verify results/<run folder>   # recompute the metrics from frames.csv
and check summary.json
fsoc-tracker gui
```

`batch` runs every scenario over every seed and writes `envelope.md`, a table of the worst and mean results per scenario.

A scenario can also change its disturbances during the run with a `schedule`. Each entry gives a time and only the settings that change; an atmosphere preset fills contrast, brightness, blur and turbulence unless the entry sets them:

```
schedule:
  - {t_s: 10, disturbance: {atmosphere: fog}}
  - {t_s: 20, disturbance: {jitter_px: 10, platform_motion: linear,
platform_px_frame: 12}}
```

Changes made live in the application are saved in the run's `scenario.yaml` in the same form, at the frame they took effect on, so `fsoc-tracker run -s <that file>` repeats the run exactly.

Evaluator-supplied files

Benchmark Performance 1 gives you scenario parameters. Copy `configs/scenarios/TEMPLATE_evaluator.yaml`, which lists every field with the PS row it implements, fill in the given values, save it under `configs/scenarios/`, and it appears in the Scenario picker of both applications; or run it with `fsoc-tracker run -s <file>`. Fields the evaluators do not specify stay at the PS defaults.

Benchmark Performance 2 gives you `.mp4` files. No scenario file is needed: open the file in the desktop application, drop it on the web app, or run `fsoc-tracker video <file>`. The per-frame CSV then carries `det_x`, `det_y` (the measured centroids) for comparison with the evaluators' predefined values, and the report carries acquisition, re-acquisition, lock retention and FPS. If they give the true positions, load them as a ground-truth file (section 5) and the errors are computed against them.

7. Output files

Every run gets one label and one folder named after it:

```
results/FSOC_sim_<scenario>_seed<N>_<YYYYMMDD-HHMMSS>/      simulated run
results/FSOC_video_<file name>_<YYYYMMDD-HHMMSS>/           Benchmark 2 run on a video
```

for example `results/FSOC_sim_clear_line_seed7_20260922-101530/`. The four files inside carry the same label as a prefix, so a report copied out of its folder still says which scenario, seed and time it belongs to:

File	Contents
`<label>_frames.csv`	One row per frame: time, state, detection, estimate, camera pose, commands, truth, errors, processing time, and `segment` (0 until the first disturbance change, then 1, 2, ...).
`<label>_summary.json`	All metrics with their definitions and pass/fail against the specification; `designation` (followed target, index, mode, cue, all target names, ambiguous frames, redesignations, beacon code and the spots it turned down) and `checks` (the scenario check notes).; `segments` (the figures of each disturbance setting, when they changed during the run).
`<label>_report.pdf`	Specification check, the Targets and Followed lines, the scenario check notes and the metric table (continued on the next page when long); time series; paths, histograms and model probabilities. A run whose disturbances changed has one more page, a row per setting.
`<label>_scenario.yaml`	The exact parameters used, including any disturbance changes as `schedule`, so the run can be repeated.

With `--out <folder>` on the command line the folder is yours; the files inside are still labelled. Batch runs write `results/batch/<time>/<scenario>_seed<N>/` with labelled files and an `envelope.md` table. The web app names its downloads the same way.

Every figure in `<label>_summary.json` can be checked against the log: `fsoc-tracker verify <run folder>` rebuilds the metrics from `<label>_frames.csv` alone, with the same code that made them,

and lists any value that differs (exit code 1). The end-of-run summary names the command.

Handoff to fine pointing. The problem statement's coarse stage works "before fine pointing mechanism can take over". The state tile and the camera view say *handoff ready* once the tracker is locked and its estimate has stayed within the handoff radius (default 10 px, the PS row 17 limit) of the window centre for the hold time (default 1 s); both are in the Tracker section. The report gives the time it was first ready and the share of the run it held; the CSV has a `handoff` column.

8. Metric definitions

Metric	Definition
Acquisition time	First frame in TRACK with the tracked beacon within the capture radius (30 px) of the window centre. Spec: 2 s or less.
Tracking error	Distance from the true beacon to the window centre, over frames after acquisition. Spec: 10 px or less (mean).
Tracking error, vibration removed	The same with the per-frame camera vibration subtracted from the truth: the part a rate-limited gimbal can physically follow.
Centroiding error	Distance from the measured beacon centre to the true centre.
Tracked	Percentage of frames after acquisition in which the tracker held the beacon (state TRACK), whether or not the camera had it centred. A high tracked rate with a low lock rate means the camera, not the tracker, could not keep up.
Lock retention	Percentage of frames after acquisition in TRACK with the beacon inside the capture radius. Target loss is 100 minus this. Spec: loss under 5 percent.
Re-acquisition time	Time from losing lock to regaining it. A loss not regained by the end of the run counts with its length so far (also reported as lock lost at end). Spec: 1 s or less.
FPS (`fps_mean`)	Frames over processing time: 1000 divided by the mean processing time in ms. Spec: 20 or more.
`fps_inst_mean`	The mean of the per-frame rates 1 / processing time. Higher than `fps_mean` when frame times vary; for comparison only.
Handoff ready	Locked, with the estimate within the handoff radius (10 px) of the window centre for the hold time (1 s): coarse alignment stable enough for fine pointing to take over. Reported as the first time and the share held after it.
Frames lost	Share of frames the camera link lost (frame loss setting). A lost frame breaks lock by definition, so the report adds **lock retention on the frames received** beside the usual lock figure.
Segment	When the disturbances change during a run, each setting is a segment with its own tracking, vibration-removed and centroiding error, lock, tracked rate and FPS. The run's overall figures span all segments.

9. Troubleshooting

- A value changed after you typed it: it was outside the accepted range and was clamped. The scenario check shows it as "Corrected" with the old and new value.

- The beacon is never acquired: read the scenario check. "Cannot be met" notes name the setting that makes the run impossible.
- The camera keeps swinging around a still beacon: check that kd is 0. A derivative gain above 0 makes the camera oscillate by about +/-10 px.
- The window does not start on Linux: install `libxcb-cursor0` (Qt 6 requirement).
- Low frame rate: reduce the screen size, disable Poisson noise, or uncheck Real-time pacing to see the true processing speed.
- "Cannot open video": the file must be readable by OpenCV (H.264 `.mp4` is safest).
- The AI detector is not used: `models/beacon_heatmap.onnx` is missing (it is looked up in the working folder and in the package folder); the classical detector runs alone. In the standard scenarios the AI supplies 0 percent of measurements anyway: it only fills gaps. Retrain with `python training/train_heatmap.py`.