1 A new dataset: 9 real signals, multiple views each

Folder structure = object identity: each numbered folder holds 3–5 photos of one fixed signal, taken from different camera positions.



Each photo also carries in a json file: camera GPS, azimuth/pitch/roll, focal length, a bounding box around the signal, and a monocular depth estimate.

2 Inside one object folder: photos and JSON

Object 1: one physical traffic signal, photographed from four camera positions. Every photo has its own JSON file. The yellow box is the bounding_box stored in that file.

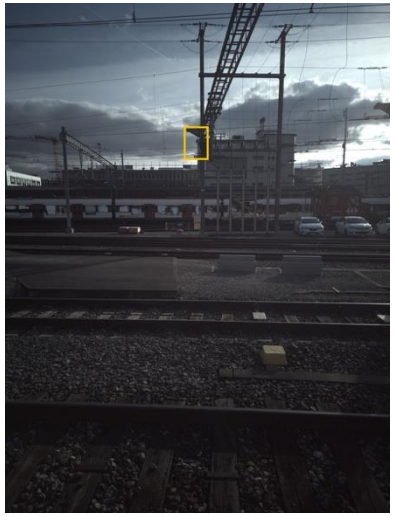


Photo 1 · Camera 1

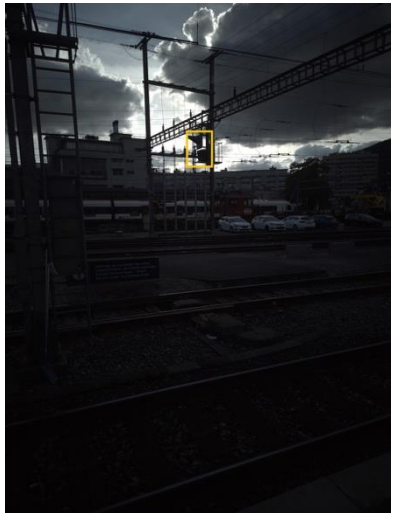


Photo 2 · Camera 2

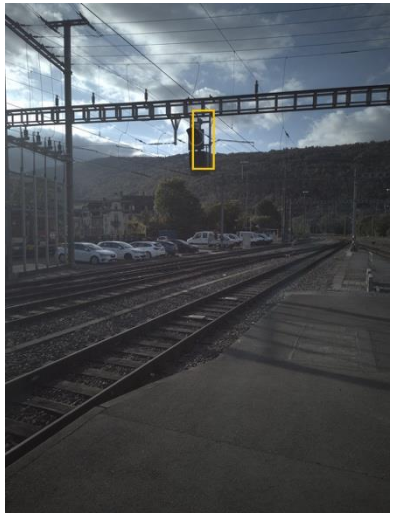


Photo 3 · Camera 3



Photo 4 · Camera 4

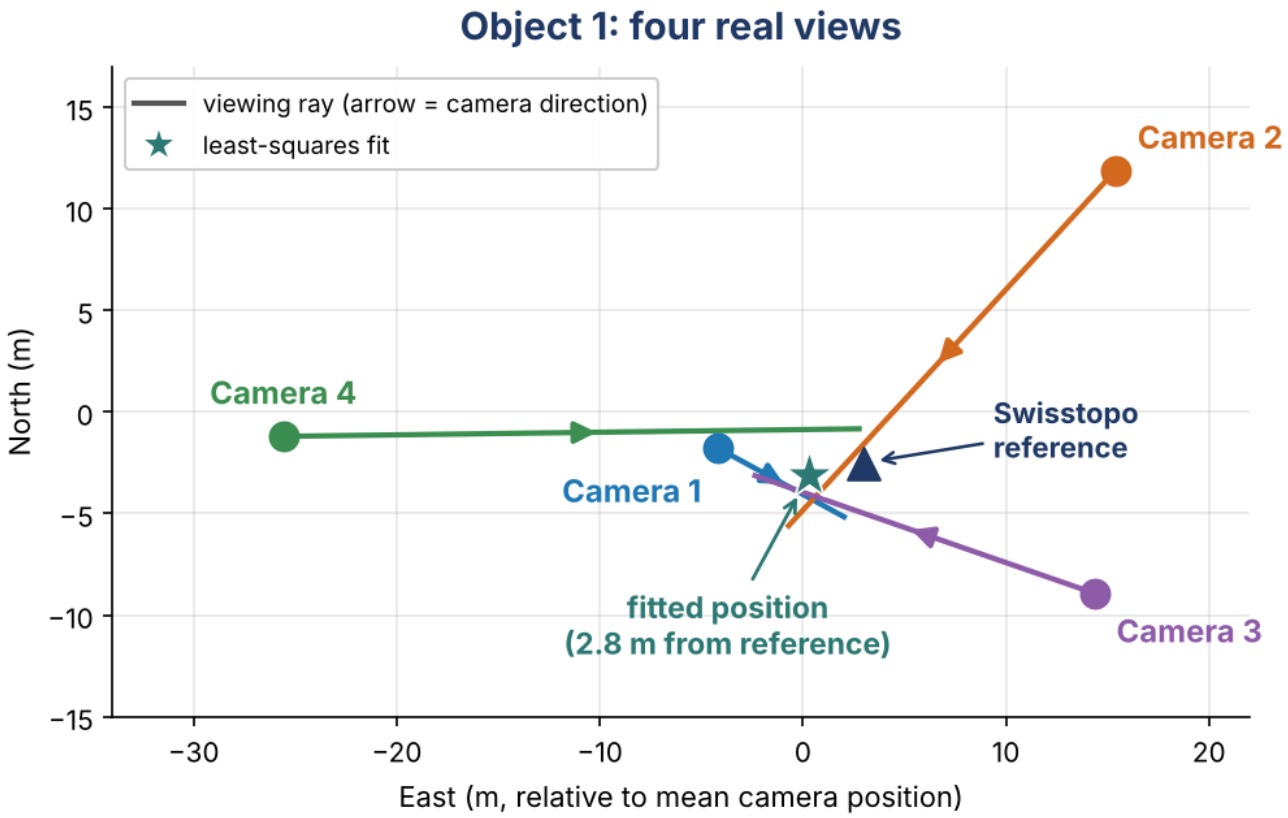
Same signal, different positions → different bearings. The JSON stores where the camera was and how it was pointing, where the signal sits in the image (bounding box), and an estimated distance to it. That is everything triangulation needs.

Photo 1 (Object 1): JSON excerpt


```
{
  "image_id": "eed57250-…",
  "gps_lat": 47.13317007,
  "gps_lon": 7.24174344,
  "gps_accuracy_m": 2.57,
  "azimuth_deg": 116.49,
  "pitch_deg": -3.20,
  "roll_deg": 0.50,
  "focal_length_mm": 24.04,
  "bounding_box": {
    "x1": 1505.5, "y1": 1014.4,
    "x2": 1696.2, "y2": 1388.8
  },
  "boxSpace": {"width": 3072,
    "height": 4080},
  "estimated_depth_m": 8,
  "device_model": "Pixel 7"
}
```


***Bold** = used by the pipeline today. **Teal** = reported GPS accuracy, planned for weighting (slide 5). One real record (photo 1 of Object 1), values rounded; image_uri, altitude and timestamp left out.*

3 Worked example: triangulating Object 1



1. Bounding box → bearing

Each photo's box position plus the camera azimuth gives a direction from the camera to the signal (82° assumed field of view).

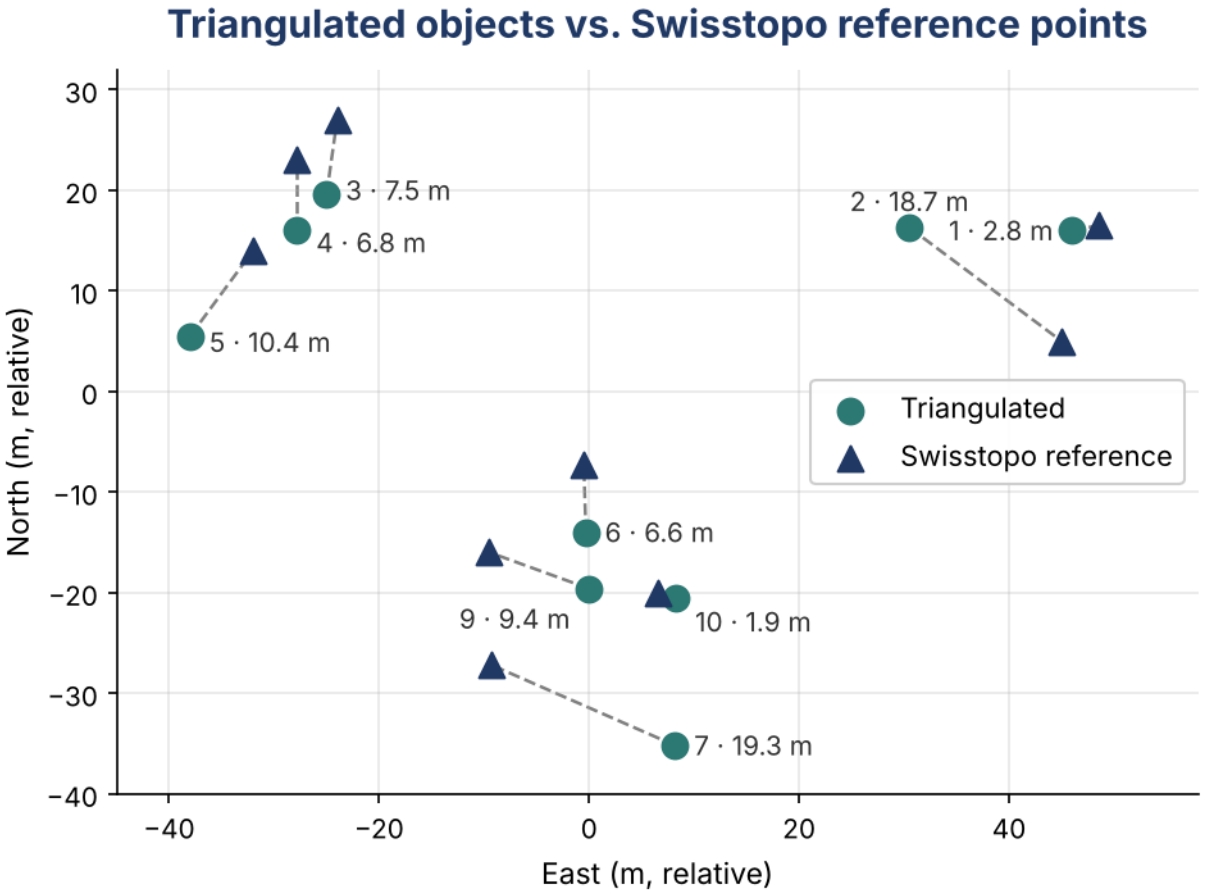
2. Direction + depth → a ray

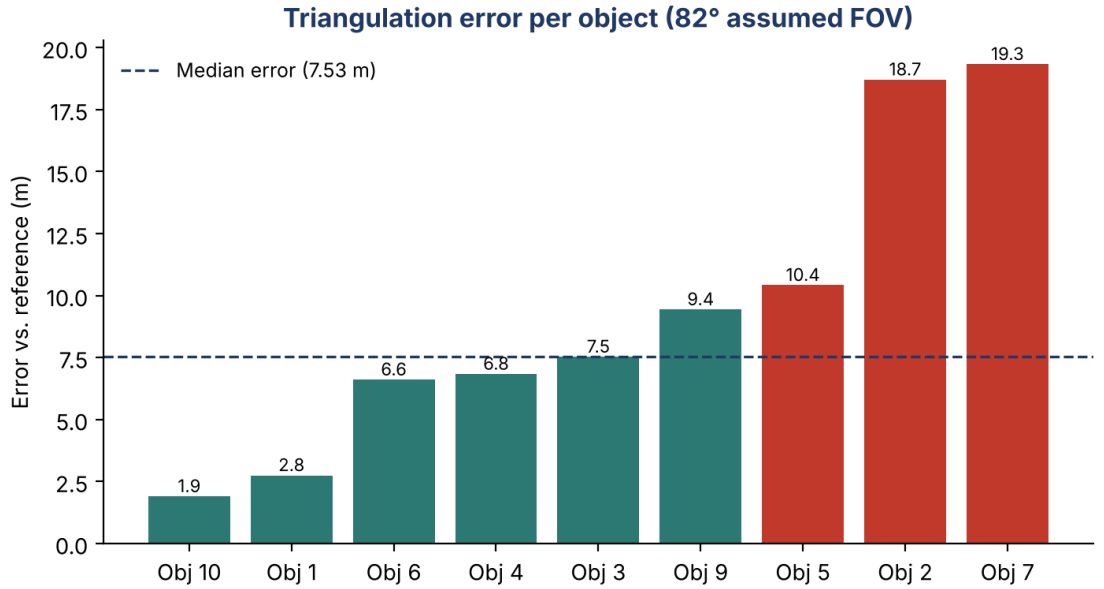
Every photo gives one viewing ray. The depth estimates (8–22 m) are only rough: camera 2 says 9 m, but the fitted point is about 21 m away. So, they only get a light weight in the fit.

3. Several rays → one fitted point

Lines from real photos never meet in exactly one point, so least squares finds the position closest to all of them. Here cameras 1, 2 and 3 cross within about 1 m of each other, and camera 4's line passes about 2 m from the fitted point. The fitted point lands 2.8 m from the Swisstopo reference.

4 Validated against real Swisstopo reference points





Median error 7.53 m across all 9 objects (mean 9.28 m) — all 9 real signals triangulated from independent camera views and matched against the Swisstopo reference markers.

5 What we learned, and what's next

Checked, and ruled out

- Swept the assumed camera field of view (50–110°) against the reference points.
- Error barely moved (6.7 m → 6.4 m) — field of view (FOV) is not the main error source.
- Part of the gap may not even be ours: Swisstopo's own GNSS measurements are only meter-accurate without differential correction, and phone GPS adds further error indoors or under the roof.

Concrete next steps

- Investigate objects 2, 5 and 7 — consistently largest errors regardless of FOV.
- Derive true FOV from focal length + sensor width, not a spec-sheet value.
- Factor in pitch/roll, not just azimuth, for tilted shots.
- Test weighting by reported GPS accuracy on all nine objects (see right).

How GPS-accuracy weighting would work

- Every JSON record reports gps_accuracy_m, the phone's own estimate of how far off its position fix may be.
- Each ray gets the weight 1 / accuracy², and the fit minimises the weighted squared distances to the rays. Rays from a precise location contributes more.
- Limits: it only covers camera-position error, not compass or field-of-view error. With 9 reference points it should be judged on new captures, not tuned on these.