0) Pipeline and what “axes” mean

- We estimate **rotation** R_k and **translation** t_k between two frames, accumulate them, and then draw the camera’s **X/Y/Z axes in the world frame** plus its position.
- **Key difference**
  - **X/Y axes**: mostly revealed by **rotation** R (robust, depth-agnostic).
  - **Z axis (forward/backward)**: mostly revealed by **translation direction** t (no absolute scale with a single camera).

1) OpenCV calls in plain words

(1) `cv2.goodFeaturesToTrack`

- **What**: finds “trackable” corner points.
- **Why**: so we can follow the same points in the next frame.
- **Main args**: `maxCorners`, `qualityLevel`, `minDistance`.
- **Returns**: $(N, 1, 2)$ array of points.

(2) `cv2.calcOpticalFlowPyrLK`

- **What**: tracks where each point moved in the next frame (pyramidal Lucas–Kanade).
- **Returns**: new positions + a success mask; we keep only successful matches.

(3) `cv2.findEssentialMat`

- **What**: robustly (RANSAC) estimates the **Essential matrix** E from point matches and intrinsics.
- **Why**: E encodes both **rotation** R and **translation direction** t:

$$E = [\mathbf{t}]_\times R, \qquad \tilde{\mathbf{x}}_2^\top E\,\tilde{\mathbf{x}}_1 = 0$$

  with normalized coordinates $\tilde{\mathbf{x}} = K^{-1}\mathbf{x}$.

(4) `cv2.recoverPose`

- **What**: decomposes E into R and t (unit-length direction), picking the physically valid solution via **cheirality** (positive depths).
- **Scale ambiguity**: monocular VO cannot recover the **magnitude** of t.

(5) `cv2.Rodrigues` / `cv2.projectPoints`

- **Rodrigues** converts between rotation matrix and rotation vector (API convenience).
- **projectPoints** maps 3D world points to pixel coordinates given rvec/tvec and intrinsics+distortion (used to draw axes on the video).

2) How X/Y axes are “detected” (via rotation)

- **Intuition**: even small yaw/pitch/roll produces a global, depth-weak image motion pattern; this makes R relatively easy to estimate.
- **Math**: from E, SVD + cheirality give R. The **columns of R_{WC}** are exactly the **unit directions** of camera **X/Y/Z** in world.
- **In code**: accumulate `R_CW ← R_k @ R_CW`, then `R_WC = R_CW.T`; draw $t_{WC} + R_{WC}[L,0,0]^\top$ and $t_{WC} + R_{WC}[0,L,0]^\top$.
- **Takeaway**: **X/Y directions come from rotation**, largely independent of unknown depth/scale.

3) How the Z axis is “detected” (via translation)

- **Intuition**: forward/backward motion creates a **radial parallax** toward the **epipole**; this reveals the **direction of t**.
- **Limitation**: monocular = **no absolute scale** of t. We get direction only; magnitude needs an external scale.
  - In code, positions are in arbitrary units; `--scale` applies your real-world meter scale at save time.
- **Degeneracies**:
  - **Pure rotation**: no translation → can't infer t.
  - **Straight-ahead motion**: tiny parallax → unstable.
  - **Planar scenes**: homography dominates → weak translation estimate.
- **Takeaway**: **Z is observable mainly as translation direction**, and its reliability depends on scene/motion; magnitude must be calibrated.

4) Minimal formulas (reference)

$$E = [\mathbf{t}]_\times R,\ \tilde{\mathbf{x}}_2^\top E\,\tilde{\mathbf{x}}_1 = 0,\ R_{CW}^{(k)} = R_k R_{CW}^{(k-1)},\ t_{CW}^{(k)} = R_k t_{CW}^{(k-1)} + t_k,\ R_{WC} = R_{CW}^\top,\ t_{WC} = -R_{CW}^\top t_{CW}$$

5) Practical notes

- **X/Y** are robust thanks to RANSAC-backed $E \to R$.
- **Z** needs **external scale** (e.g., known baseline) → use `--scale`.
- The code re-detects features when they drop and re-orthogonalizes rotations for numeric stability.

In computer vision, detecting motion along the **x- and y-axes** relies on tracking how points shift across consecutive image frames. If a point moves horizontally or vertically in the image, OpenCV can measure this displacement as a change in pixel coordinates, which indicates lateral motion. These shifts are relatively straightforward to detect because they correspond directly to image-plane movement.

By contrast, detecting motion along the **z-axis** (forward or backward) depends on how the spacing between points changes rather than their direct displacement. As an object approaches the camera, features spread outward from the **epipole** in a radial pattern; as it recedes, they contract toward the epipole. This "radial disparity" is harder to detect, but it provides depth information that x- and y-axis motion alone cannot reveal.