Visual Pose Estimation

Graduate Course INTR-6000P

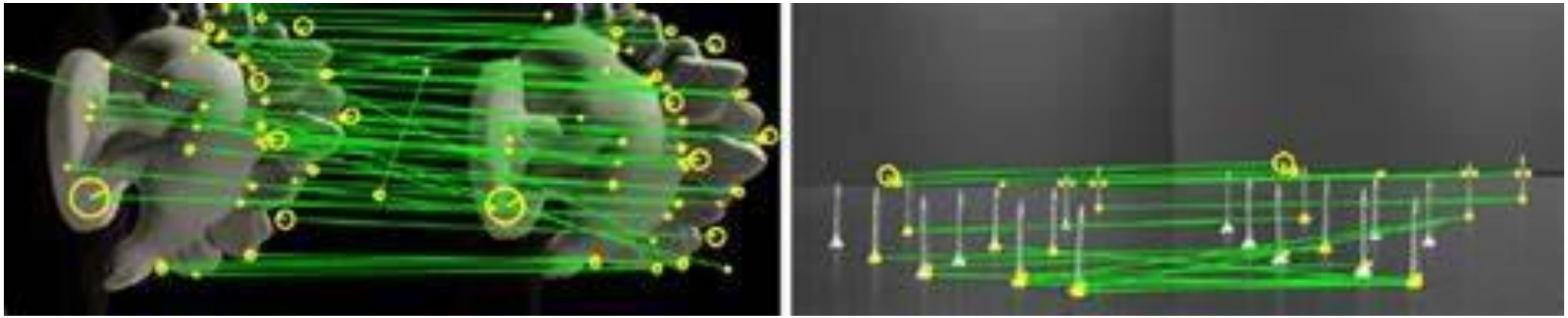
Week 4 - Lecture 7

Changhao Chen

Assistant Professor

HKUST (GZ)

Recap: Visual Feature



Keypoint (Location): The (x, y) coordinates of the feature.
Descriptor (Signature): A numerical vector that describes the visual appearance of the patch around the keypoint.

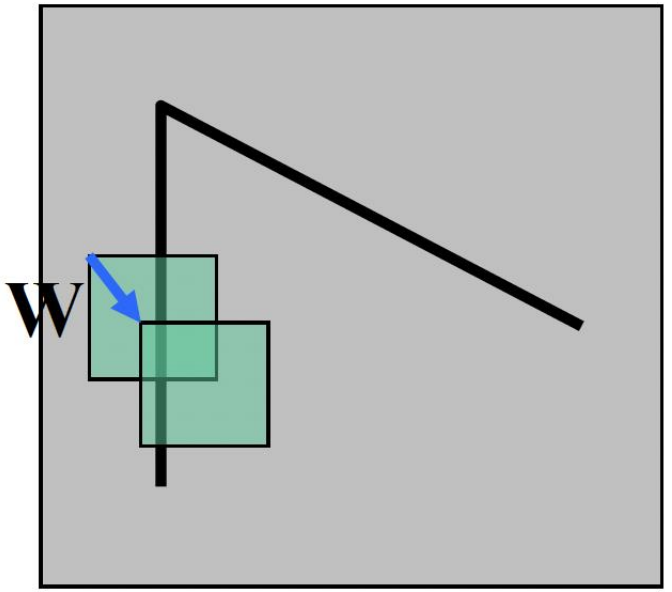
Main Tasks:

- Feature Detection;
- Feature Tracking;
- Feature Matching.

Recap: Corner Detection

We find corners by looking for windows where a shift in any direction causes a large change in pixel intensity.

- For a small image window, we calculate the amount of change caused by shifting it in every possible direction.
- This change is measured using the Sum of Squared Differences (SSD) of pixel intensities.
- A true corner is identified when this change value is large for all possible directions of shift.



The change function E(u,v) is formally defined as the Sum of Squared Differences (SSD) within a window around a point (u,v). Corners are subsequently localized at the pixels where the value of E(u,v) is a local maximum and surpasses a predefined threshold.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

Recap: Faster Feature Detection

SURF (Speeded-Up Robust Features):

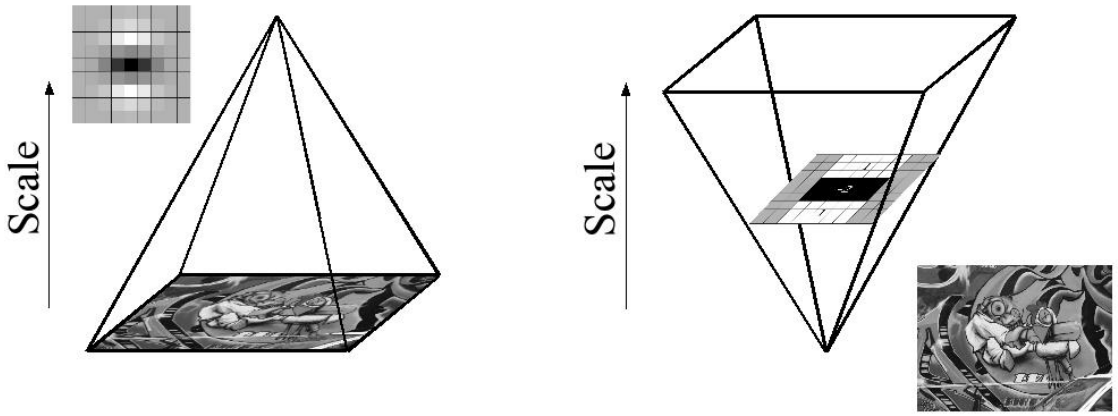
Similar to SIFT but uses approximations (integral images) for much faster computation. A good speed/accuracy trade-off.

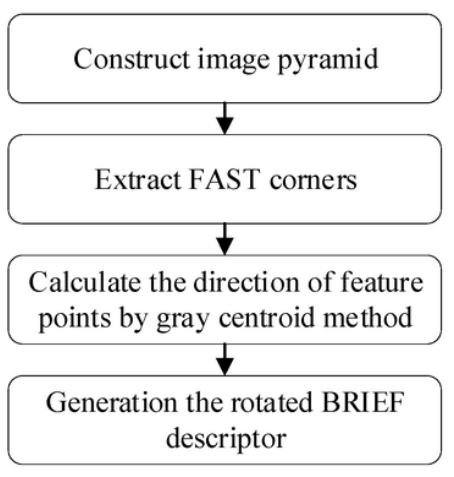
ORB (Oriented FAST and Rotated BRIEF):

FAST: Very quick keypoint detector (looks at a circle of pixels).

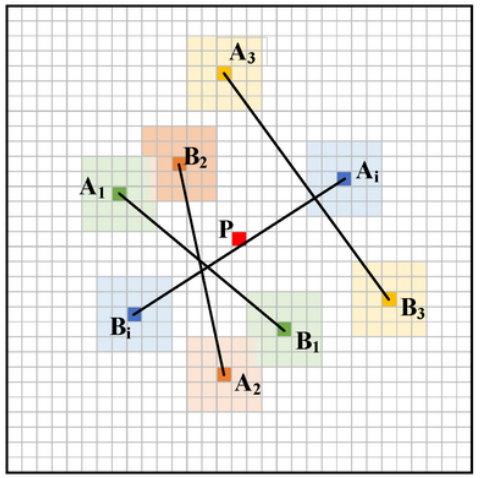
BRIEF: Very simple binary descriptor (compares pixel intensities).

ORB = FAST + BRIEF + improvements. It's binary, meaning it's extremely fast and low on memory. Crucial for real-time vehicle systems.





(a)



(b)

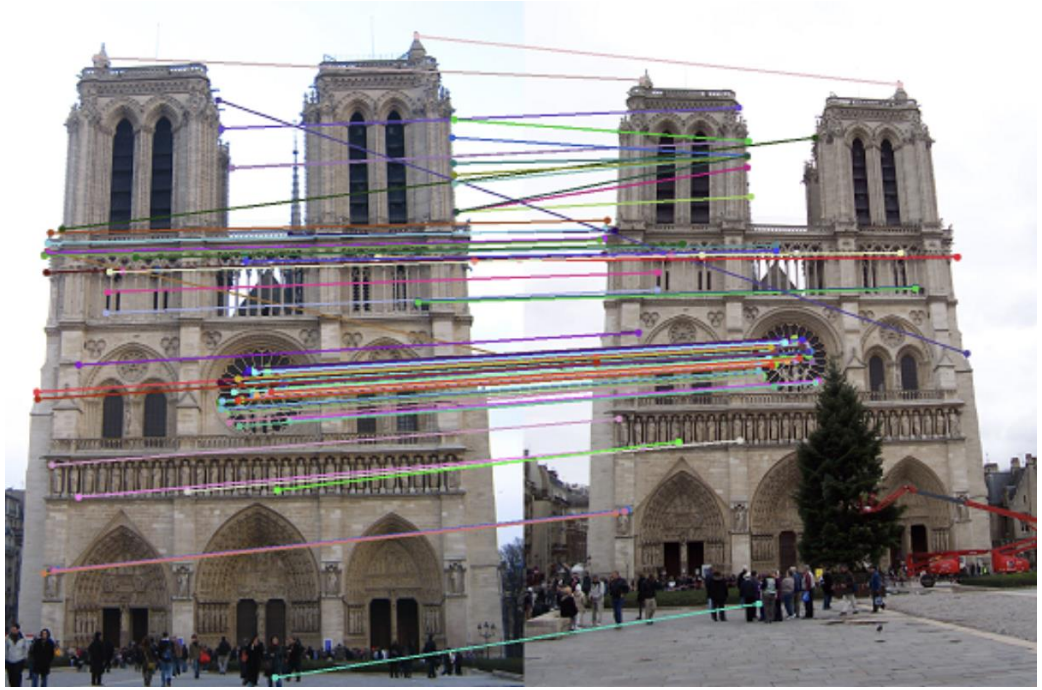
Recap: Feature Matching

Solve the problem of data association between images.

Calculate the descriptor distance between features, which represents the similarity between features.

For real-valued descriptors (SIFT, SURF): Use Nearest Neighbor search (e.g., L2 distance). Use Ratio Test (Lowe's ratio) to reject ambiguous matches.

For binary descriptors (ORB, BRIEF): Use Hamming distance (number of different bits). Much faster to compute.



Recap: Feature Matching

RANSAC (RANdom SAmple Consensus)

Many matches are incorrect. A simple least-squares fit would be severely skewed by these outliers. RANSAC is designed to find the correct model in spite of these errors.

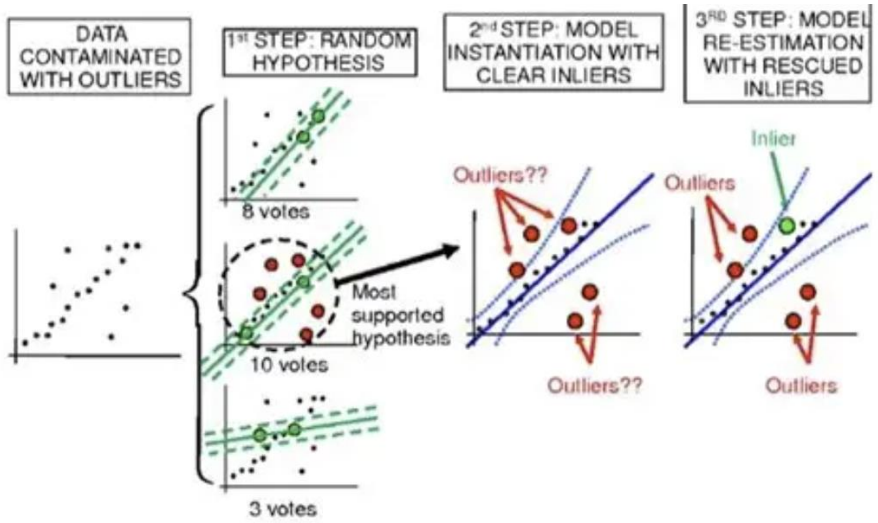
1) Random Sampling: Randomly select the smallest number of points needed to define the model (e.g., 4 points to estimate a homography).

2) Model Estimation: Compute a candidate model from this small, hopefully all-inlier, subset.

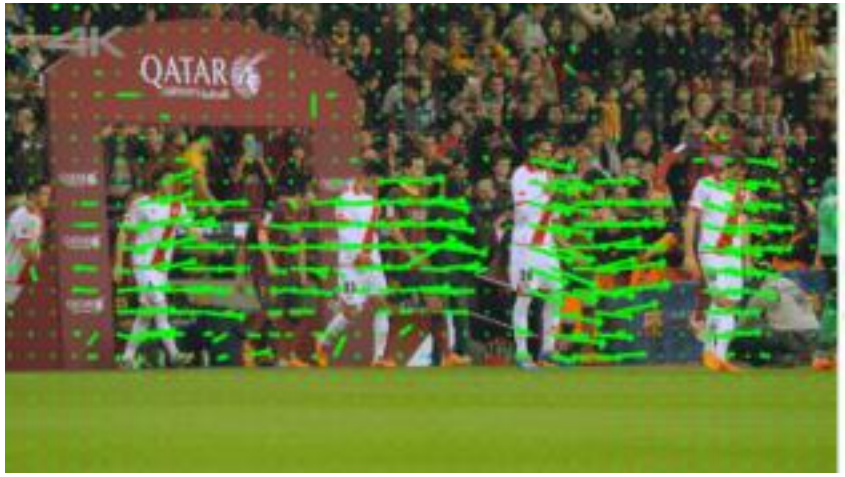
3) Consensus (Voting): Test all other data points against this model. Points that fit the model well within a tolerance are considered inliers and form the "consensus set."

4) Iterate & Select: Repeat this process many times. The model with the largest consensus set (the most inliers) is chosen as the best model.

5) Re-estimate: Finally, the model parameters are re-estimated using all the inliers in the best consensus set for a robust fit.



Recap: Feature Tracking: The Optical Flow



Matching: Between two arbitrary images (e.g., from different cameras).

Tracking: Following a feature through a sequence of images (a video from a single camera).

Optical Flow: The pattern of apparent motion of objects between consecutive frames caused by the movement of the object or the camera.

Assumption: Brightness Constancy - a point looks the same in consecutive frames.

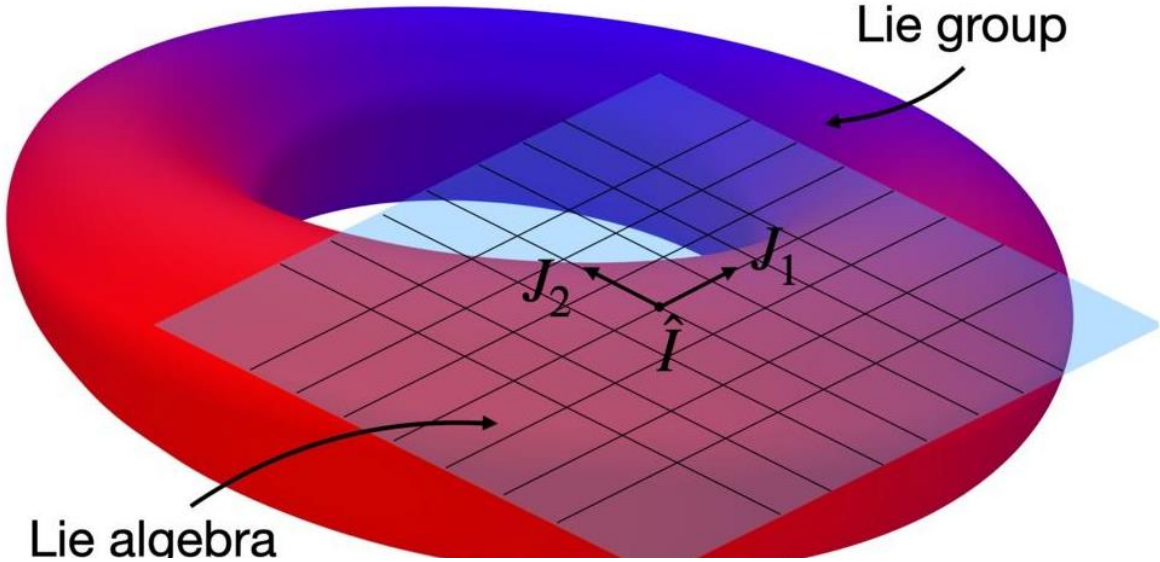
Lucas-Kanade Method: A classic, efficient algorithm for sparse optical flow (tracking specific feature points).

Recap: Lie Group

A group that is also a smooth manifold, where the group operations (composition and inversion) are smooth.

A continuous set of transformations that you can:

- Compose: T1 ∘ T2 is also a transformation.
- Invert: T^{-1} exists and is also a transformation.
- It's smooth.



Lie Groups. A group $\mathcal{G}$ is said to be a *Lie group* embedded in $\mathbb{R}^N$ if:

- $\mathcal{G}$ is a manifold in $\mathbb{R}^N$;
- the group operations (composition and inverse) are smooth (infinitely differentiable)

Visual Pose Estimation

1. 2D-2D Pose Estimation (Monocular)

Input: Corresponding 2D points in two consecutive images.

Output: Relative camera rotation (R) and translation (t) up to scale.

Challenge: Scale ambiguity.

2. 3D-2D Pose Estimation (PnP)

Input: 2D image points and their corresponding known 3D points (from a map or previous reconstruction).

Output: Absolute 6-DOF camera pose (R, t).

Advantage: Provides metric scale.

2D-2D Correspondence

Input:

Two images of the same rigid scene, taken from two different camera positions.

A set of 2D-2D correspondences: {p_i <-> p'_i}

p_i = (u_i, v_i, 1) is a keypoint in Image 1 (in homogeneous coordinates).

p'_i is the matching keypoint in Image 2.

Output:

The relative pose (R, t) of Camera 2 relative to Camera 1.

Epipolar Geometry

Objective: To understand how the epipolar constraint p'^T E p = 0 is derived from the geometry of two views.

Notation:

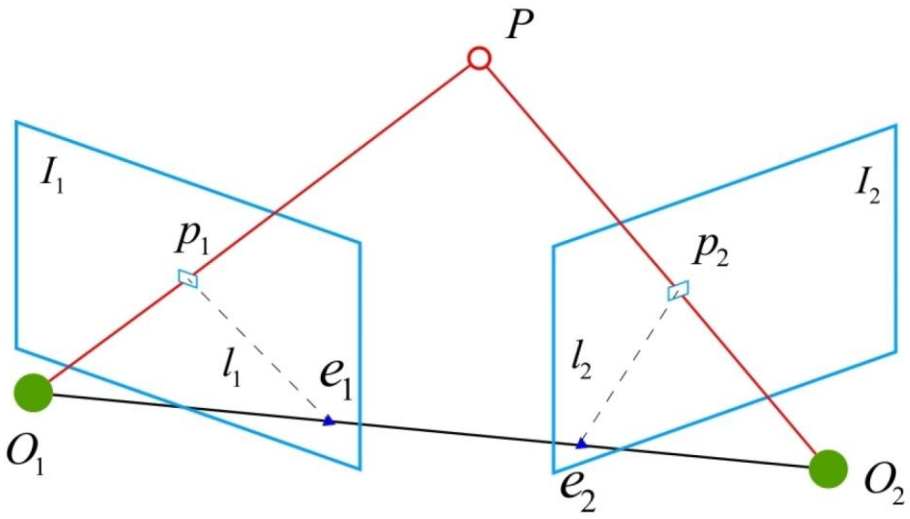
P = [X, Y, Z]^T: A 3D world point in the coordinate system of the first camera.

p = [u, v, 1]^T: The 2D projection of P onto the image plane of the first camera (in homogeneous coordinates).

p' = [u', v', 1]^T: The 2D projection of P onto the image plane of the second camera.

O, O': The optical centers of the first and second cameras.

R, t: The rotation and translation that transform a point from the first camera's coordinate system to the second's. X' = R X + t.



Epipolar Geometry

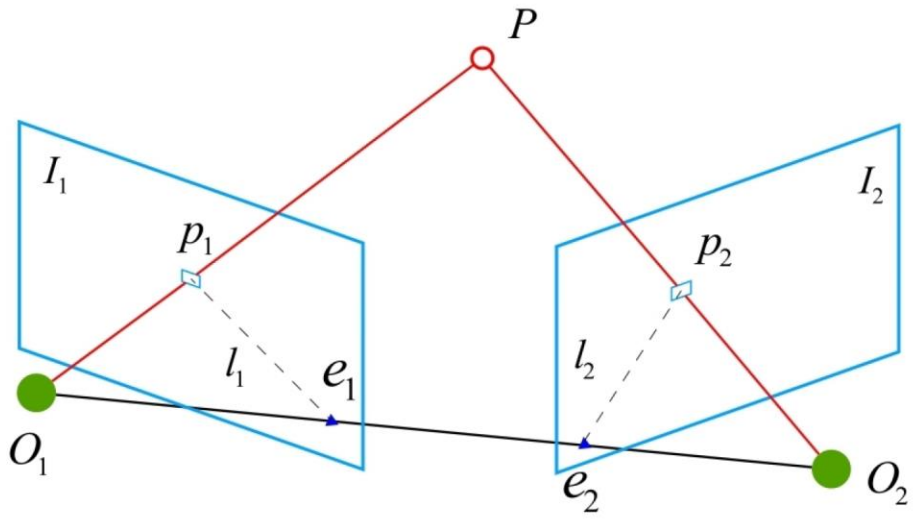
P (O->P), t (O->O'), and P' (O'->P).

All vectors lie on the same plane (the epipolar plane). Therefore, the vector P' is orthogonal to the normal of the plane defined by t and P.

This gives us the scalar triple product:

$$\mathbf{P'}^T(\mathbf{t} \times \mathbf{P}) = 0$$

This is the most important geometric insight.



Epipolar Geometry

We know the transformation between camera frames:
P' = R P + t

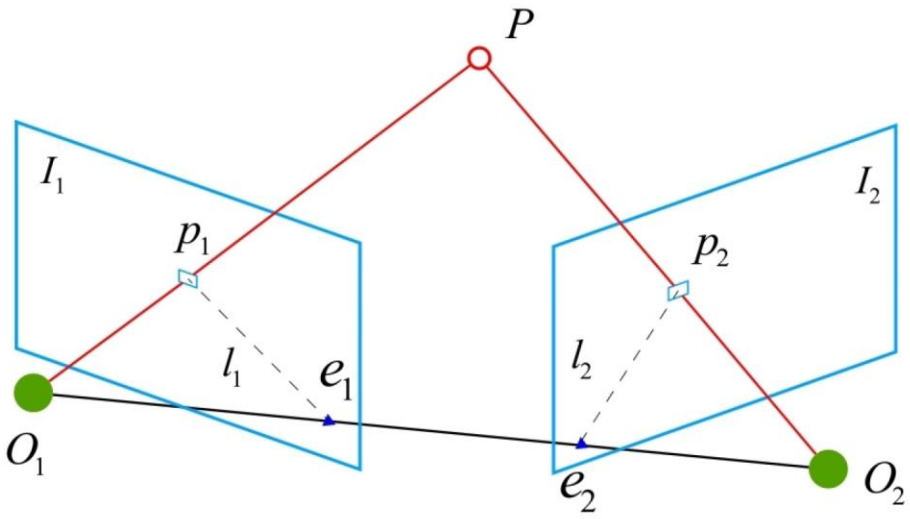
We can express the first camera's point P in terms of the second:
P = R^T (P' - t)

Substitute into our constraint:

$$\mathbf{P'}^T \left(\mathbf{t} \times \left(\mathbf{R}^T(\mathbf{P'} - \mathbf{t})\right)\right) = 0$$

Since t × t = 0, this simplifies beautifully to:

$$\mathbf{P'}^T \left(\mathbf{t} \times \left(\mathbf{R}^T\mathbf{P'}\right)\right) = 0$$



Epipolar Geometry

The Skew-Symmetric Matrix Trick

The cross product t × v can be rewritten as a matrix multiplication:

$$\mathbf{t} \times \mathbf{v} = [\mathbf{t}]_\times \mathbf{v}$$

Where the skew-symmetric matrix is:

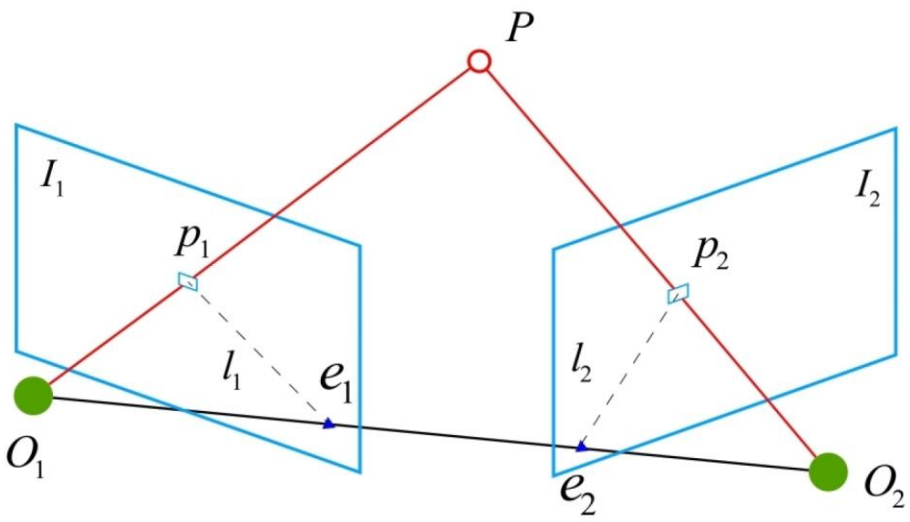
$$[\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

Apply the trick:

$$\mathbf{t} \times (\mathbf{R}^T \mathbf{P}') = [\mathbf{t}]_\times \mathbf{R}^T \mathbf{P}'$$

Our constraint becomes:

$$\mathbf{P}'^T ([\mathbf{t}]_\times \mathbf{R}^T) \mathbf{P}' = 0$$



Epipolar Geometry

We define the result of that multiplication as a single matrix:

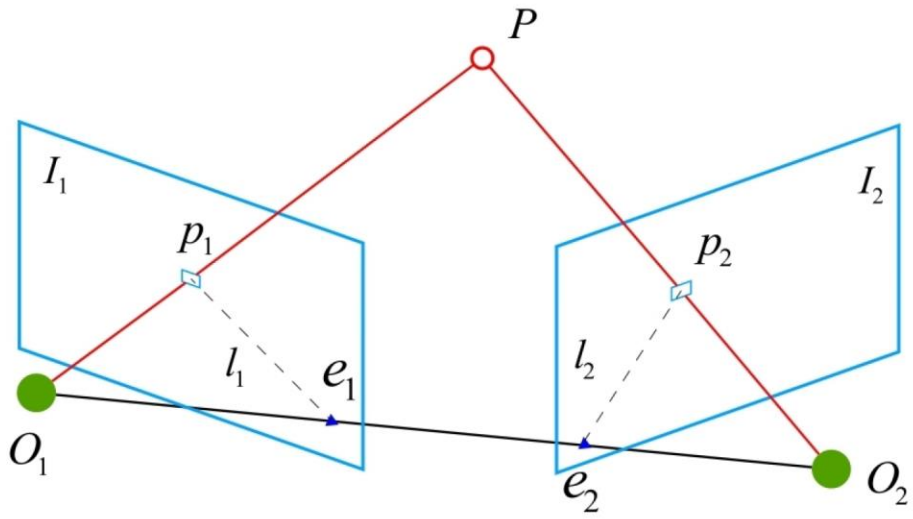
The Essential Matrix

$$\mathbf{E} = [\mathbf{t}]_{\times}\mathbf{R}^T$$

This allows us to write the coplanarity constraint in its most compact form:

$$\mathbf{P'}^T\mathbf{E}\mathbf{P'} = 0$$

This equation uses the 3D point P' as seen in the second camera's frame.



Epipolar Geometry

How do we relate the 3D point P' to the 2D pixel p' we actually see? The Pinhole Camera Model

For a calibrated camera (known intrinsics K):

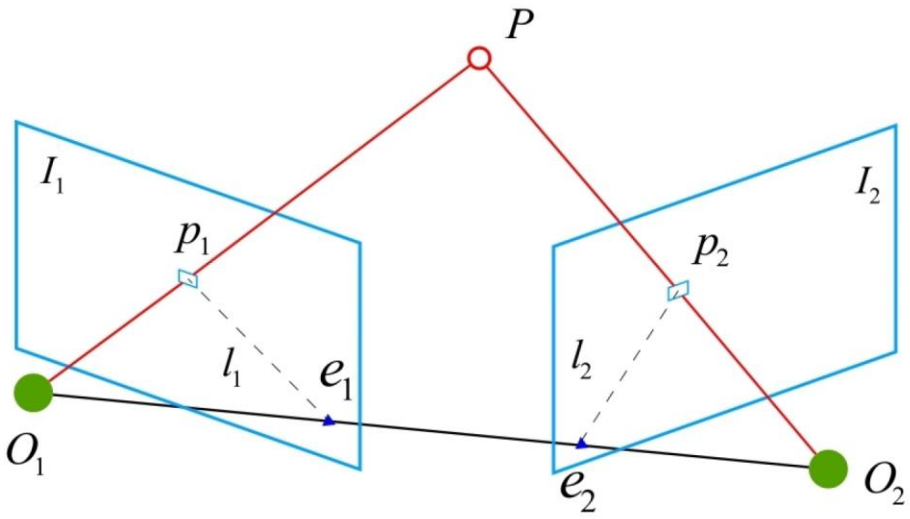
$$\mathbf{p}' = \mathbf{K}\mathbf{P}'_{norm}$$

where P'_norm = P' / Z' is the normalized image coordinate (a 3D point on the plane Z=1).

Therefore, the normalized coordinate is:

$$\mathbf{P}'_{norm} = \mathbf{K}^{-1}\mathbf{p}'$$

(The same applies for p and P in the first camera)



Epipolar Geometry

The Fundamental Matrix

Our 3D constraint must also hold for normalized coordinates:

$$\mathbf{P'}_{norm}^{T}\mathbf{E}\mathbf{P}_{norm} = 0$$

Now substitute the camera model:

$$(\mathbf{K}^{-1}\mathbf{p'})^{T}\mathbf{E}(\mathbf{K}^{-1}\mathbf{p}) = 0$$

Rearrange using (AB)^T = B^T A^T:

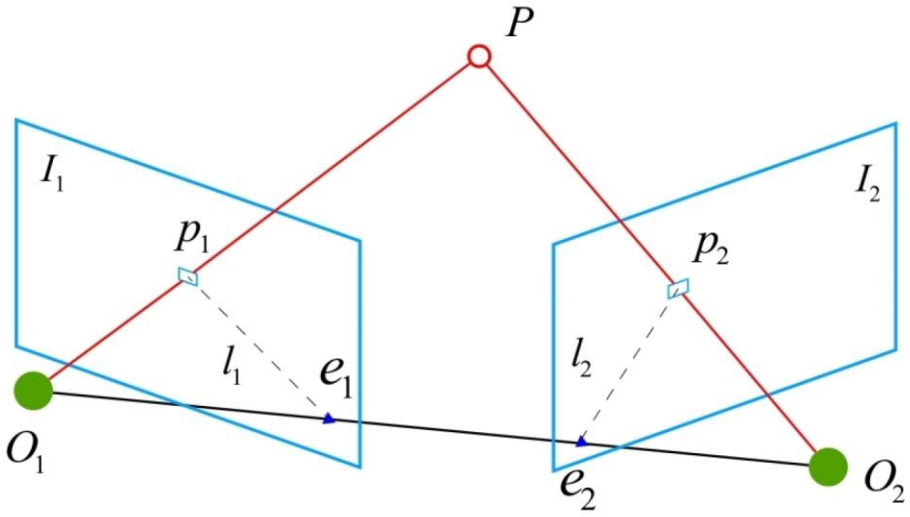
$$\mathbf{p'}^{T}(\mathbf{K}^{-T}\mathbf{E}\mathbf{K}^{-1})\mathbf{p} = 0$$

We define the term in parentheses as the Fundamental Matrix:

$$\mathbf{F} = \mathbf{K}^{-T}\mathbf{E}\mathbf{K}^{-1}$$

The fundamental constraint for two-view geometry:

$$\mathbf{p'}^{T}\mathbf{F}\mathbf{p} = 0$$



What this means: For a point p in the first image, its corresponding point p' in the second image must lie on a line defined by F p.

Epipolar Geometry

The Epipolar Line

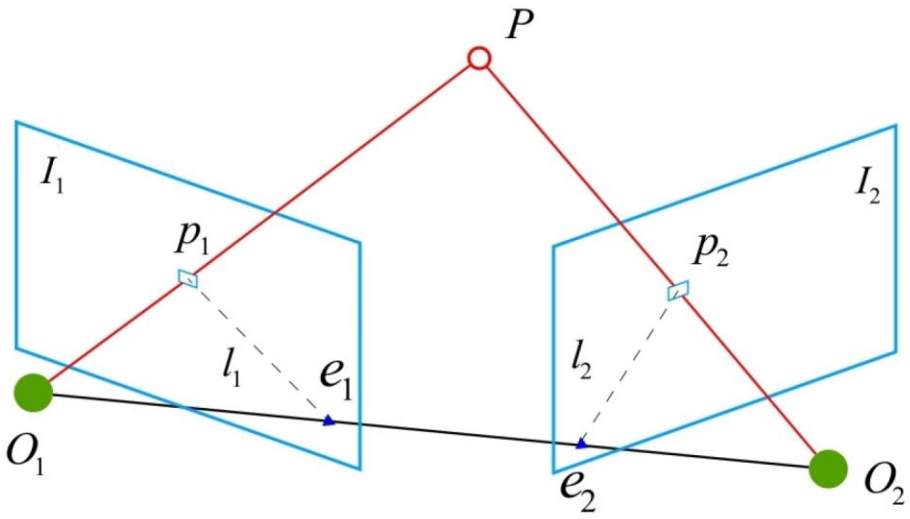
The equation p'^T F p = 0 means p' lies on the line l' where:

$$\mathbf{l}' = \mathbf{F}\mathbf{p}$$

l' is the epipolar line in the second image.

This reduces the search for a match from a 2D area to a 1D line!

This is crucial for outlier rejection via algorithms like RANSAC.



Epipolar Geometry

The epipolar constraint is mathematically encoded in the Essential Matrix E.

$p'^T * E * p = 0$

This must hold for every correct correspondence p <-> p'.

$E = [t]_x * R$

- It's a 3x3 matrix that combines the rotation R and translation t.
- $[t]_x$ is the skew-symmetric matrix form of the translation vector t. This is a direct link to the se(3) Lie algebra!

Properties of E:

- Has 5 degrees of freedom (3 for rotation, 2 for translation direction; scale is ambiguous).
- Its singular values are [σ, σ, 0].

The Eight-Point Algorithm

Use multiple point correspondences to set up a system of equations to solve for E.

Step 1: Reformulate the Constraint

- Take the equation $p'^T E p = 0$.
- Rewrite it as a linear system: [u'u, u'v, u', v'u, v'v, v', u, v, 1] * e = 0
- where $e = [E_{11}, E_{12}, E_{13}, E_{21}, ..., E_{33}]^T$ is the vectorized version of E.

Step 2: Stack Equations

- For each of the N point correspondences, we get one equation.
- Stack them into a matrix: A * e = 0, where A is an N x 9 matrix.

Step 3: Solve the System

- We need at least 8 points (hence the name) to get a unique solution (up to scale) for this homogeneous system.
- Solve using SVD: $A = U \Sigma V^T$. The solution for e is the last column of V (the singular vector corresponding to the smallest singular value).

Camera Pose Estimation

We have E, now we need to extract R and t.

Perform SVD on the computed E: $E = U \Sigma V^T$.

There are four possible solutions for (R, t):
$(U W V^T, +u_3)$
$(U W V^T, -u_3)$
$(U W^T V^T, +u_3)$
$(U W^T V^T, -u_3)$

where W is a fixed matrix: [[0, -1, 0], [1, 0, 0], [0, 0, 1]] and u_3 is the third column of U.

The Cheirality Check
For all four solutions, triangulate one 3D point X from its correspondences p and p'.
The correct solution is the one where the triangulated point X is in front of both cameras (has a positive depth $Z > 0$ in both camera coordinate systems).

Questions

Question 1) What if the camera intrinsics (K) are unknown?

We use the Fundamental Matrix F instead.

- The constraint is $p'^T F p = 0$.
- $F = K'^{-T} E K^{-1}$. It relates normalized image coordinates instead of pixel coordinates.
- The 8-point algorithm works exactly the same way for F.

Question 2) Dealing with Noise and Outliers:

The basic 8-point algorithm is sensitive to incorrect matches (outliers).

Solution: Use RANSAC (Random Sample Consensus).

- Randomly select 8 point correspondences.
- Compute a candidate E or F.
- Count how many other points satisfy the epipolar constraint (are "inliers").
- Repeat for many iterations. Keep the E/F with the most inliers.
- Recompute E/F using all the inliers for a final, clean solution.

The 2D-2D Pose Estimation

1) Input: Image pair + 2D-2D correspondences (from SIFT, ORB, etc.).

2) Normalize Coordinates: (Optional but recommended) Transform points using K^{-1} to work with E for better numerical stability.

3) Robust Estimation: Use the 8-Point Algorithm inside a RANSAC loop to compute the Essential Matrix E.

4) Pose Extraction: Perform SVD on E to get the four possible (R, t) solutions.

5) Disambiguation: Perform a Cheirality Check (triangulation) to find the single physically possible pose.

6) Output: The relative camera pose $T = [R \mid t]$, an element of SE(3).

The 2D-2D Pose Estimation

Limitations:

The Fundamental Problem: We recover a unit vector for t.
We know direction but not magnitude.

Why is this catastrophic for a car? "I'm moving forward"
vs. "I'm moving forward at 5 km/h vs. 50 km/h".

Other issues: Degenerate motion (pure rotation),
requires triangulation for structure.

3D-2D Pose Estimation (PnP)

Question: "I have a pre-built map of the world. I see it with my camera. Where am I?"

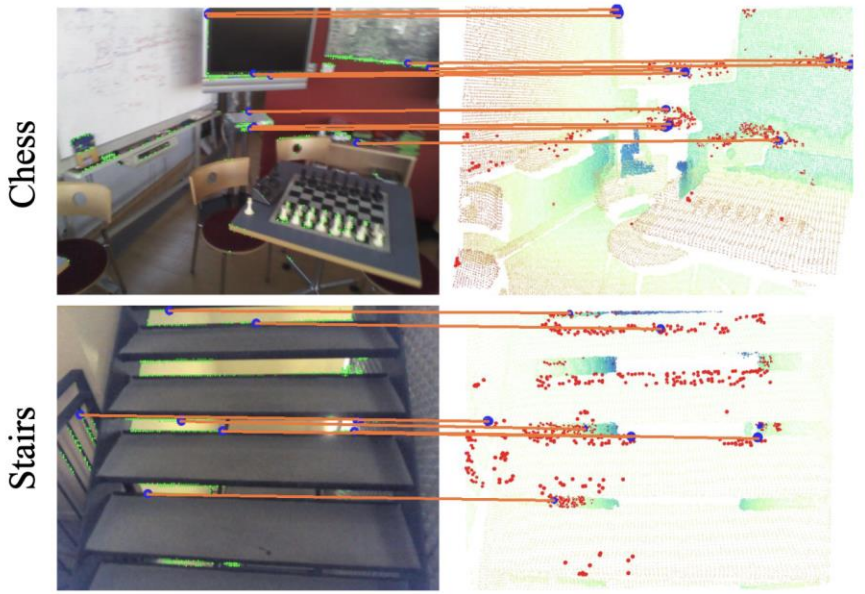
Input:

3D Map Points: Known 3D landmarks ($P_i = [X_i, Y_i, Z_i]^T$) from a prebuilt map or SLAM.

2D Image Points: Their current 2D projections ($p_i = [u_i, v_i]^T$) in the camera image.

Output: The 6-DOF camera pose (Rotation R and Translation t) relative to the map.

This is the Perspective-n-Point (PnP) problem.



3D-2D Pose Estimation (PnP)

Solves Scale: Unlike 2D-2D methods, PnP provides metric scale directly from the map.

Efficiency: Extremely fast pose estimation for each new camera frame.

Enables Localization: The core algorithm for matching a live camera feed to a pre-built HD Map.

Foundation for Tracking: Used in every major SLAM system (like ORB-SLAM) for tracking the camera's position in the local map.

3D-2D Pose Estimation (PnP)

The Pinhole Camera Model & The Goal

The projection of a 3D point P to a 2D pixel p is given by:

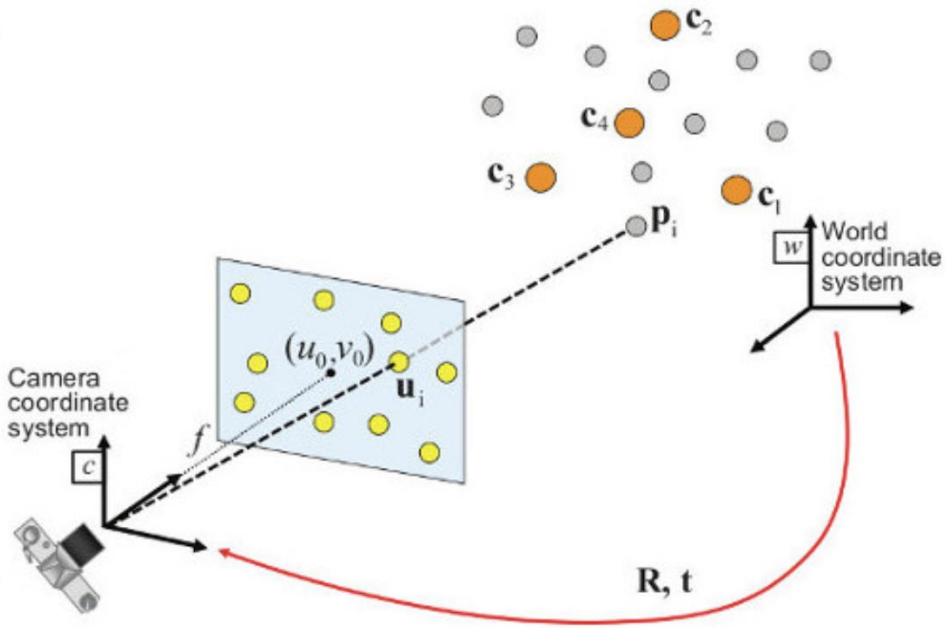
$$s\,\mathbf{p} = \mathbf{K}[\mathbf{R}\,|\,\mathbf{t}]\,\mathbf{P}$$

where s is a scale factor (the point's depth).

Known: K (camera intrinsics), P (3D point), p (2D pixel).

Unknown: R (rotation), t (translation).

Our goal: Find the R and t that best satisfy this equation for all n point correspondences (P_i, p_i).



3D-2D Pose Estimation (PnP)

The Reprojection Error
We can't solve the equations perfectly due to noise. Instead, we find the pose that minimizes the reprojection error.

Reprojection Error for a single point:

$$\text{error}_i = || \mathbf{p}_i - \pi(\mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{P}_i) ||^2$$

where π([x, y, z]^T) = [x/z, y/z]^T is the perspective projection function.

Total Error we want to minimize:

$$(\mathbf{R}, \mathbf{t})^* = \arg\min_{\mathbf{R},\mathbf{t}} \sum_i || \mathbf{p}_i - \pi(\mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{P}_i) ||^2$$

This is a non-linear least squares problem.

3D-2D Pose Estimation (PnP)

Solving PnP: A Taxonomy of Methods

- ✓ Direct Linear Transform (DLT): A linear, non-minimal solution. Sensitive to noise.
- ✓ Iterative Methods: Minimize reprojection error from an initial guess. Accurate but needs a guess.
- ✓ EPnP (Efficient PnP): A non-iterative, accurate, and fast method. Excellent for real-time.
- ✓ Robust Methods: Combine a minimal solver with RANSAC to handle outliers.
- ✓ Bundle Adjustment

We'll explore the most important ones.

3D-2D Pose Estimation (PnP)

Direct Linear Transform (DLT)

Idea: Rearrange the projection equation to eliminate the scale factor s and set up a linear system of equations.

From:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

We can solve for the entire projection matrix P = K[R|t] linearly, and then later extract R and t from P.

3D-2D Pose Estimation (PnP)

DLT – Derivation

Let P = K[R|t] be a 3x4 matrix. The equation is:

$$s\,\mathbf{p} = \mathbf{P}\mathbf{P}_{hom}$$

This is a homogeneous equation. We can eliminate s by crossing both sides with p:

$$\mathbf{p} \times (\mathbf{P}\mathbf{P}_{hom}) = 0$$

This cross-product gives two independent linear equations per point correspondence. With 6 points, we get 12 equations to solve for the 12 unknowns of P.

3D-2D Pose Estimation (PnP)

DLT - Pros and Cons

Pros:

- Simple and straightforward to implement.
- Provides a closed-form solution.

Cons:

- Not minimal: Requires 6 points instead of the theoretical minimum of 3.
- Sensitive to noise: Minimizes an algebraic error, not the geometric reprojection error.
- The extracted rotation matrix R from P may not be orthonormal and must be "fixed" (e.g., via SVD).

Use Case:

Good for getting an initial estimate, but rarely used alone in practice.

3D-2D Pose Estimation (PnP)

EPnP (Efficient PnP)

Idea: Express every 3D point P_i as a weighted sum of 4 non-coplanar control points C_j.

$$\mathbf{P}_i = \sum_{j=1}^{4} \alpha_{ij} \mathbf{C}_j$$

The entire problem transforms from solving for N 3D points to solving for the 4 control points in the camera coordinate system.

This is a much smaller problem! The coordinates of the 4 control points become the unknowns (12 unknowns).

3D-2D Pose Estimation (PnP)

EPnP - The Steps

1. Select 4 control points in the world frame (e.g., via centroid and PCA).

2. Compute the barycentric coordinates α_ij for each 3D point P_i w.r.t. the control points.

3. The projection equation for each point becomes a linear function of the camera-coordinate control points c_j:

$$s_i \mathbf{p}_i = \mathbf{K} \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j$$

4. Solve this linear system for the c_j's. This is the core linear step.

5. Recompute the original 3D points in the camera frame from the c_j's.

6. Compute the pose (R, t) that best aligns the world points P_i with their camera-frame equivalents (using absolute orientation, e.g., via SVD).

3D-2D Pose Estimation (PnP)

EPnP Advantages

1. Non-Iterative: Fast and deterministic.
2. Accurate: Performance is on par with the best iterative methods.
3. Scalable: Complexity is linear in the number of points.
4. Implementation: It's the default method in OpenCV (cv::SOLVE_PNP_EPNP).

Problems:

- Feature matching between a camera image and a map is never perfect.
- Outliers (incorrect matches) will completely corrupt any PnP solution.
- We need a way to find the correct pose and the set of correct matches (inliers) simultaneously.

3D-2D Pose Estimation (PnP)

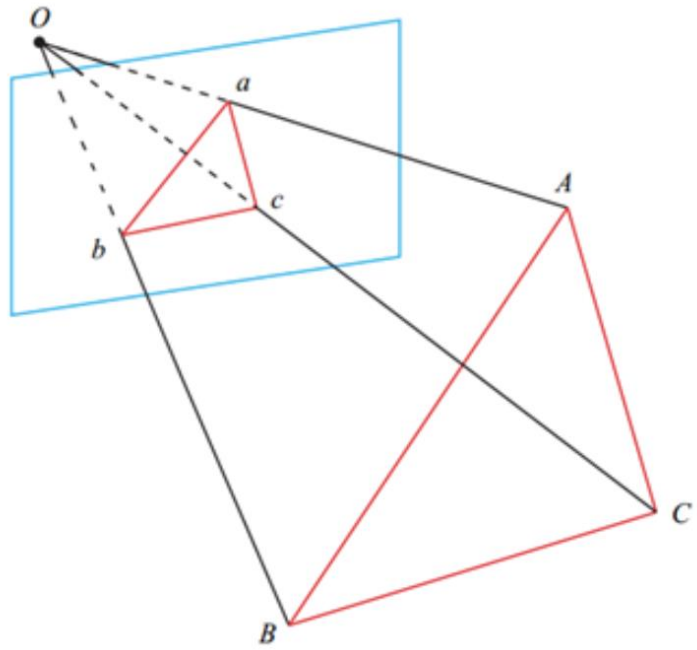
RANSAC + PnP

We use RANSAC (RANdom SAmple Consensus) as a robust framework around a minimal PnP solver.

The Minimal Case: P3P

Only 3 points are needed to compute a finite number of possible camera poses.

We can use a minimal solver (e.g., P3P) inside the RANSAC loop.



3D-2D Pose Estimation (PnP)

The RANSAC-PnP Pipeline

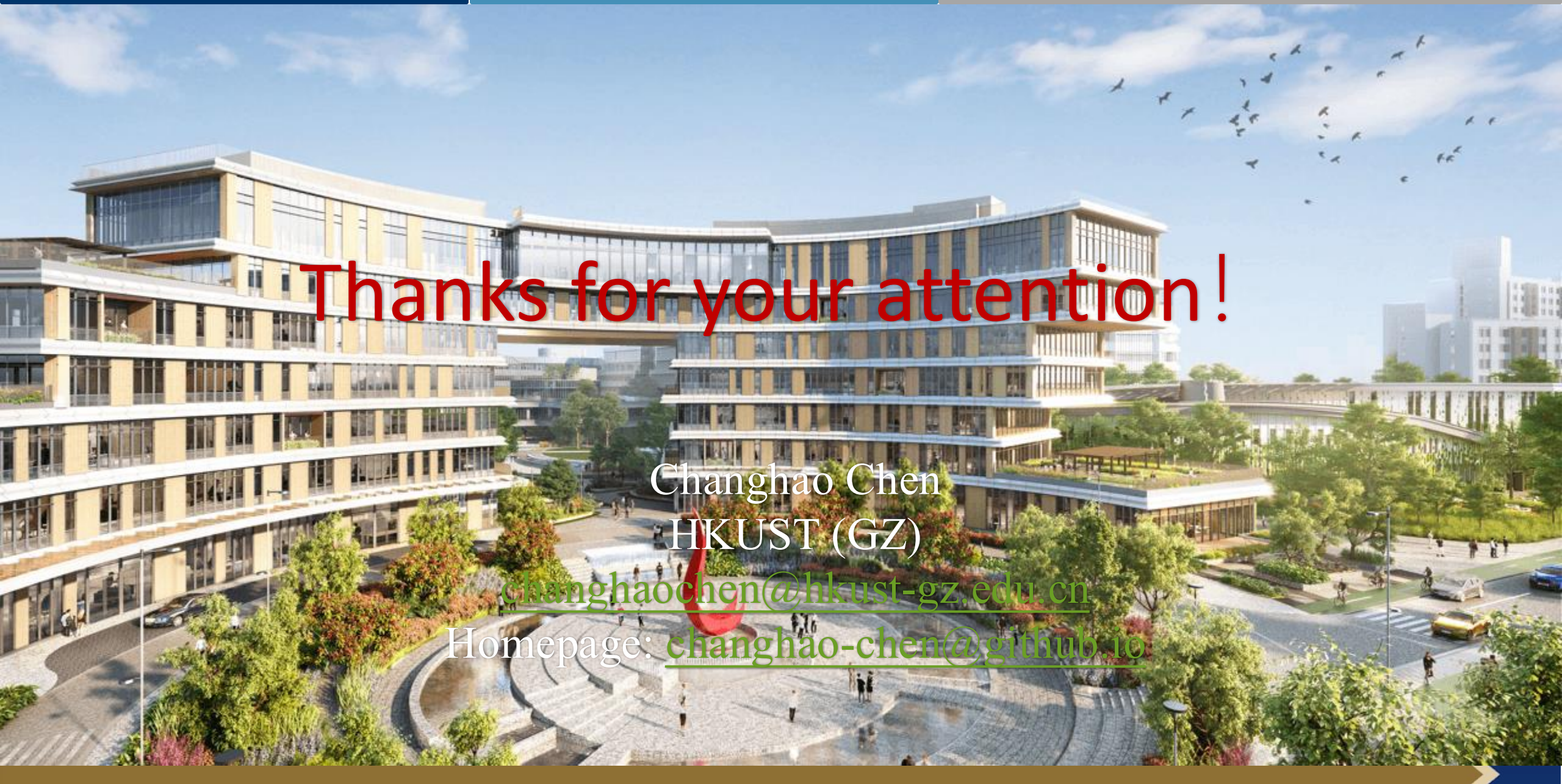
1. Randomly select a minimal sample of 3 points.

2. Compute pose hypotheses using the P3P algorithm on this sample.

3. Test each hypothesis on all other points: Calculate the reprojection error for each point; Points with error below a threshold are inliers.

4. Repeat for many iterations. Keep the hypothesis with the most inliers.

5. Refine the winning pose using all inliers with a non-linear optimizer (like Levenberg-Marquardt) minimizing the reprojection error.



Thanks for your attention!

Changhao Chen
HKUST (GZ)
changhaochen@hkust-gz.edu.cn
Homepage: changhao-chen@github.io