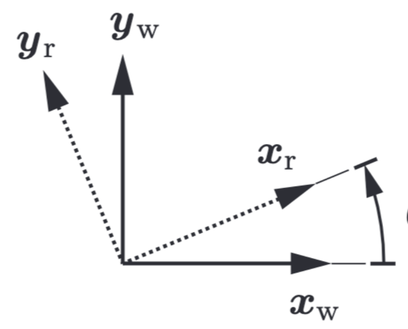
Feature Detection and Matching

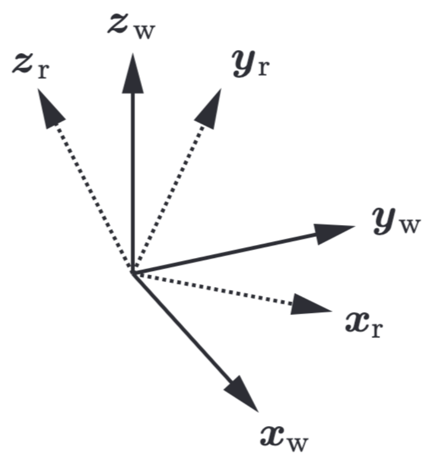
Graduate Course INTR-6000P

Week 3 - Lecture 5

Changhao Chen
Assistant Professor
HKUST (GZ)

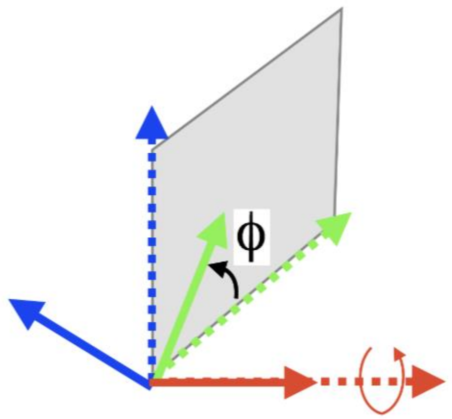
Recap: Euler Angles and Rotation Matrix





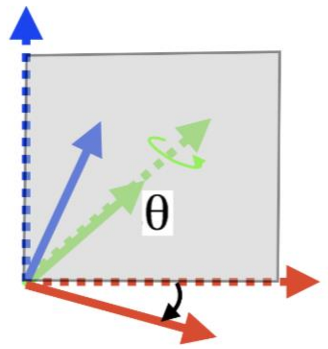
$$R_r^w = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin\theta & \cos(\theta) \end{bmatrix}$$

Rotation around the **x** axis



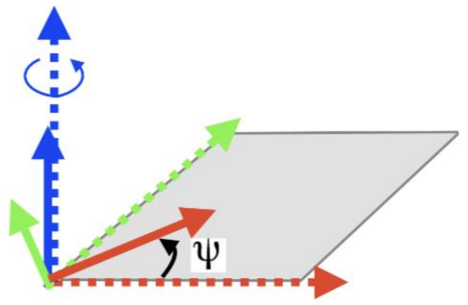
$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

Rotation around the **y** axis



$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Rotation around the **z** axis



$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Euler's Theorem implies that only three parameters (Euler angles) are needed to represent a 3D rotation.

A rotation matrix is a special matrix used to rotate vectors in a Euclidean space (like 2D or 3D) around the origin (the point (0,0)) by a given angle.

$$R_r^w = R_x(\phi_1)\, R_y(\theta_1)\, R_x(\psi_1)$$

Recap: Camera Pose

Defining Pose: Position and Orientation

The pose of a body frame {r} relative to a world frame {w} is completely described by:

Position: $\mathbf{t}_r^w$ - The translation from the origin of {w} to the origin of {r}.

Orientation: $\mathbf{R}_r^w$ - The rotation from {r} to {w}.

The pair $(\boldsymbol{R}_{\mathrm{r}}^{\mathrm{w}}, \boldsymbol{t}_{\mathrm{r}}^{\mathrm{w}})$ is the complete geometric representation of frame {r} in {w}.

A pose has 6 degrees of freedom (3 for translation, 3 for rotation). We can combine these into a single, convenient transformation matrix.

$$\boldsymbol{T}_{\mathrm{r}}^{\mathrm{w}} = \begin{bmatrix} \boldsymbol{R}_{\mathrm{r}}^{\mathrm{w}} & \boldsymbol{t}_{\mathrm{r}}^{\mathrm{w}} \\ \mathbf{0}_d^\top & 1 \end{bmatrix}$$

Recap: Rigid-body Transformations

We now generalize from pure rotation to full pose (rotation and translation).

Given:

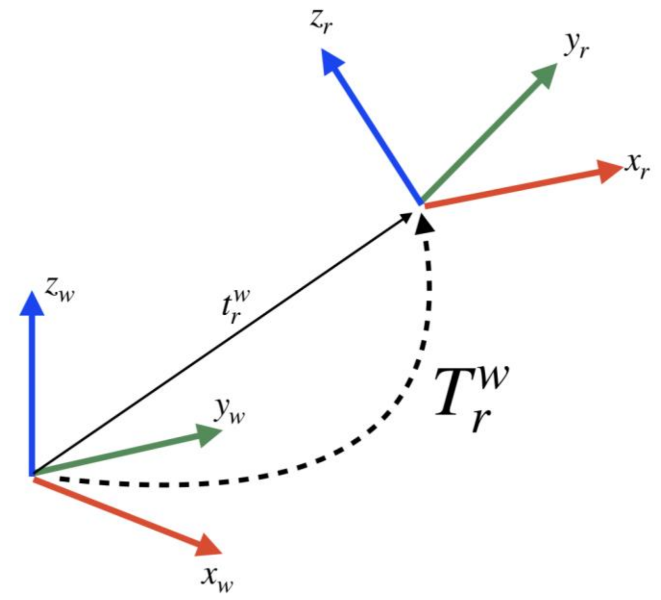
A point's coordinates in frame {r}: $\mathbf{p}^\mathrm{r}$

The pose of {r} relative to {w}, represented by the transformation matrix: $\mathbf{T}_\mathrm{r}^\mathrm{w}$

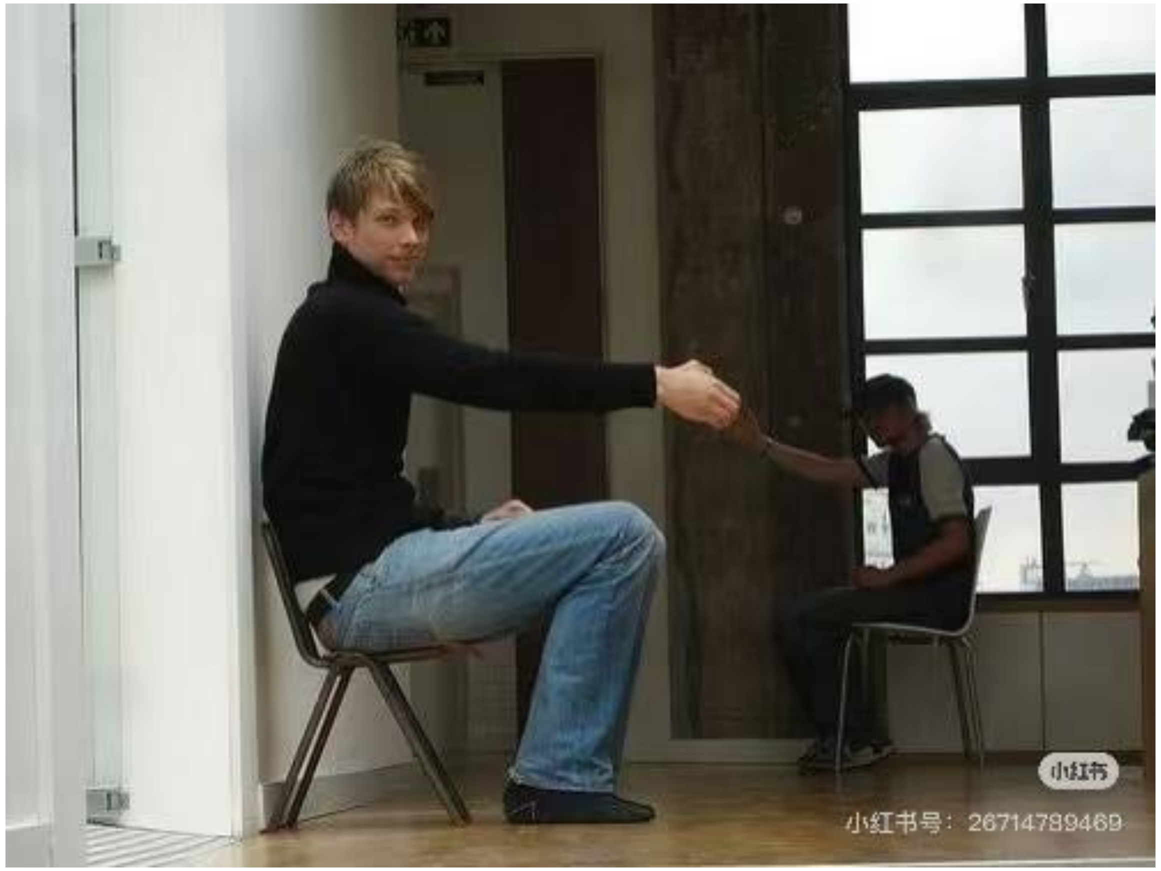
Compute: The point's coordinates in frame {w}: $\mathbf{p}^\mathrm{w}$

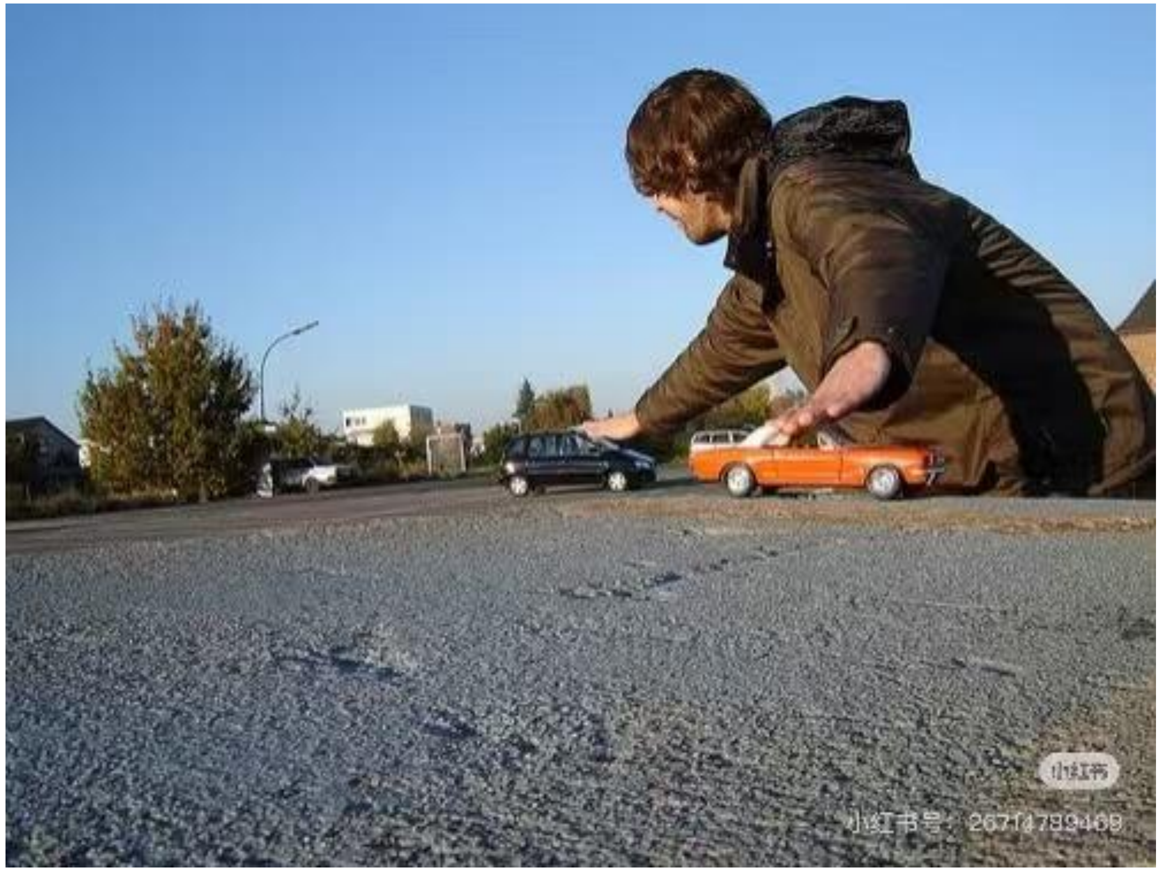
The transformation is given by:

$$\boldsymbol{p}^\mathrm{w} = \boldsymbol{R}_\mathrm{r}^\mathrm{w} \boldsymbol{p}^\mathrm{r} + \boldsymbol{t}_\mathrm{r}^\mathrm{w}$$

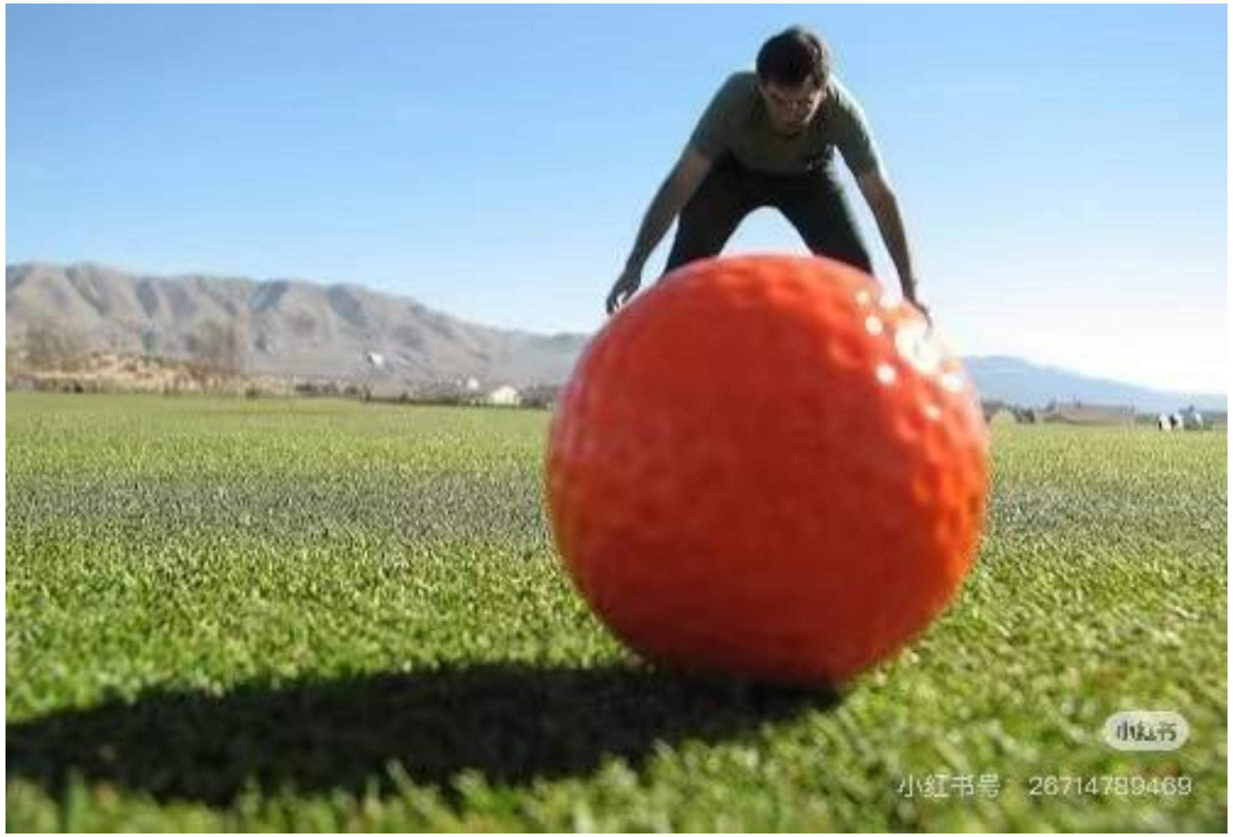


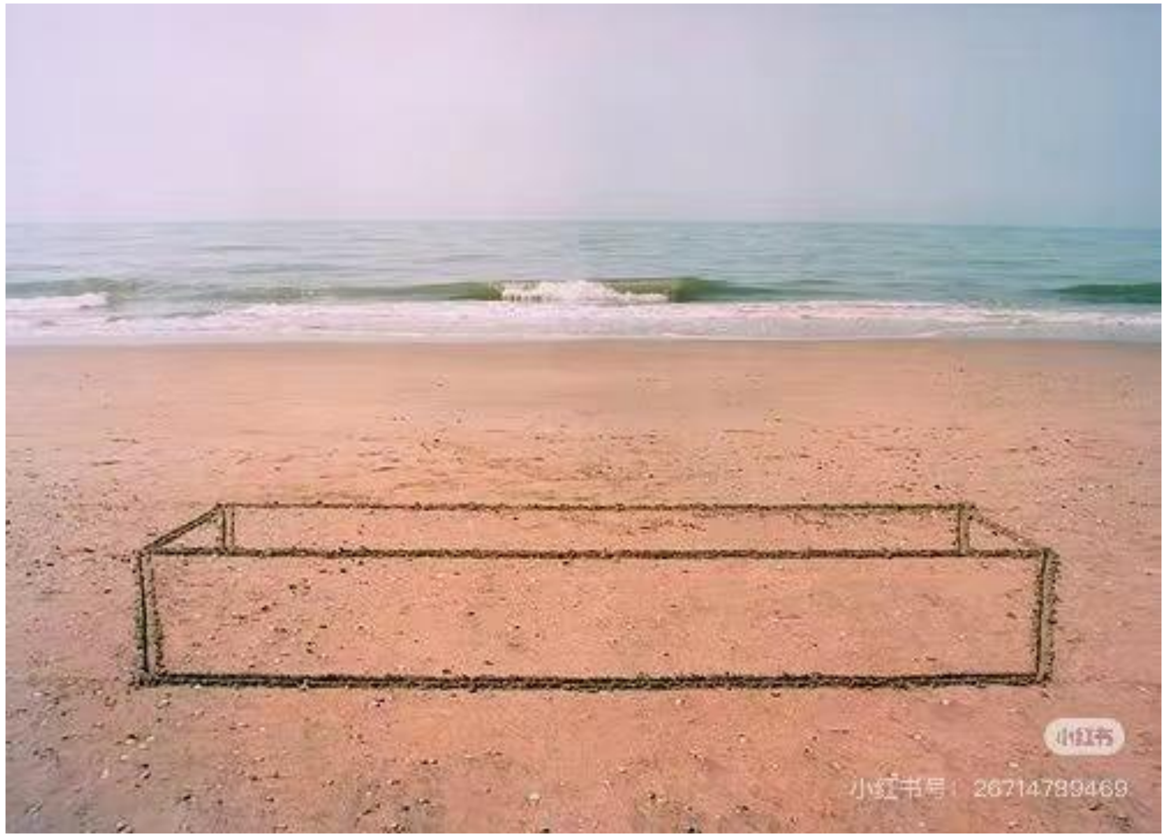
Recap: Some Photos





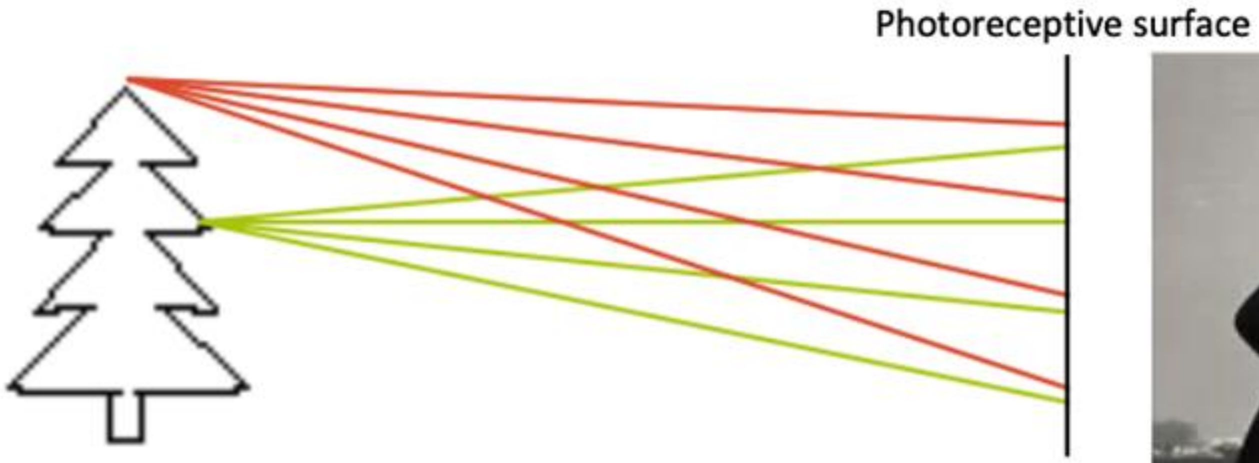
Recap: Some Photos

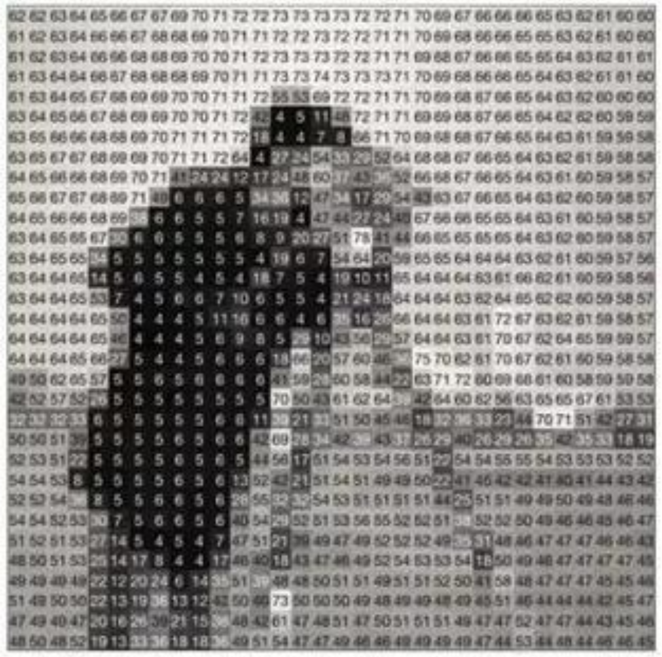




Recap: Digital Cameras

- How to capture an image of the world
  - (Digital) Cameras -> capture light -> converted to digital image
  - Light is reflected by the object and scattered in all directions
  - if we simply add a photoreceptive surface, the captured image will be extremely blurred



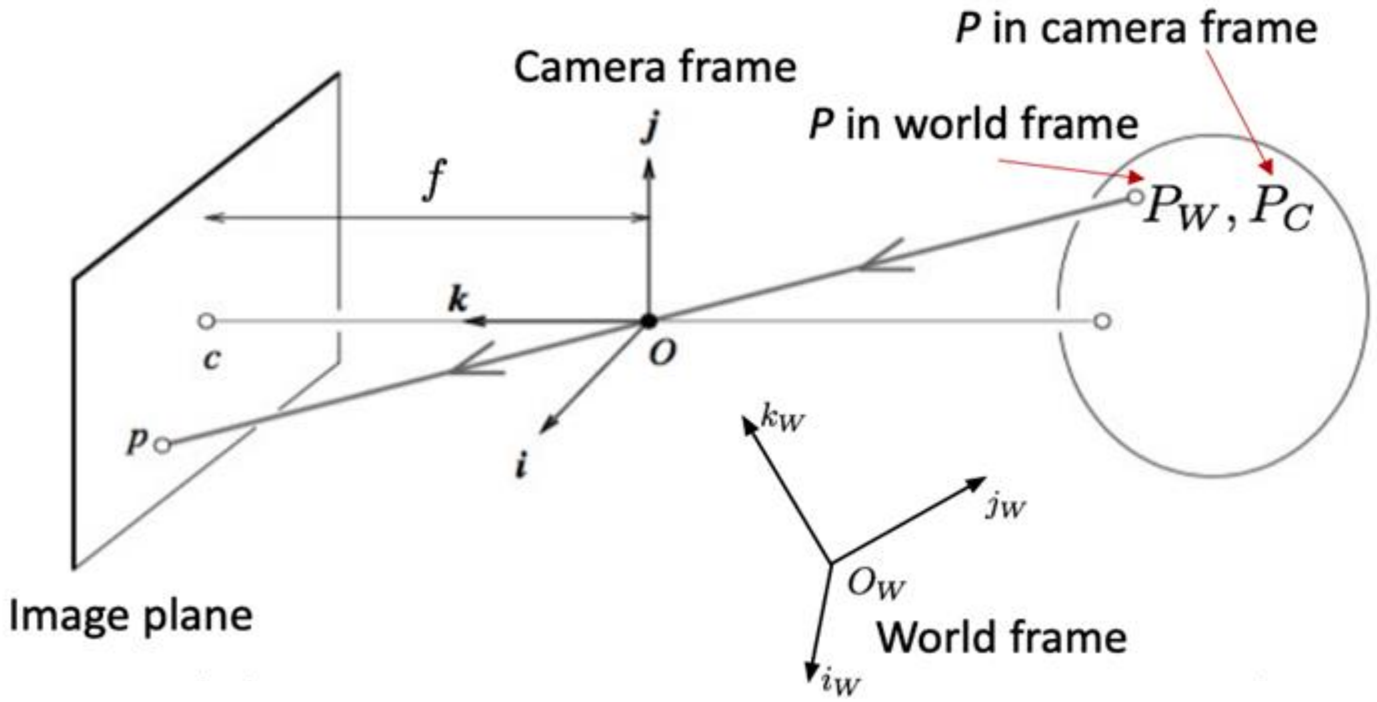


Recap: Perspective Projection

1) World to Camera Transformation (P_w → P_c)
Transform the point from world coordinates to camera coordinates using the camera's pose (rotation R and translation t).

2) Perspective Projection (P_c → p)
Project the 3D point in the camera frame onto the normalized image plane using the pinhole model: (x, y) = (fX/Z, fY/Z).

3) Image to Pixel Transformation (p → (u, v))
Convert the metric image coordinates into discrete pixel coordinates using the camera's intrinsic calibration matrix K.



P_w and P_c represent the same physical point in 3D space, but their coordinate values differ because they are expressed in different reference frames.

Recap: Homogeneous coordinates

- Collecting all results

$$p^h = [K \quad 0_{3\times1}]P_C^h = K[I_{3\times3} \quad 0_{3\times1}]\begin{bmatrix} R & t \\ 0_{1\times3} & 1 \end{bmatrix} P_W^h$$

- Hence

$$\boxed{p^h = \overbrace{\underbrace{K}_{\text{Intrinsic parameters}}[\underbrace{R \quad t}_{\text{Extrinsic parameters}}]}^{\text{Projection matrix } M} P_W^h}$$

R: rotation
t: translation
Rt: extrinsic matrix

Degree of freedom:
K: 4
Rt: 6

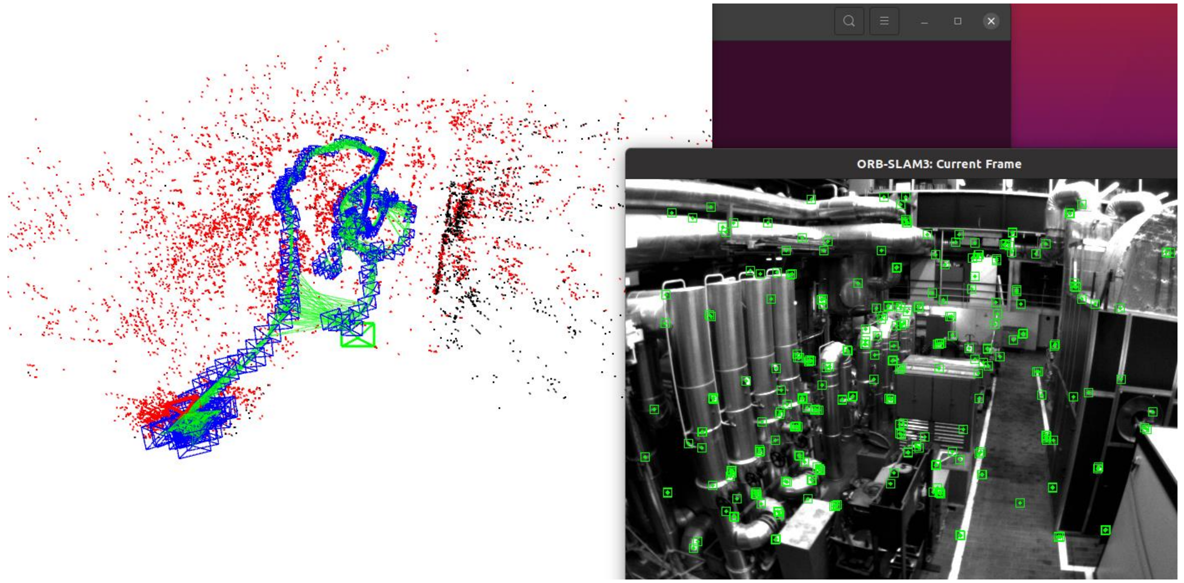
Visual Feature

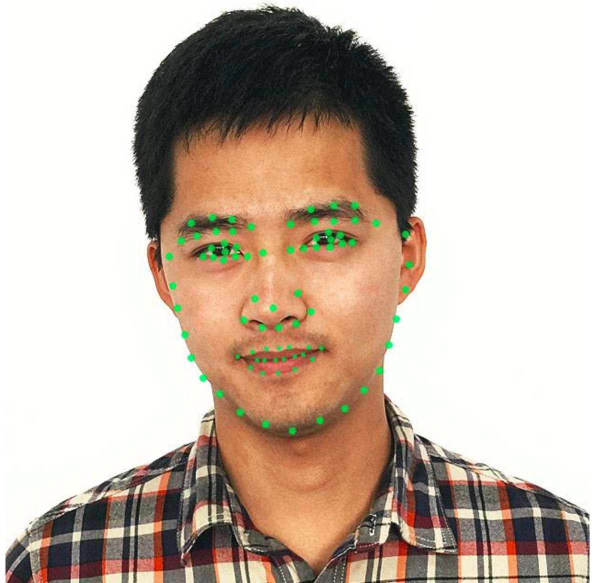
Features are key information in an image (such as corner points, edges, etc.).

Distinct, recognizable patterns

They are the foundation for:

- Visual Odometry / SLAM: Estimating ego-motion.
- Object Recognition & Tracking: Following other cars, pedestrians.
- 3D Reconstruction: Building a map of the world.

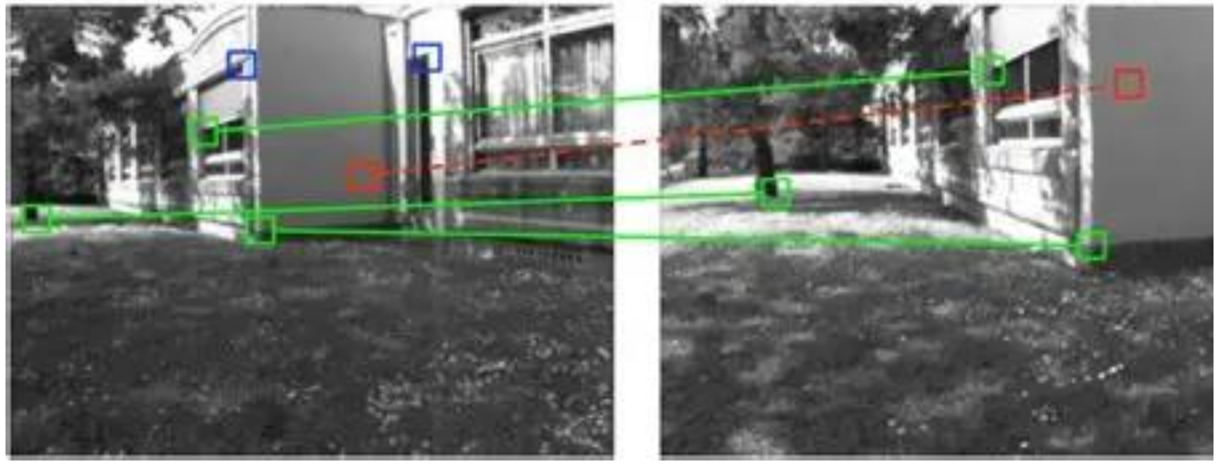




Visual Feature

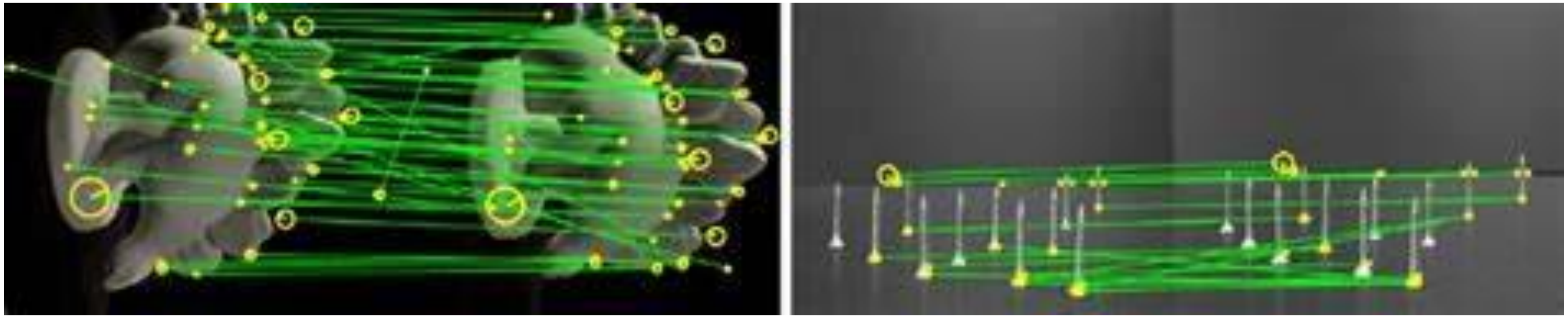
What Makes a Good Feature?

- Repeatability: Can be found again in a different image of the same scene.
- Distinctiveness: Carries enough information to be distinguished from others.
- Locality: Robust to occlusion and clutter.
- Efficiency: Can be computed quickly.



Challenges:
Scale changes, viewpoint changes, illumination changes (day/night), motion blur, repetitive textures (e.g., a brick wall), real-time computation requirements.

Visual Feature

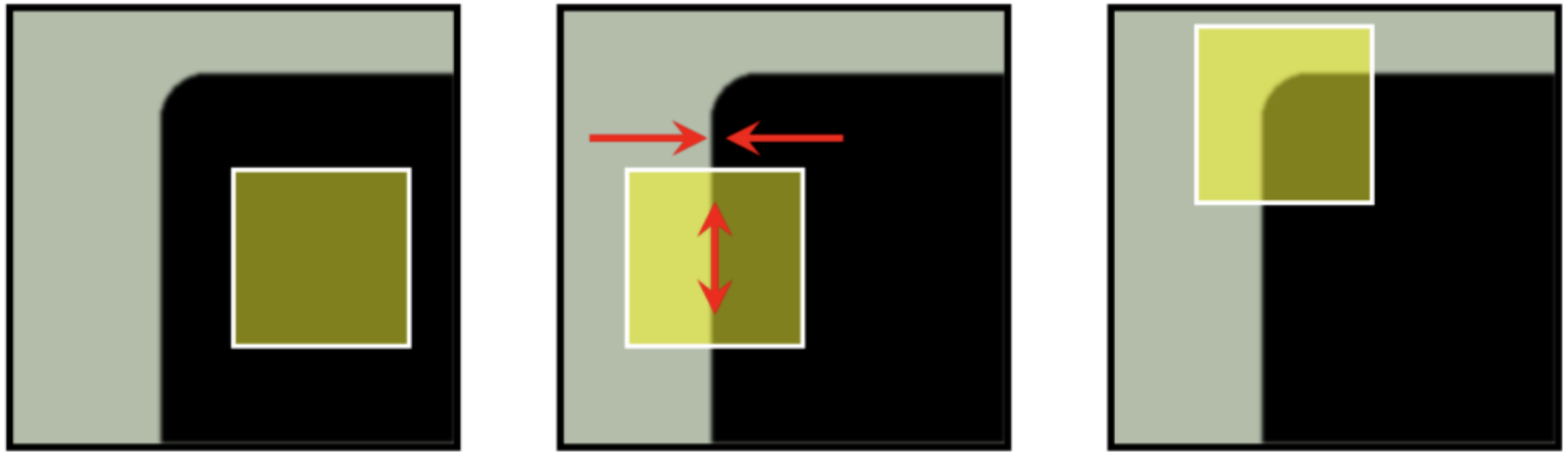


Keypoint (Location): The (x, y) coordinates of the feature.
Descriptor (Signature): A numerical vector that describes the visual appearance of the patch around the keypoint.

Main Tasks:

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

Feature Detection

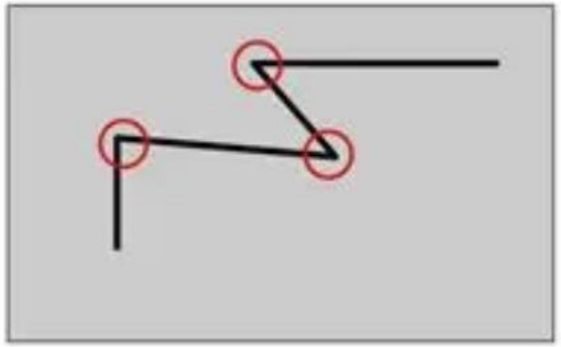


Flat: no change in all directions

Edge: no change along the edge direction

Corner: significant change in all directions with small shift

Corner Detection



A corner is a point where the image intensity changes significantly in multiple directions.

Uses a sliding window and looks for large changes in intensity in all directions.

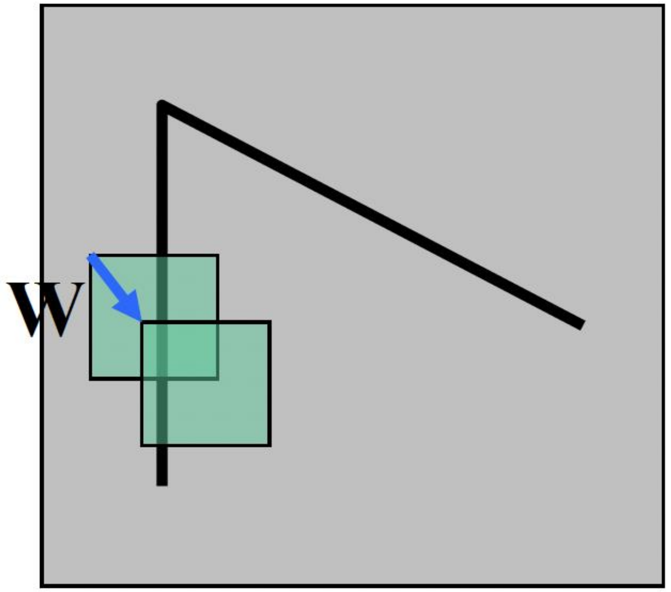
Simple, rotation-invariant.

But not scale-invariant.

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.

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

Corner Detection

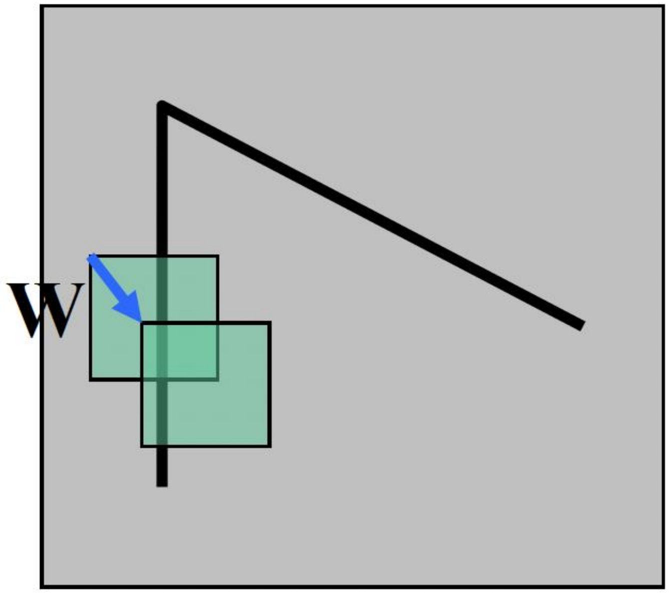
1) We apply a Taylor expansion to approximate the change function, E(u,v):

$$E(u,v) \approx [u \quad v] \left(\sum \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

2) Introduce the Structure Tensor (M): This simplification leads us to a 2x2 matrix M, which summarizes all gradient information within the window.

$$M = \sum w(x,y) \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix}$$

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$



3) Eigenvalue Analysis: The eigenvectors of M show the directions of strongest and weakest intensity change. Their corresponding eigenvalues (λ_1, λ_2) quantify the magnitude of these changes.

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

4) Calculate the Corner Response (R): Instead of checking every direction, we compute a single score R from λ_1 and λ_2. A high R value indicates a true corner.

Corner Detection

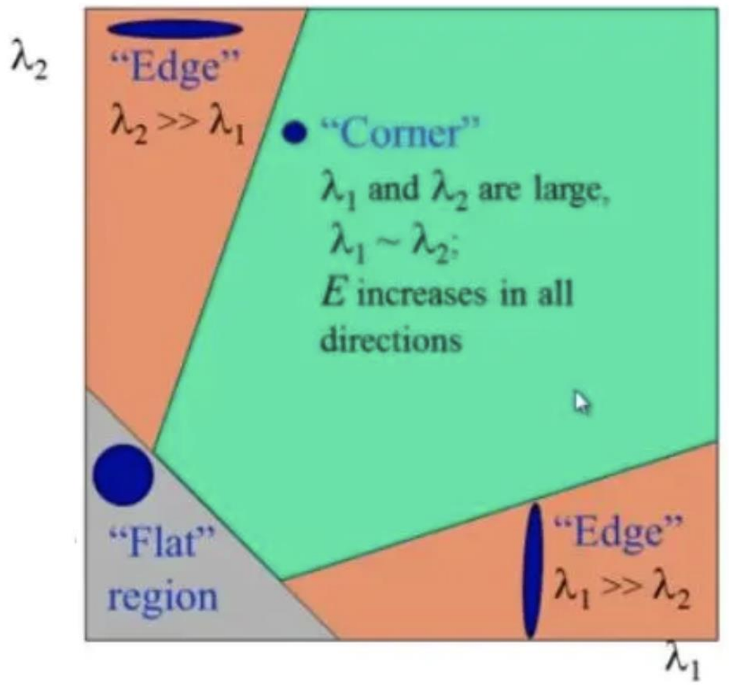
$$R = \det M - k(\mathit{trace}\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$\mathit{trace}\, M = \lambda_1 + \lambda_2$$

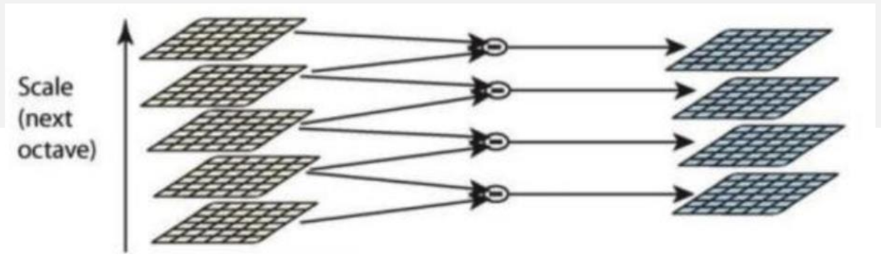
The corner response score, R, allows us to classify image regions based on the eigenvalues λ_1 and λ_2:

When $|R|$ is small, which happens when $\lambda 1$ and $\lambda 2$ are small, the region is flat.
•When R<0, which happens when $\lambda 1 >> \lambda 2$ or vice versa, the region is an edge.
•When R is large, which happens when $\lambda 1$ and $\lambda 2$ are large and $\lambda 1 \sim \lambda 2$, the region is a corner.

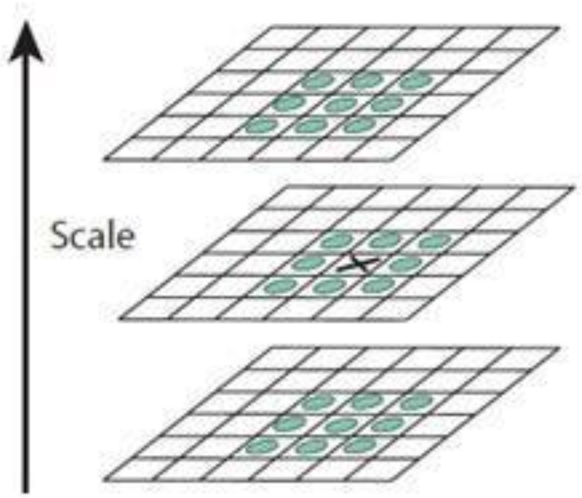


The Harris Corner Detector

Scale-Invariant Detection



SIFT(Scale Invariant Feature Transform)



1. Scale-Space Extrema Detection

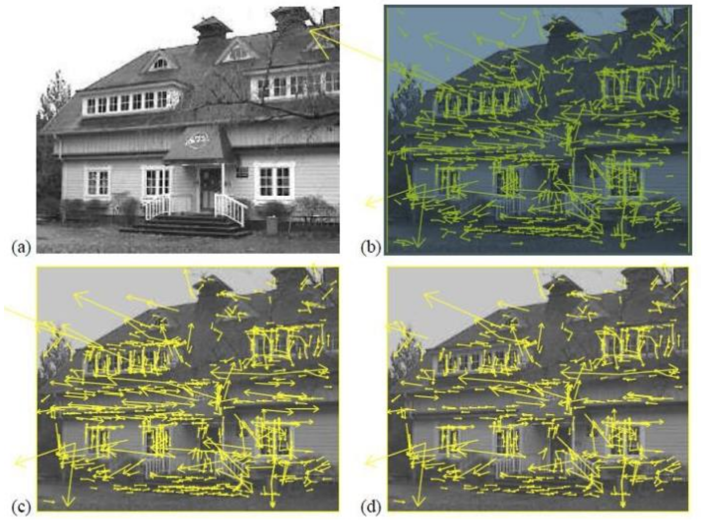
Find locations and scales that are identifiable across different views of the same object.

Details: SIFT searches for stable features across a continuous scale space, which is represented by a Gaussian pyramid. It uses a Difference-of-Gaussians (DoG) function to efficiently identify potential keypoints that are stable across scales. This ensures the features are scale-invariant.

2. Keypoint Localization

Refine the candidate keypoints and filter out unstable ones (e.g., low-contrast points or points along edges, which are sensitive to noise).

Details: The algorithm fits a model to nearby data for accurate location, scale, and ratio of principal curvatures.
This step ensures that only the most distinctive and stable keypoints are retained.

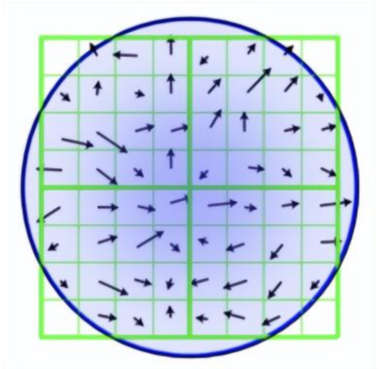


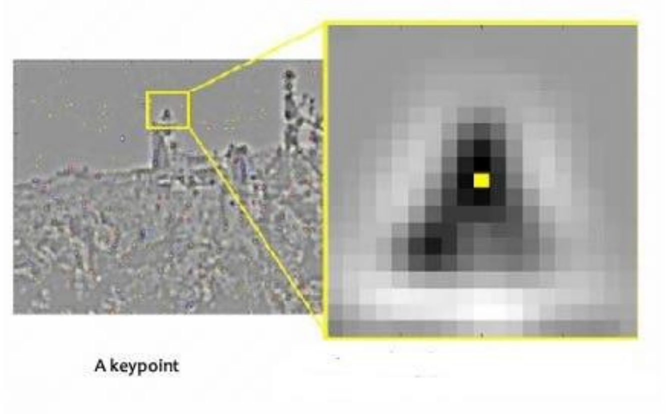
Scale-Invariant Detection

3. Orientation Assignment

Achieve rotation invariance by assigning a consistent orientation to each keypoint based on local image properties.

Details: For the pixels around each keypoint, SIFT computes the gradient magnitude and direction. It creates a histogram of these directions, and the dominant orientation(s) of the local gradient are assigned to the keypoint. All future operations are performed relative to this orientation, making the descriptor robust to image rotation.

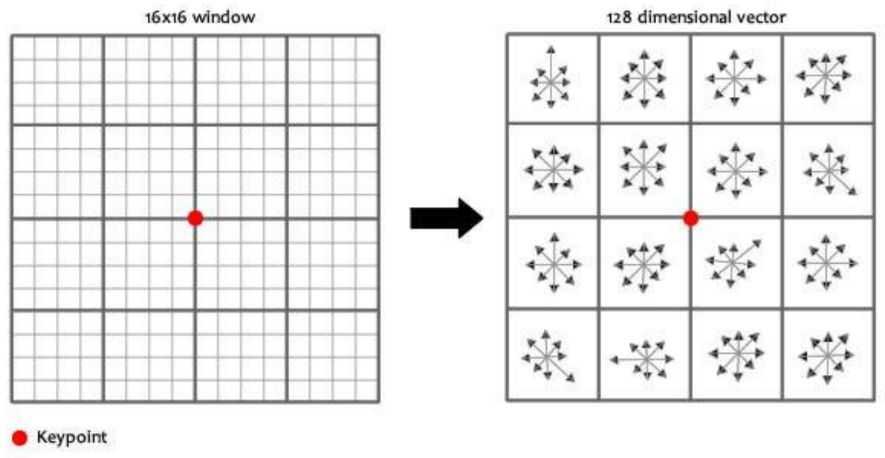




4. Keypoint Descriptor Generation

Create a highly distinctive and robust numerical representation (a "fingerprint") for the local image patch around the keypoint.

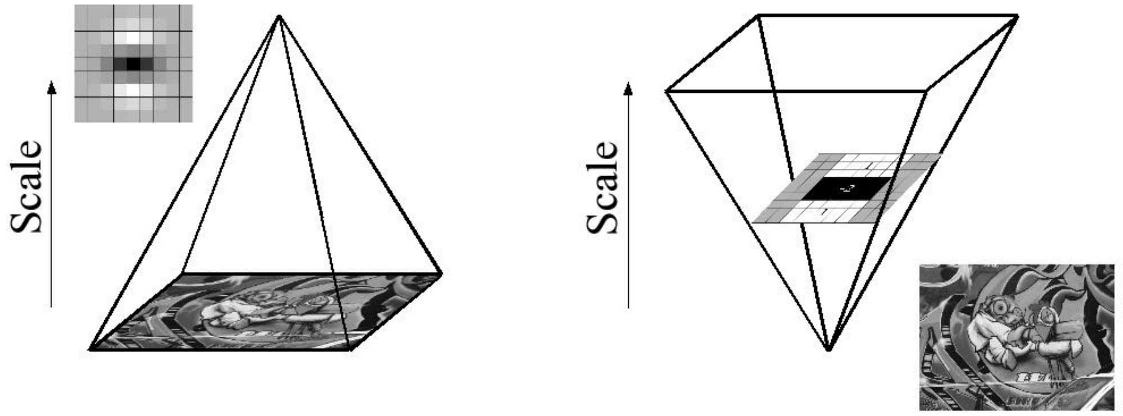
Details: The algorithm samples the local image gradients around the keypoint, aligned to its assigned orientation. It then creates a set of orientation histograms over a 4x4 grid of sub-regions, resulting in a 128-element feature vector (8 orientation bins x 4x4 grid). This vector is normalized to reduce the effects of changes in illumination.



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.

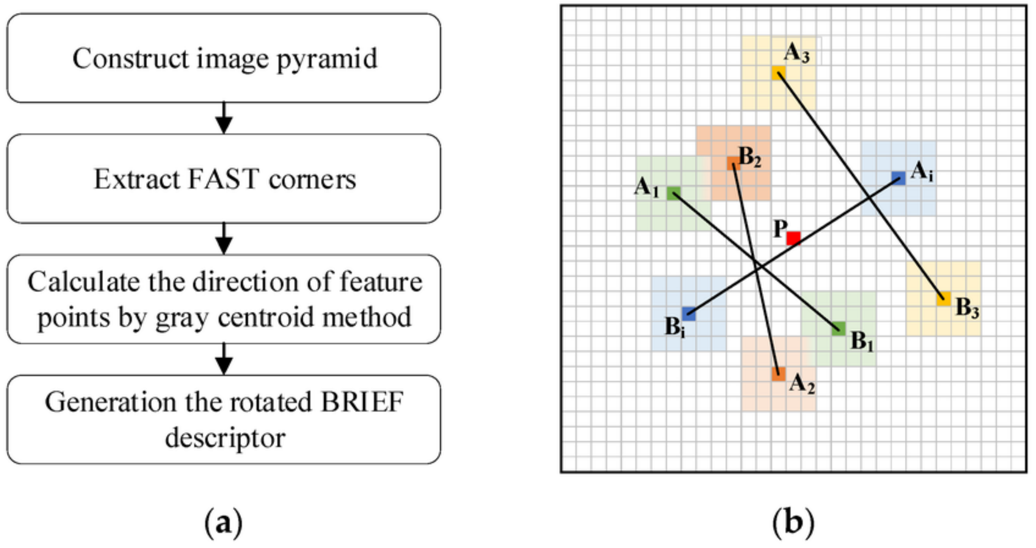


ORB (Oriented FAST and Rotated BRIEF):

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.


Construct image pyramid
Extract FAST corners
Calculate the direction of feature points by gray centroid method
Generation the rotated BRIEF descriptor
(a)
A_3
B_2
A_1
A_i
P
B_i
B_1
B_3
A_2
(b)

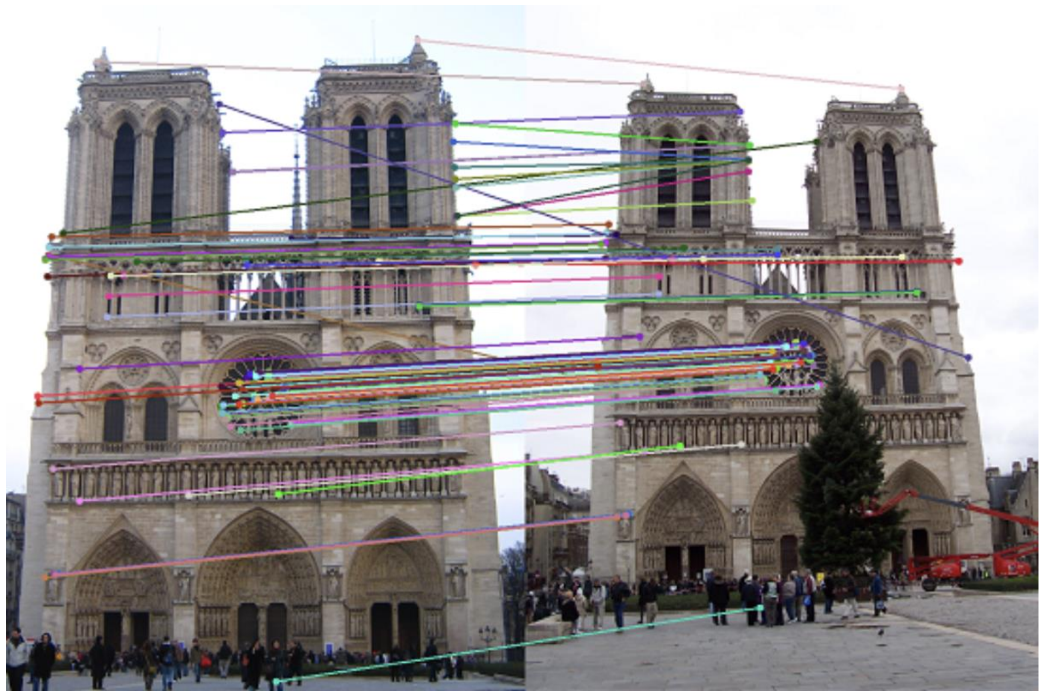
Feature Matching

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.



Feature Matching

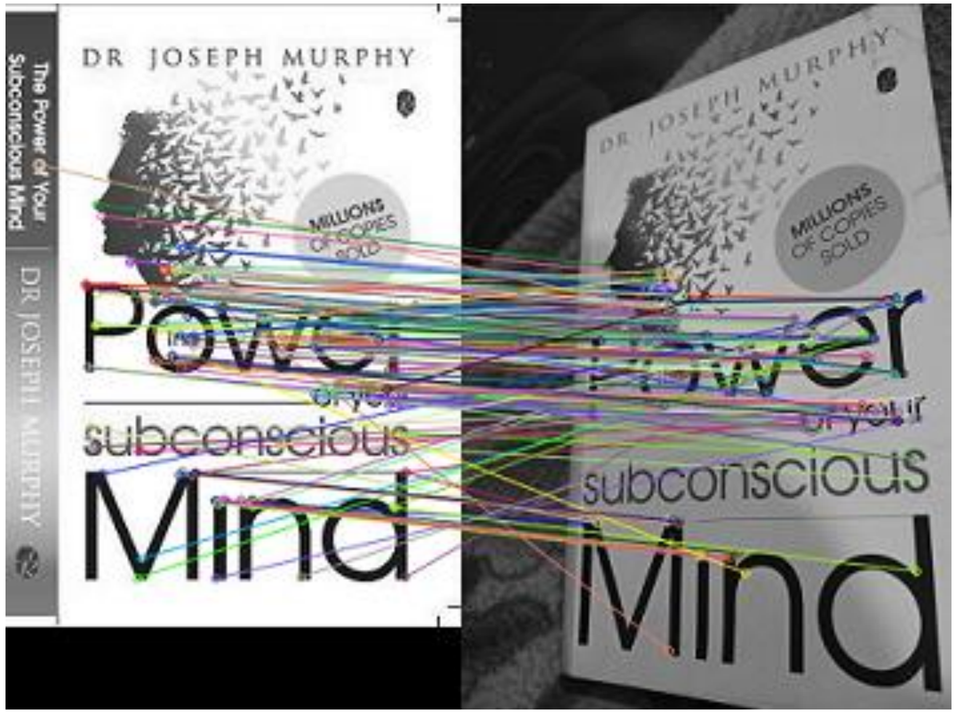
OpenCV provides BFMatcher and FlannBasedMatcher.

Brute-Force Matcher (BFMatcher):

- For every single descriptor in the first image, the Brute-Force matcher compares it against every single descriptor in the second image.
- It calculates a distance (e.g., Euclidean distance for SIFT, Hamming distance for ORB) for every possible pair.
- The best match is the one with the smallest distance.

Pros: Simple and guaranteed to find the absolute best match (the nearest neighbor).

Cons: Computationally expensive (slow) for large numbers of features. The time complexity is O(N*M), which becomes impractical for large datasets.

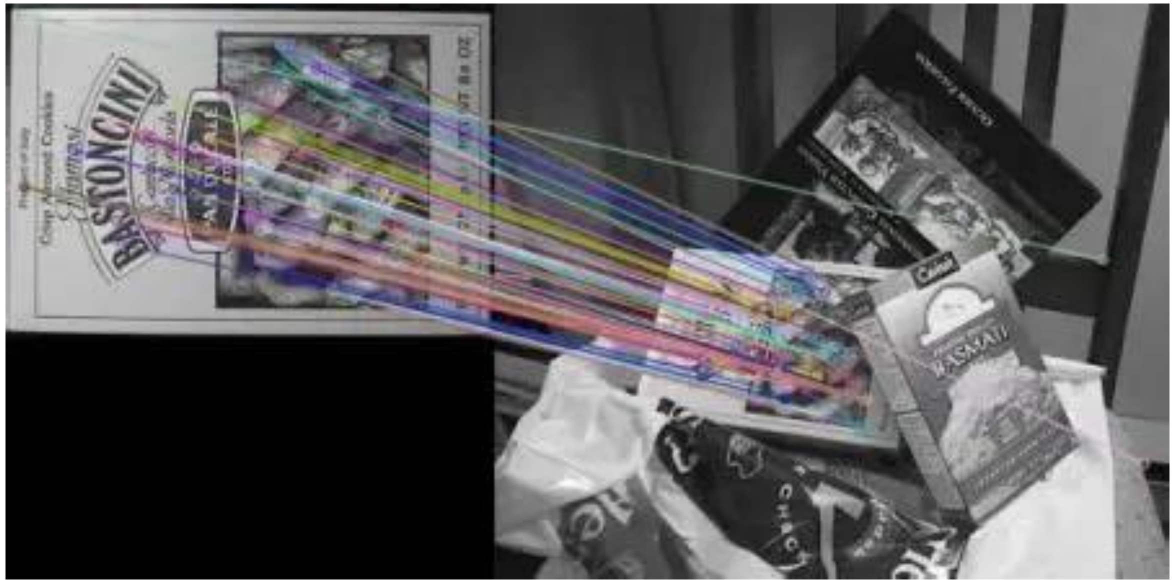


Feature Matching

Fast Library for Approximate Nearest Neighbors (FLANN)

FLANN uses optimized data structures (like KD-Trees) to intelligently organize the descriptor data, allowing for much faster searches.

- It pre-processes the dataset of descriptors from the second image into an efficient search structure.
- Instead of comparing every pair, it performs an optimized, approximate search to find the closest neighbors.



Pros: Extremely fast, often orders of magnitude faster than Brute-Force for large feature sets. Essential for real-time applications.

Cons: It finds approximate nearest neighbors; there is a small chance it might not find the absolute best match, though it is almost always correct.

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.

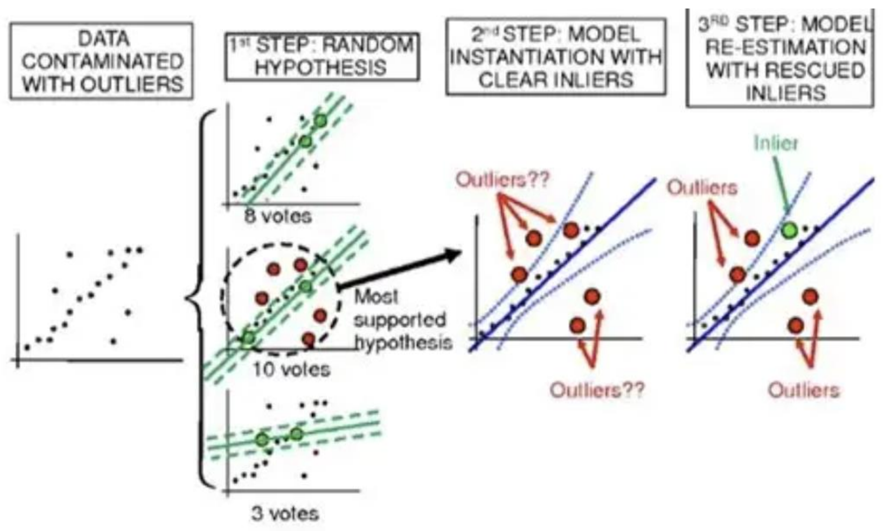
1) Random Sampling: Randomly select the smallest number of points needed to define the model (e.g., 4 points to estimate a homography).

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

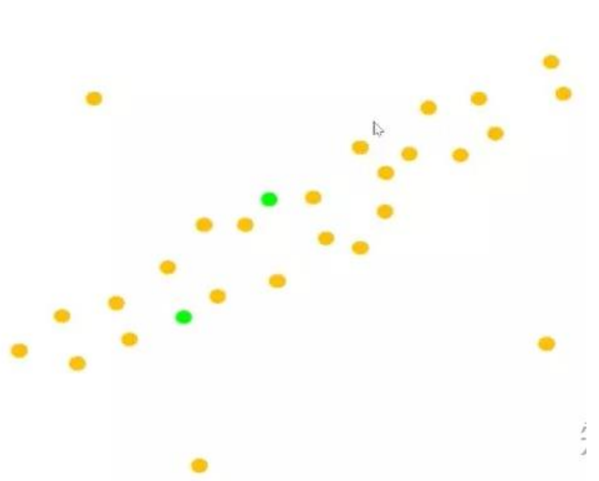
3) Consensus (Voting): Test all other data points against this model. Points that fit the model well within a tolerance are considered inliers and form the "consensus set."

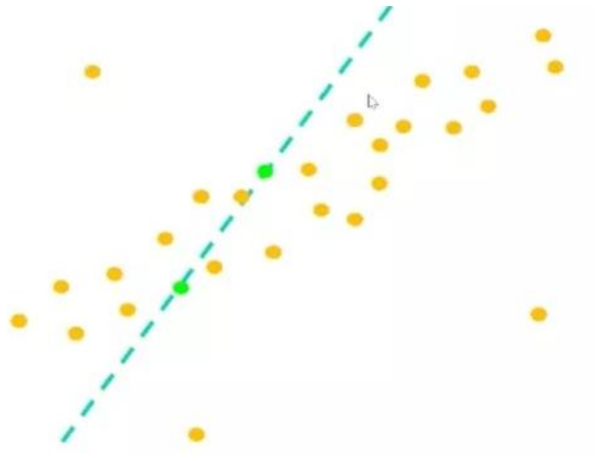
4) Iterate & Select: Repeat this process many times. The model with the largest consensus set (the most inliers) is chosen as the best model.

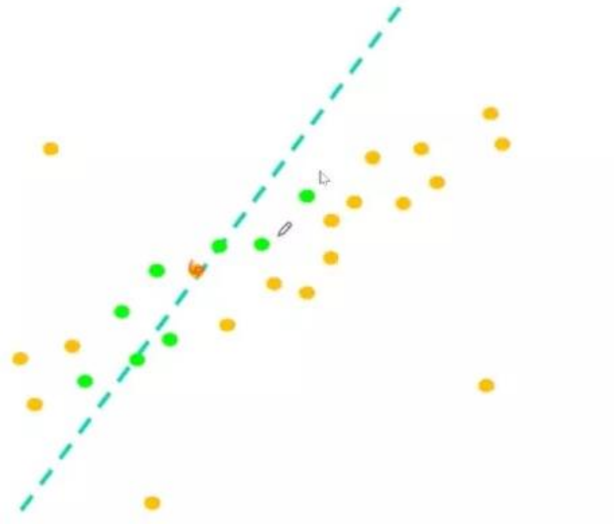
5) Re-estimate: Finally, the model parameters are re-estimated using all the inliers in the best consensus set for a robust fit.

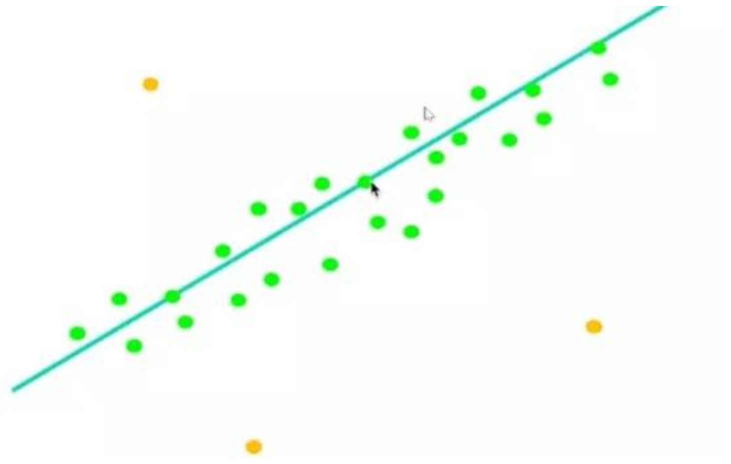

DATA CONTAMINATED WITH OUTLIERS
1st STEP: RANDOM HYPOTHESIS
2nd STEP: MODEL INSTANTIATION WITH CLEAR INLIERS
3RD STEP: MODEL RE-ESTIMATION WITH RESCUED INLIERS
8 votes
10 votes
3 votes
Most supported hypothesis
Outliers??
Outliers??
Inlier
Outliers
Outliers

Feature Matching

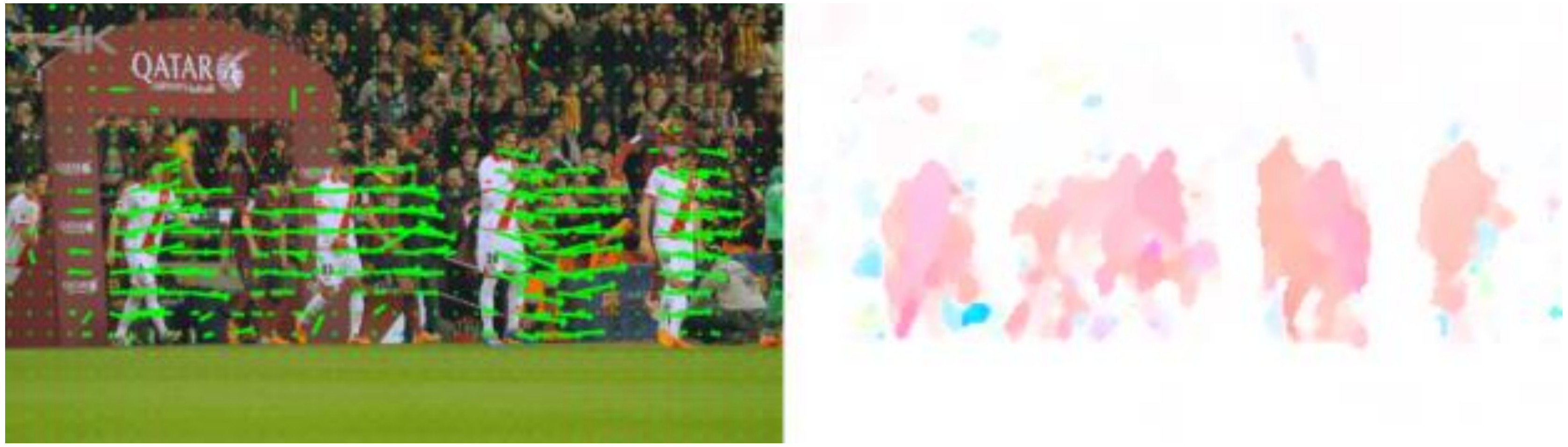

1) Random Sampling


2) Model Estimation


3) Consensus (Voting)


4) Iterate & Select

Feature Tracking: The Optical Flow



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

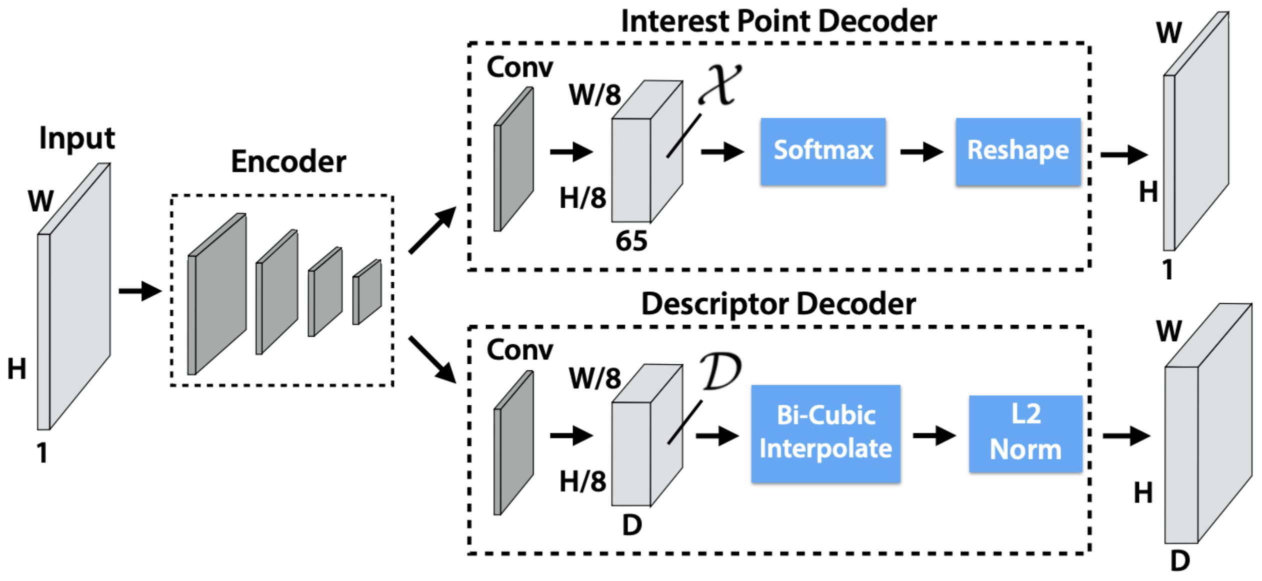
Tracking: Following a feature through a sequence of images (a video from a single camera).

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.

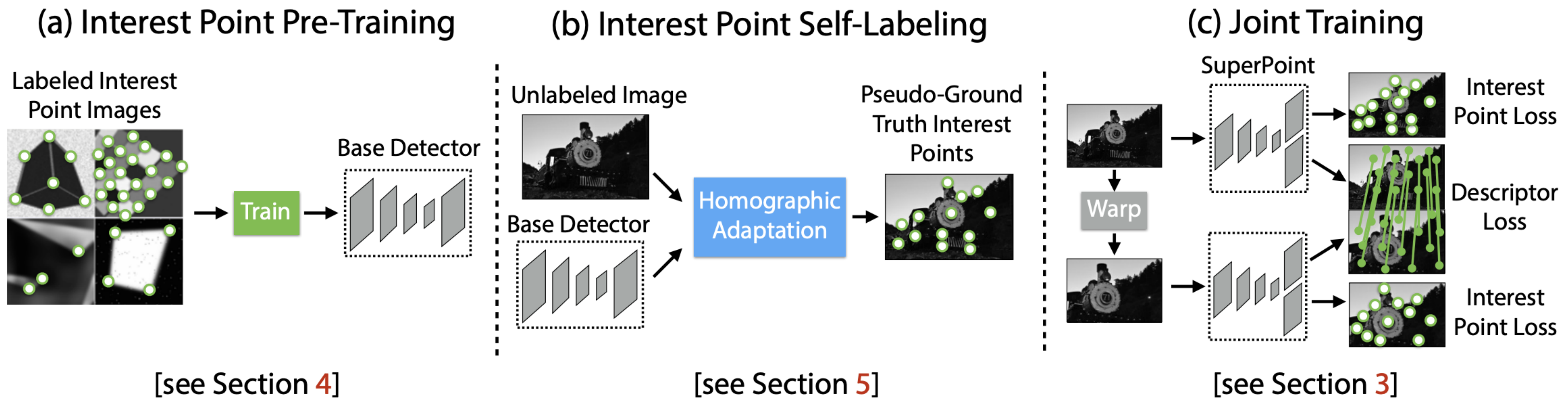
Lucas-Kanade Method: A classic, efficient algorithm for sparse optical flow (tracking specific feature points).

Deep Learning Enhanced Solutions



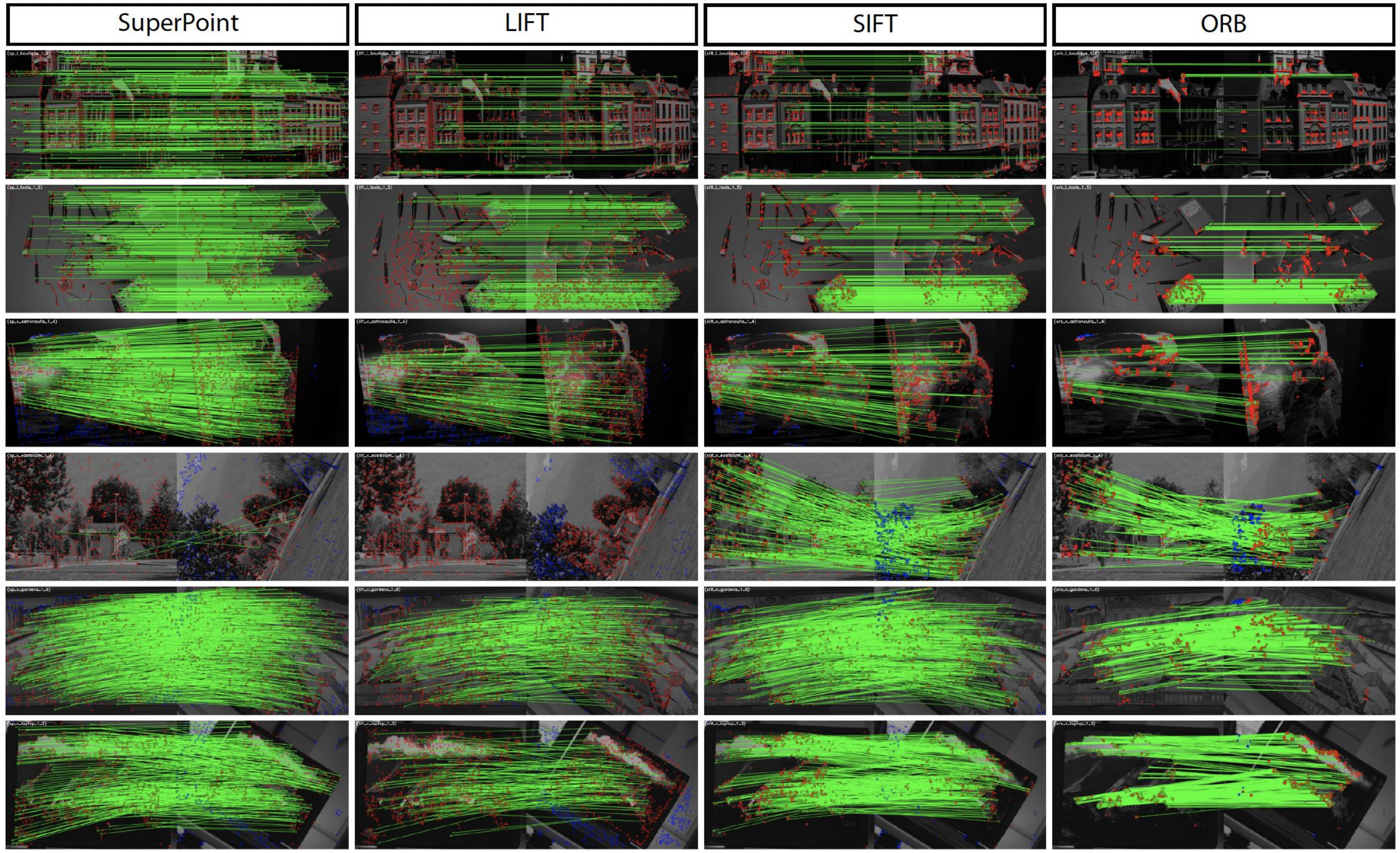
Feature Detection: SuperPoint （2018）

$$\mathcal{L}(\mathcal{X},\mathcal{X}',\mathcal{D},\mathcal{D}';Y,Y',S) = \mathcal{L}_p(\mathcal{X},Y)+\mathcal{L}_p(\mathcal{X}',Y')+\lambda\mathcal{L}_d(\mathcal{D},\mathcal{D}',S).$$

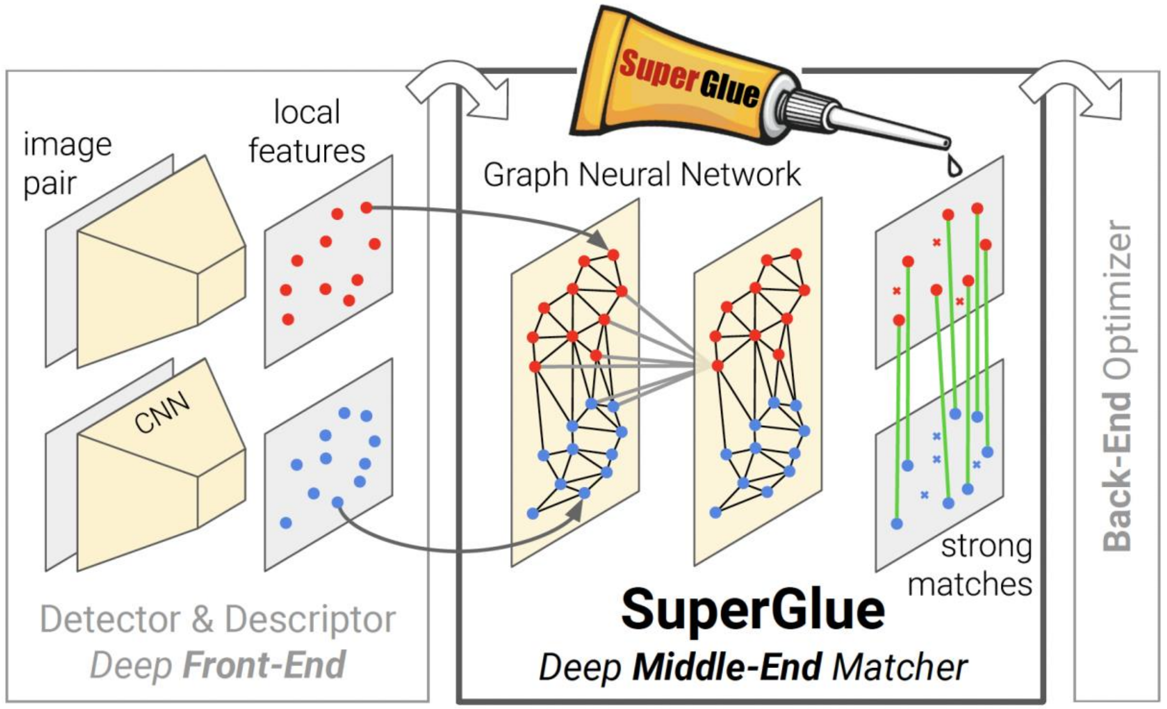


Deep Learning Enhanced Solutions

Feature Detection: SuperPoint （2018）



Deep Learning Enhanced Solutions

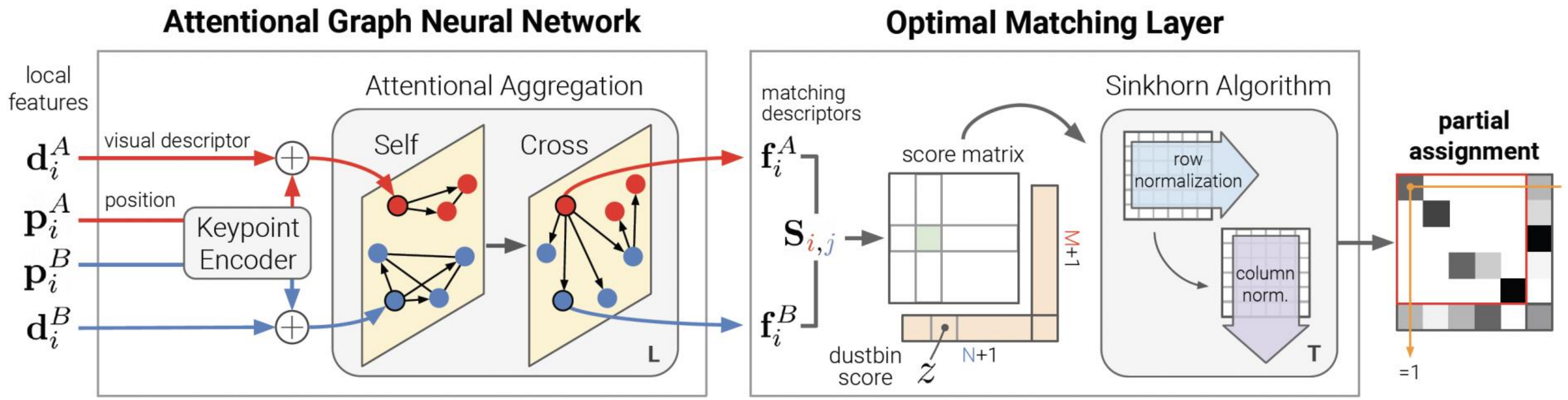


Feature Matching: SuperGlue （2020）

Input: feature points and descriptors.

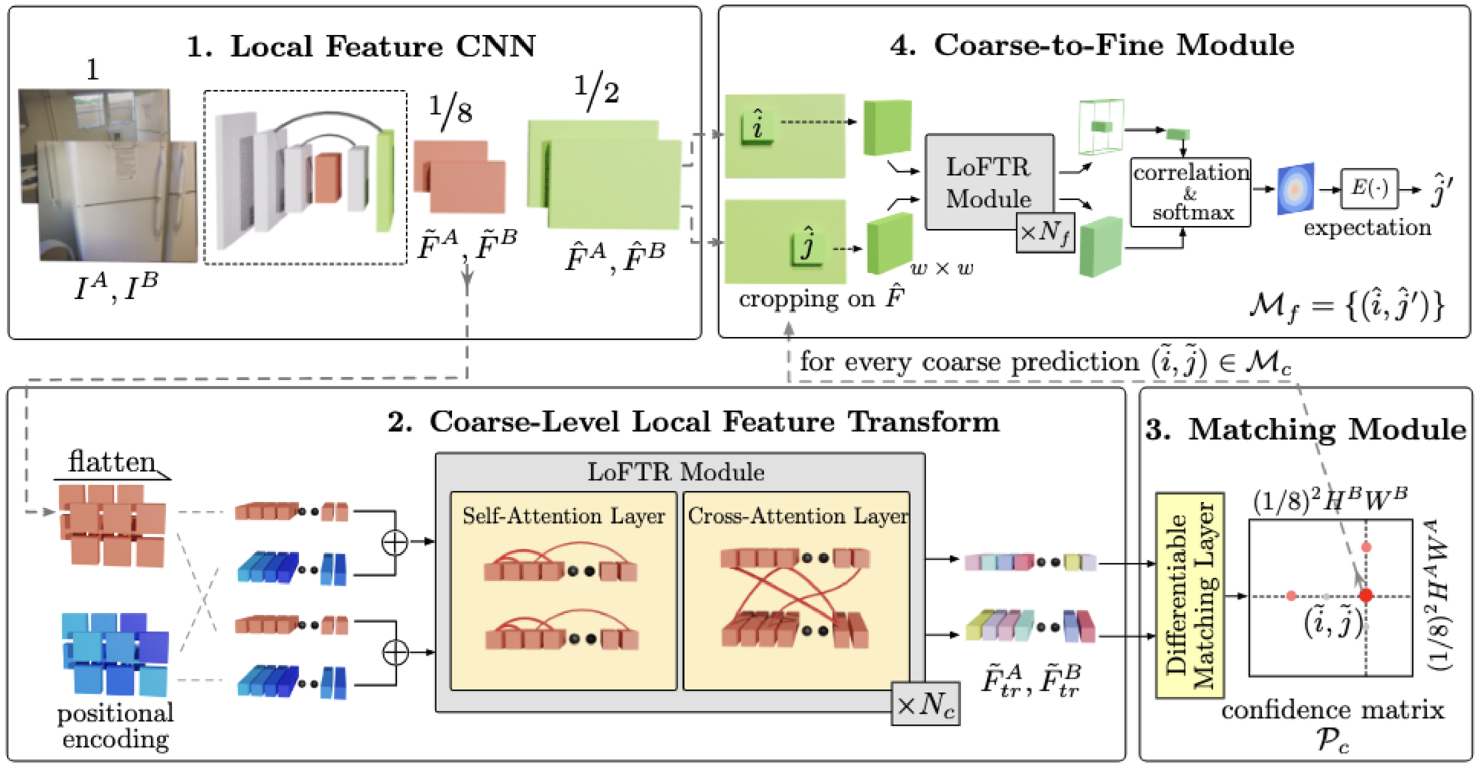
Output: the degree of feature association between two images.

- Feature Association via Optimal Transport:
- Input keypoints and descriptors; output inter-image feature correlations.
- Framework: Attentional GNN + optimal matching layer.
- Attentional GNN encodes keypoints/descriptors into matching vectors via self/cross-attention.
- Optimal matching computes scores via inner products and solves assignment with Sinkhorn.



Deep Learning Enhanced Solutions

Feature Detection and Matching: LoFTR（2021）



1) Feature Extraction: A CNN extracts coarse and fine feature maps from both images.

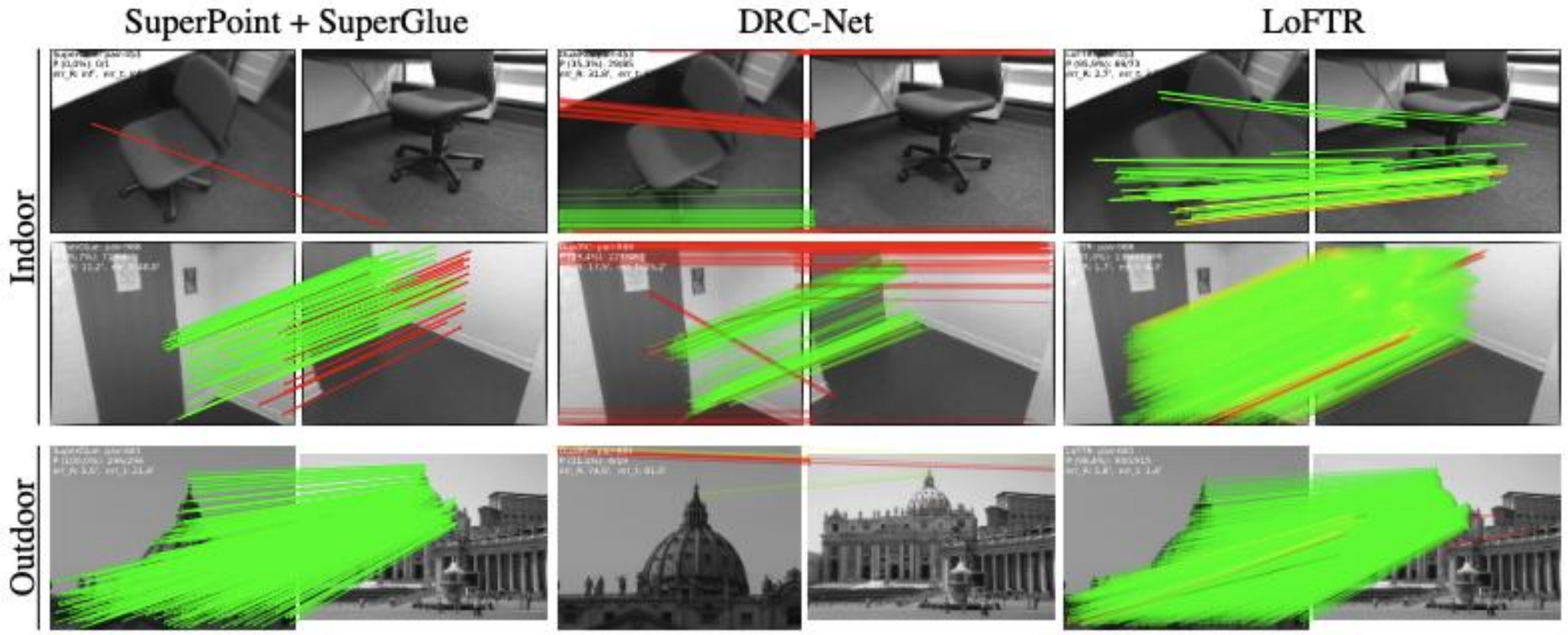
2) Feature Transformation: The coarse features are flattened, positionally encoded, and enriched with global context via a transformer using self- and cross-attention.

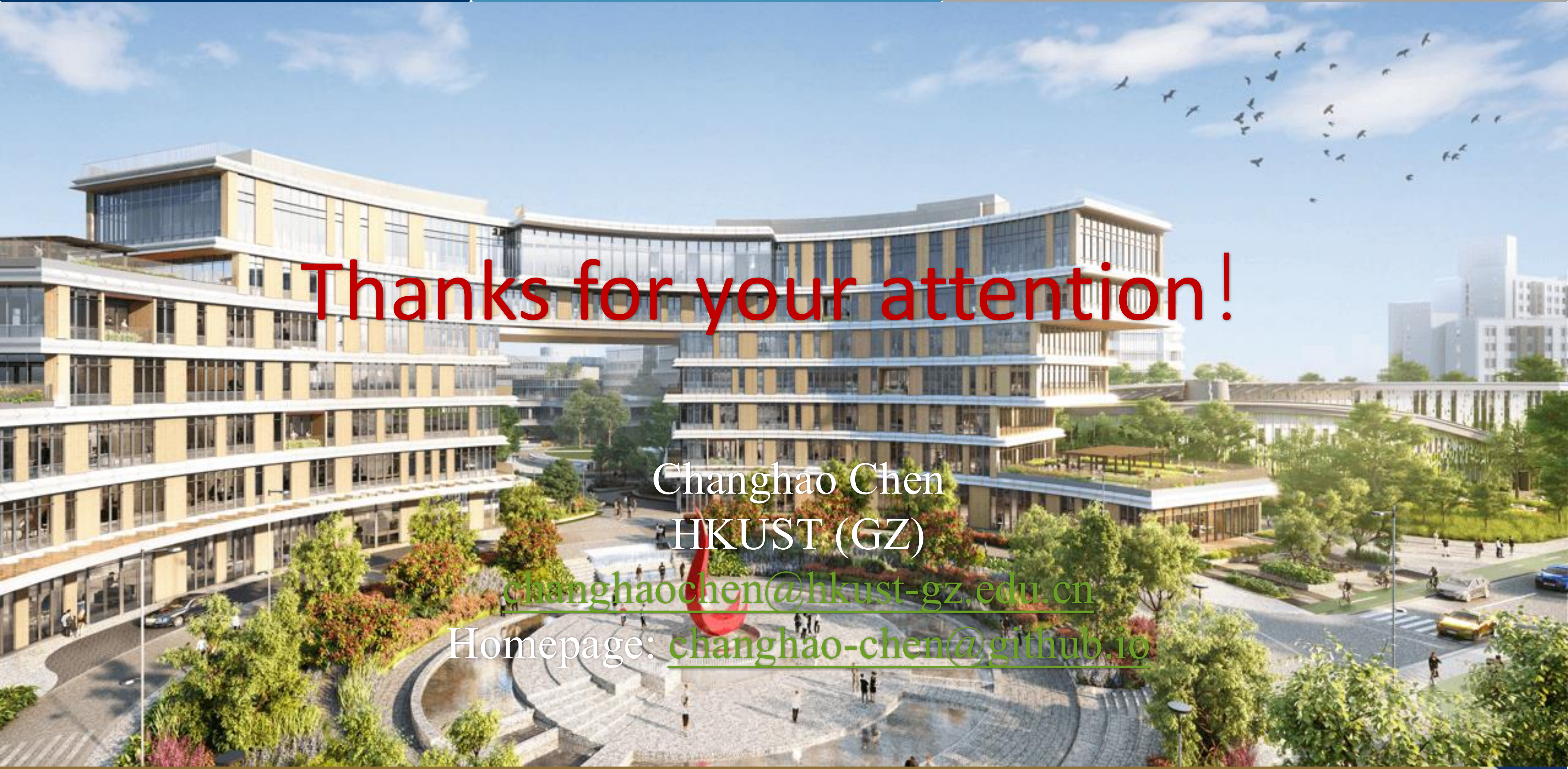
3) Coarse Matching: The transformed features are matched differentiably to produce a confidence matrix. High-confidence, mutual matches are selected as coarse predictions.

4) Fine Matching: Each coarse prediction defines a local window in the fine feature maps, where the match is refined to sub-pixel precision.

Deep Learning Enhanced Solutions

Feature Detection and Matching: LoFTR（2021）





Thanks for your attention！

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