L8 (Image Matching)

Homogeneity
$(\bar{x}, \bar{y}, w) = (\alpha\bar{x}, \alpha\bar{y}, \alpha w)$ for $\alpha \neq 0$
Converting from homogeneous to Cartesian coordinates is unique but **not** vice versa.
Transformations such as translation, rotation, scaling, shearing and perspective projection are commonly applied. Using homogeneous coordinates, all these transformations can be represented as **matrix multiplications**, making them more efficient.

In Cartesian, translation is an addition operation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

In homogeneous coordinates, translation is incorporated into a single matrix **multiplication**:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Conversion

2D point -> homogeneous point: append 1 as 3rd coordinate

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous point -> 2D point: divide by 3rd coordinate

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

Points at infinity occur when $w = 0$ (vanishing points, epipolar geometry in stereo)
Euclidean transform (rotation and/or translation):

$$x' = \bar{x} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

If the rotation is scaled: multiple sinusoids with a factor of s
Affine transform

$$x' = \bar{x} \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}$$

The upper-left 2x2 submatrix represents linear transformations such as rotation, scaling and shear. (a_{31}, a_{32})represents translation.

- Affine transforms do **not** preserve distances or angles but do **preserve parallel lines**
- **Origin does not necessarily map to origin**
- Multiple affine transformations can be combined through **matrix multiplication**
- Any affine transformations can be **decomposed** into simpler transformations such as translation, rotation, scaling, shear
  Example of decomposing affine transform into rotation and translation matrix multiplication:

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

  Translation can be further decomposed into rotation by 0 or πn angle and translation
  Vertical shear

$$x' = \bar{x} \begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  Horizontal shear

$$x' = \bar{x} \begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homography

Projective (perspective) transform
Unlike affine transforms, homography does **not** preserve the homogeneous coordinate $w \neq 1$. Normalization is necessary to get Cartesian coordinates.
Only straight lines are preserved.

$$x' = \bar{x} \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Preservation of Geometry

- Translation: orientation, length, angle, parallelism, straight lines
- Euclidean: length, angle, parallelism, straight lines
- Similarity(scaling): angle, parallelism, straight lines
- Affine: parallelism

- Projective: straight lines

Correspondence Map

There are two ways to find correspondence map: direct methods and feature-based

Direct methods

- Based on pixel values
- Search(direct matching) or gradient-based methods
- Block matching, optical flow

Block Matching

The goal is to find the best match for a **block of pixels** in one frame by searching for its **most similar block** in a reference frame. It is only suitable for **translational motion**. Block matching works by minimizing a similarity metric, such as:

- Sum of Absolute Differences (SAD)
- MSE
- Sum of Squared Differences (SSD):

$$E_{SSD}[k,l] = \sum_{(m,n)\in B} (s[m,n] - g[m+k, n+l])^2$$

- sensible to changes in brightness and contrast

Search Strategies

Full Search (Brute Force):

- examines all possible displacements within the search area
- guarantees finding the global minimum

Conjugate direction search:

- alternates search in x and y directions, breaking 2D search into two 1D searches
- faster than full search but can get trapped in **local minima**

Coarse-to-Fine (Hierarchical search):

- performs a full search at a coarse resolution(low-detail image), then refines motion estimation at higher resolutions

- efficient for handling **large motion** by first capturing rough movement before refining details.

Disadvantages of Block Matching

- assumes uniform translational motion for entire blocks, meaning it does not handle rotation, occlusion well

Optical Flow

While block matching assumes **uniform translational motion** for fixed-size blocks, optical flow estimates the **motion of each pixel** independently. It estimates the **apparent motion** of pixels between two consecutive frames based on brightness constancy and smoothness constraints. It assumes that:

1. Brightness Constancy Assumption: the intensity of a pixel remains **constant** over time: $I(x, y, t) = I(x + u, y + v, t + 1)$ where (u, v) is the motion vector.
2. Spatial and temporal smoothness assumption: neighboring pixels have similar motion (except at object boundaries)

By applying Taylor expansion, we get optical flow equation:

$$I_x u + I_y v + I_t = 0$$

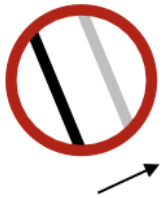
$$\frac{\delta s}{\delta x} u + \frac{\delta s}{\delta y} v = -\frac{\delta s}{\delta t}$$

Since this equation has two unknowns (**u,v**) but only one equation per pixel, additional constraints are required to solve for motion.
Optical flow is **not** a good approximation for too **large motions** because such motions tend to be locally non-linear.

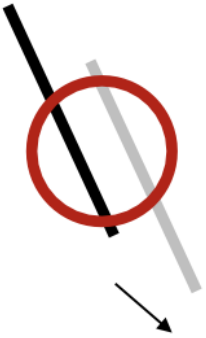
Aperture problem

When estimating motion through a **small window(aperture)**, only the motion perpendicular to the edge can be reliably detected, while motion along the edge direction remains

ambiguous.



Perceived motion
through local aperture



True motion

Barber pole illusion

Lucas-Kanade

It assumes that motion is **locally constant** within a small window of neighboring N pixels. It minimizes MSE error:

$$(S[u,v]^T - t)^2$$

The least squares solution can be obtained analytically:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (S^T S)^{-1} S^T t$$

Structure matrix $S^T S$ is given by:

$$S^T S = \begin{bmatrix} \sum_N s_x^2 & \sum_N s_x s_y \\ \sum_N s_x s_y & \sum_N s_y^2 \end{bmatrix}$$

which is similar to Harris detector structure matrix without additional weighting. This is not invertible in regions with no structure(flat areas) where both eigenvalues are zero.
Even if it is invertible, it can be ill-conditioned: ratio of eigenvalues are too large-> aperture problem
Not singular(not invertible) for corners and textured areas.
Similar to Harris detector, we add distance-based weighting factor (Gaussian):

$$\begin{bmatrix} u \\ v \end{bmatrix} = (S^T W S)^{-1} S^T W t$$

Problem with Optical Flow

- **Small displacements:** most optical flow algorithms assume that motion between consecutive frames is small. This assumption is critical because they rely on Taylor approximation of the intensity, assuming motion causes small intensity changes. Large displacements violate this assumption, leading to incorrect or unstable motion estimates.

- **Incorrect matching**: if an object moves too far between frames, the correct match may not be found within the local search window.
  The common solution is to use an image pyramid and down-sampling:
- Motion is estimated at low-resolution scale (after the downsampling), where large displacements become smaller relative to the image size.
- The estimated motion is propagated to higher-resolution levels, refining the details.
  Suppose an object moves 50 pixels between frames. At half the resolution, this motion appears as 25 pixels. At quarter resolution, it appears as 12.5 pixels.

Feature-based

Steps:

1. Detect robust keypoints (Harris, SIFT, SURF)
2. Establish correspondences based on feature descriptors(based on finding nearest neighbor in descriptor space)
3. Obtain model parameters from correspondences (homography estimation)

Homography Estimation

(Homogeneous Linear Least Squares Problem)
Given feature correspondences, estimate 8 parameters of the homography matrix $h_1, .. h_8$
One correspondence gives yields two equations. We need 4 correspondences to solve the problem. In practice, we can use many more points to improve upon noise, outliers, etc.
In matrix notation:

$$\begin{bmatrix} A_1 \\ A_2 \\ \dots \\ A_n \end{bmatrix} h = 0 \rightarrow \quad Ah = 0$$

where

$$A_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i'x_i & x_i'y_i & x_i' \\ 0 & 0 & 0 & -x_i & -y_i & -1 & y_i'x_i & y_i'y_i & y_i' \end{bmatrix}$$

$$h = \begin{bmatrix} h_1 & \dots & h_9 \end{bmatrix}^T$$

In practice, this is formulated as a cost minimization problem based on different distance metrics:

- Algebraic distance : $\epsilon = ||Ah||$
- Geometric distance: $\epsilon = \sum_i d(x_i', Hx_i)^2$
- Symmetric transfer error
- Reprojection, Sampson error

Outliers are bad matches that have huge effect on cost function minimization. So, how to reject outliers?

RANSAC

Here comes RANSAC to help.

1. Sample (randomly) the number of points required to fit the model
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence.

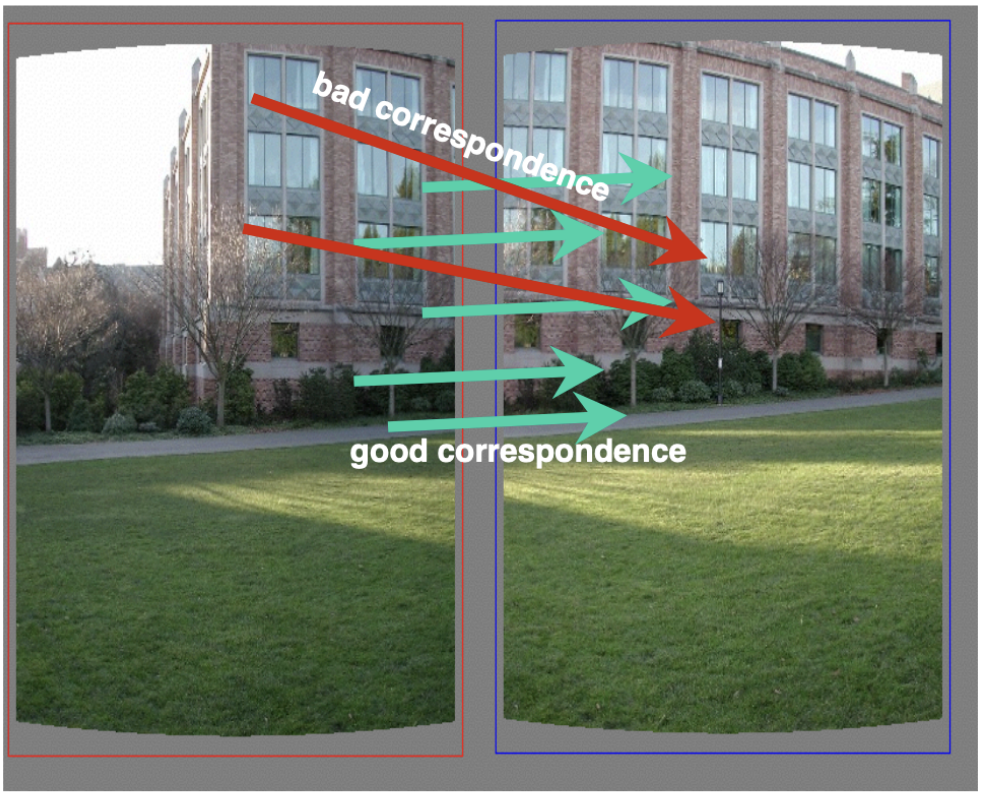
Estimating homography using RANSAC:

1. Get four point correspondences(randomly)
2. Compute H using DLT
3. Count inliers
4. Keep H if largest number of inliers
   Recompute H using all inliers
   Example: estimating translational motion

Given two images…

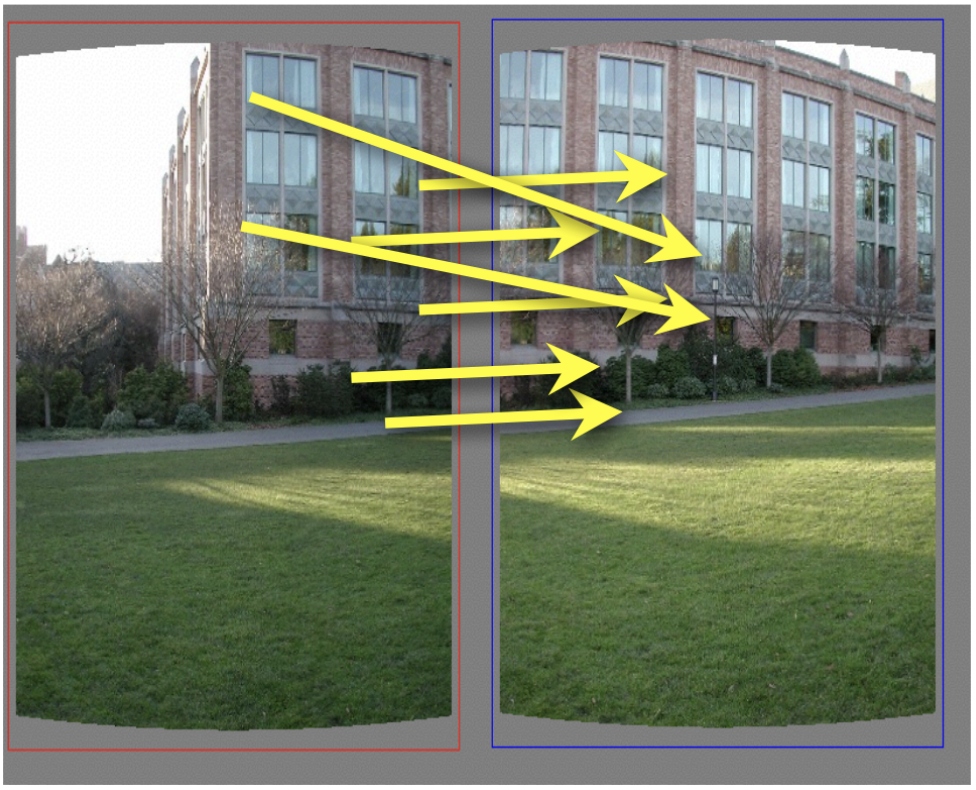
find matching features (e.g., SIFT)

Matched points will usually contain bad correspondences



how should we estimate the transform?

LLS or DLT will find the ‘average’ transform



Need only **one correspondence**, to find translation model

Pick one correspondence, count inliers

RANSAC Robustness

Having more iterations is more important than higher p_{in}ratio. p_{in} is the probability of inliers within a feature set.

- Probability of picking 4 inliers: $p = p_{in}^4$
- Probability that after N iterations we have **not** picked a set of 4 inliers:

$$p_N = (1 - p_{in}^4)^N$$

Example: $p_{\text{in}} = 0.5$

- Probability of picking 4 inliers? $p = p_{\text{in}}^4 = 0.5^4 \cong 6\%$
- Probability of not picking 4 inliers after N iterations?

$$p_N = \left(1 - p_{\text{in}}^4\right)^N \longrightarrow p_{100} \cong 0.2\%$$

Example: $p_{\text{in}} = 0.1$

- Probability of picking 4 inliers? $p = p_{\text{in}}^4 = 0.1^4 = 0.01\%$
- Probability of not picking 4 inliers after N iterations?

$$p_{100} \cong 99\% \quad p_{1000} \cong 90\% \quad p_{10000} \cong 37\% \quad p_{50000} \cong 0.7\%$$

L8 (Exercise)

- Write down the formulas for all transformations!
- Be careful about clockwise/anti-clockwise angle sign
- Be careful about y-axis direction
- Be careful about matrix multiplication order
- Remember the trick for inverse matrix calculation for a square matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \left(\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \right.$$