KENTECH
Korea Institute of Energy Technology

Advanced Computer Vision
Week 08

Oct. 25, 2022

Seokju Lee

SIFT Algorithm

Scale Invariant Feature Transform (SIFT)

and its applications for image alignment and 2D object recognition.

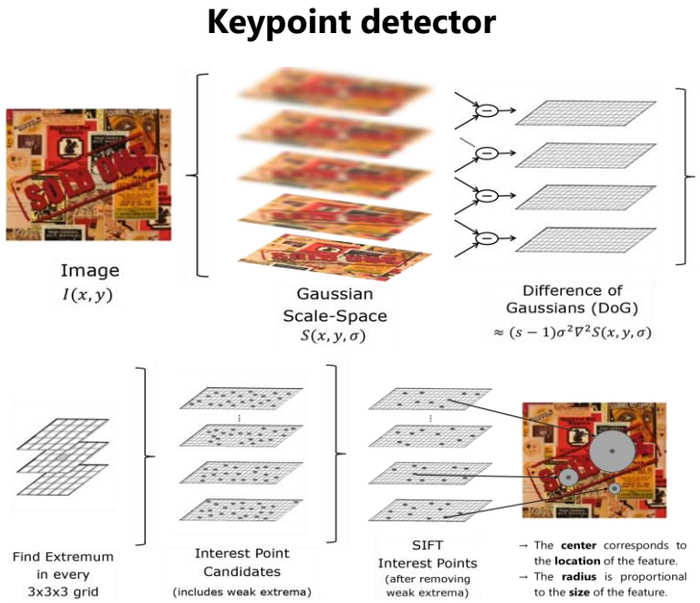
Topics:

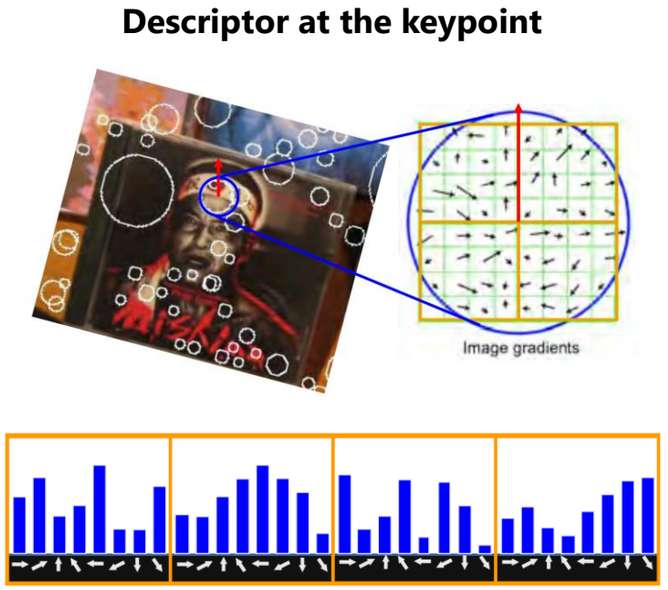
(1) What is an Interest Point?

(2) Detecting Blobs

(3) SIFT Detector

(4) SIFT Descriptor



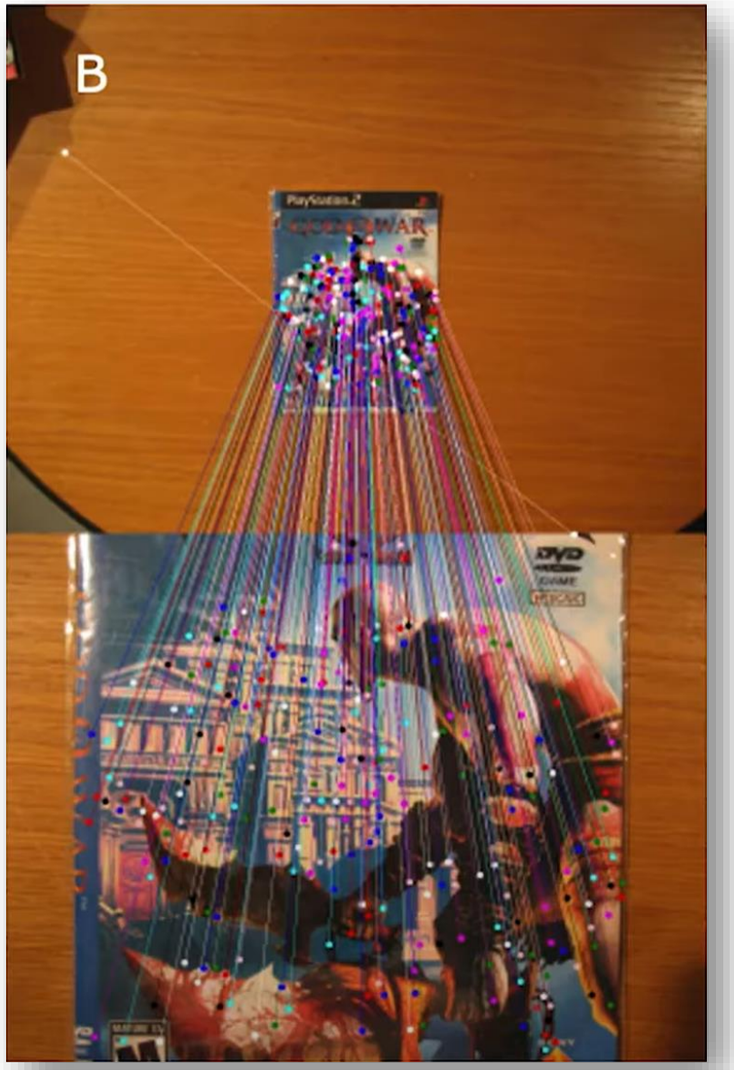


Parts of slides are by Prof. In So Kweon and Prof. Shree Nayar

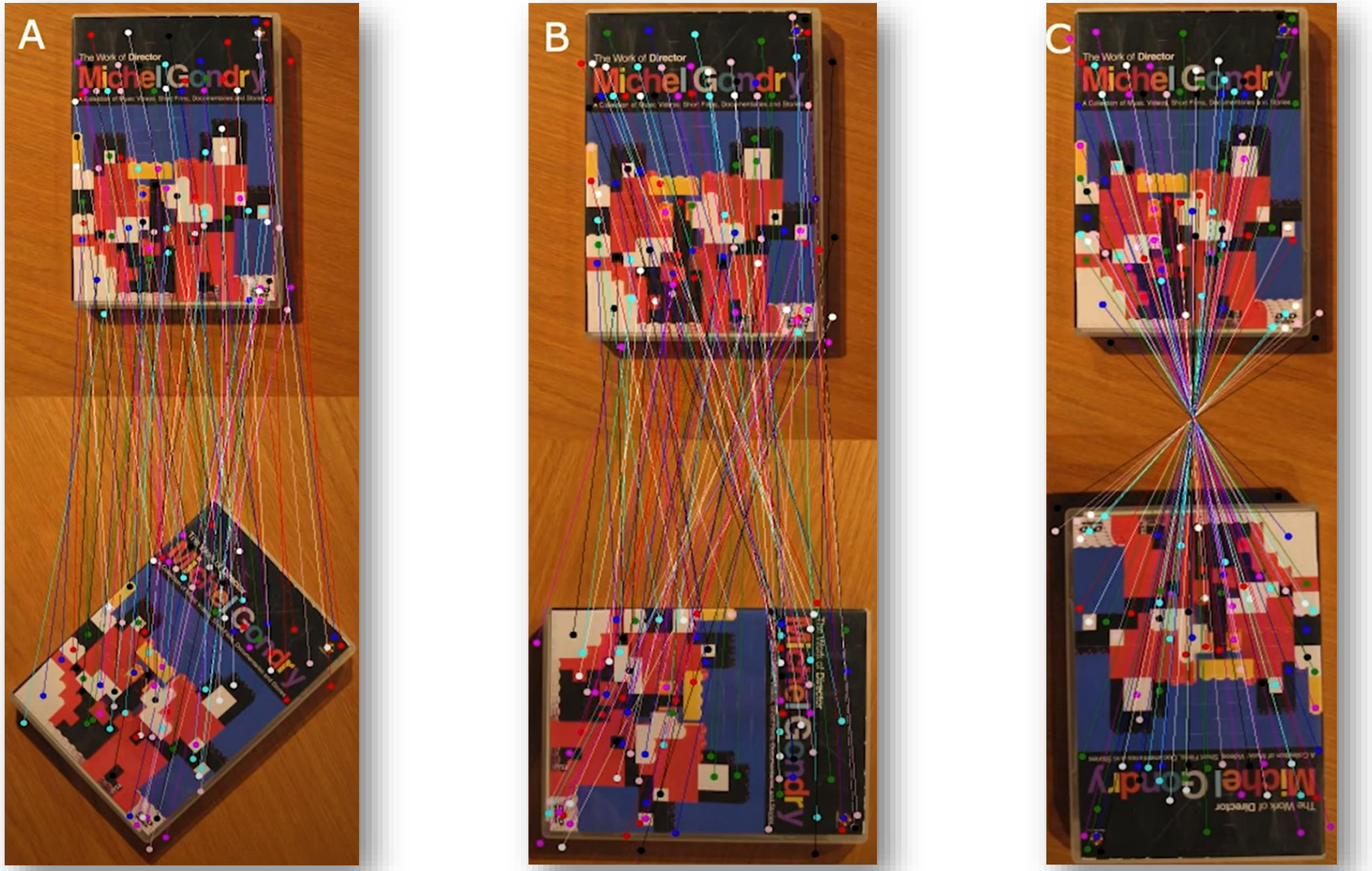
KENTECH
Korea Institute of Energy Technology

Experiment 1: SIFT Matching

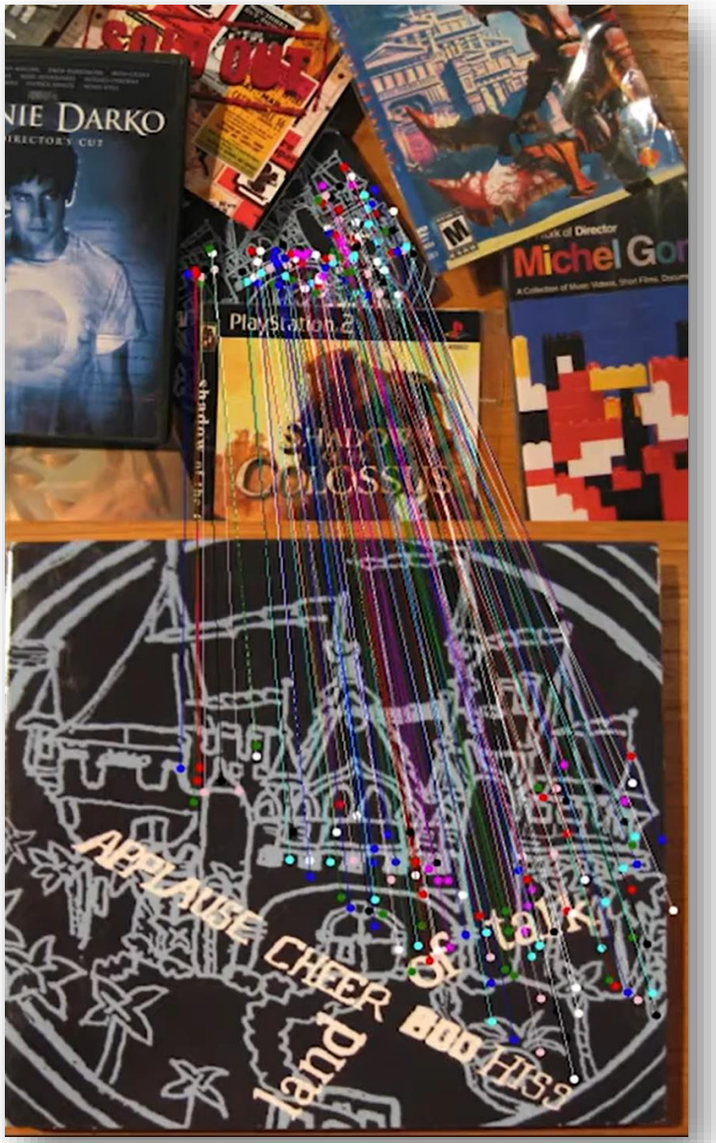
SIFT Results: Scale Invariance



SIFT Results: Rotation Invariance

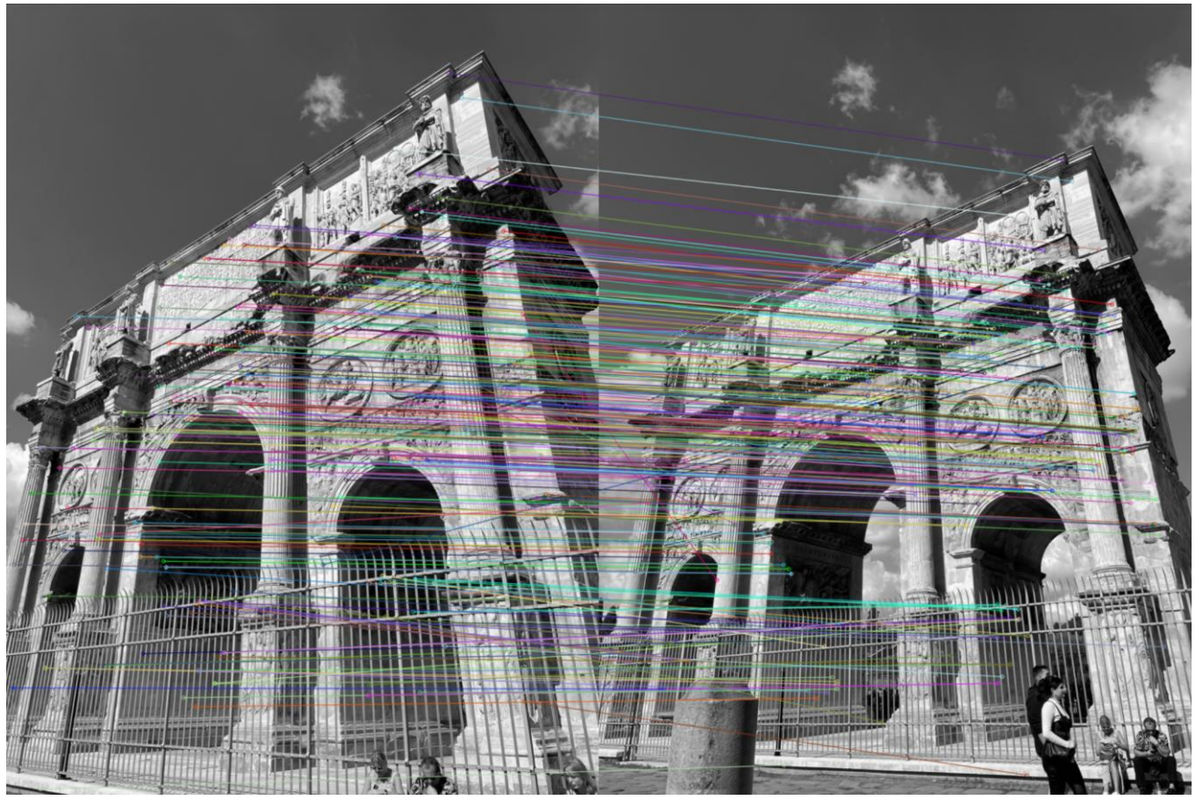


SIFT Results: Robust to Clutter

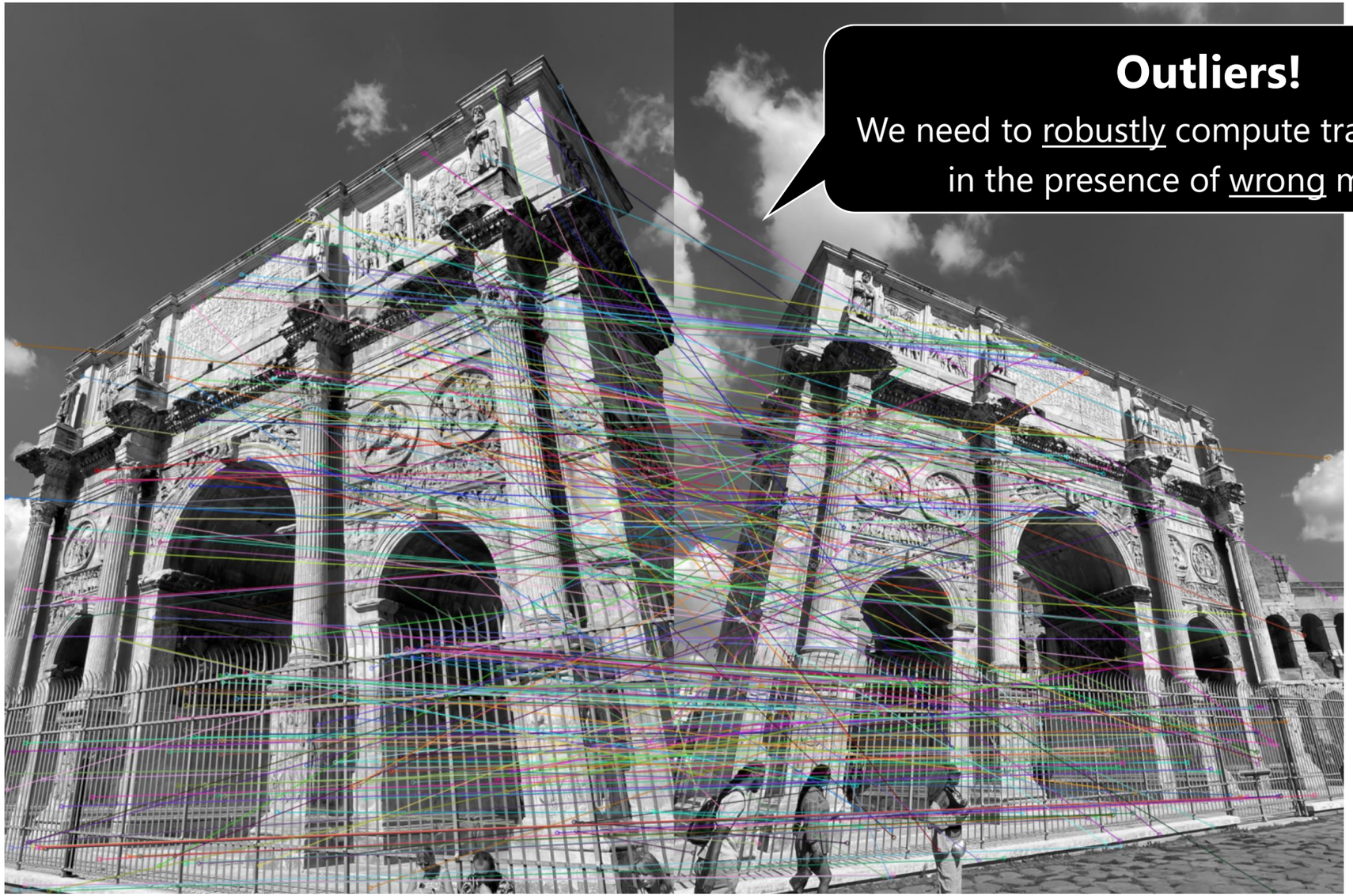


Experiment 1: SIFT Matching with Different Motions and Image Transformations

https://view.kentech.ac.kr/f088fa7f-874e-44bc-bd6d-6084b42dfdf7



How to Handle Outliers?



https://view.kentech.ac.kr

Parts of slides are by Prof. In So Kweon and Prof. Shree Nayar

KENTECH
Korea Institute of Energy Technology

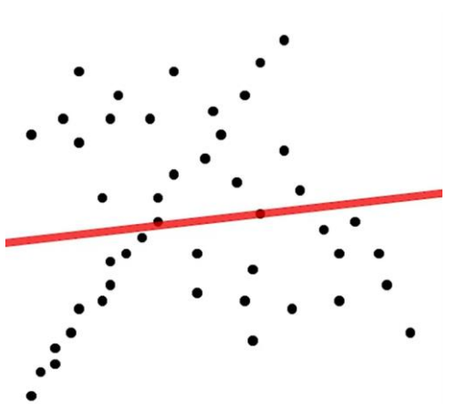
RANSAC

RANdom SAmple Consensus

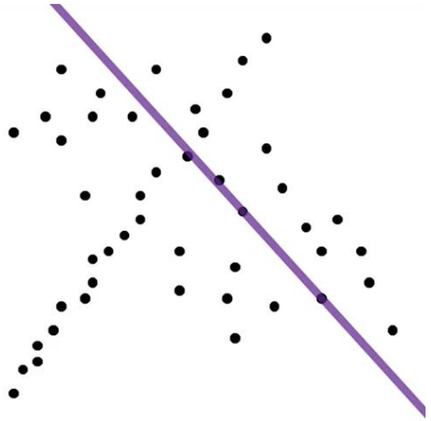
General RANSAC Algorithm :

1. Randomly choose s samples. Typically s is the minimum samples to fit a model.
2. Fit the model to the randomly chosen samples.
3. Count the number M of data points (inliers) that fit the model within a measure of error ε.
4. Repeat Steps 1-3 N times.
5. Choose the model that has the largest number M of inliers.

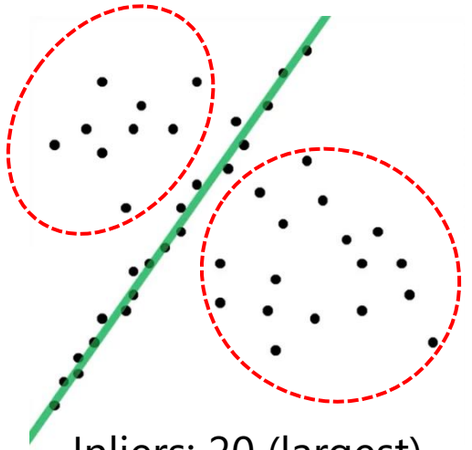
ex) Line fitting :



Inliers: 2



Inliers: 4



→ Outliers

Inliers: 20 (largest)

Parts of slides are by Prof. In So Kweon and Prof. Shree Nayar

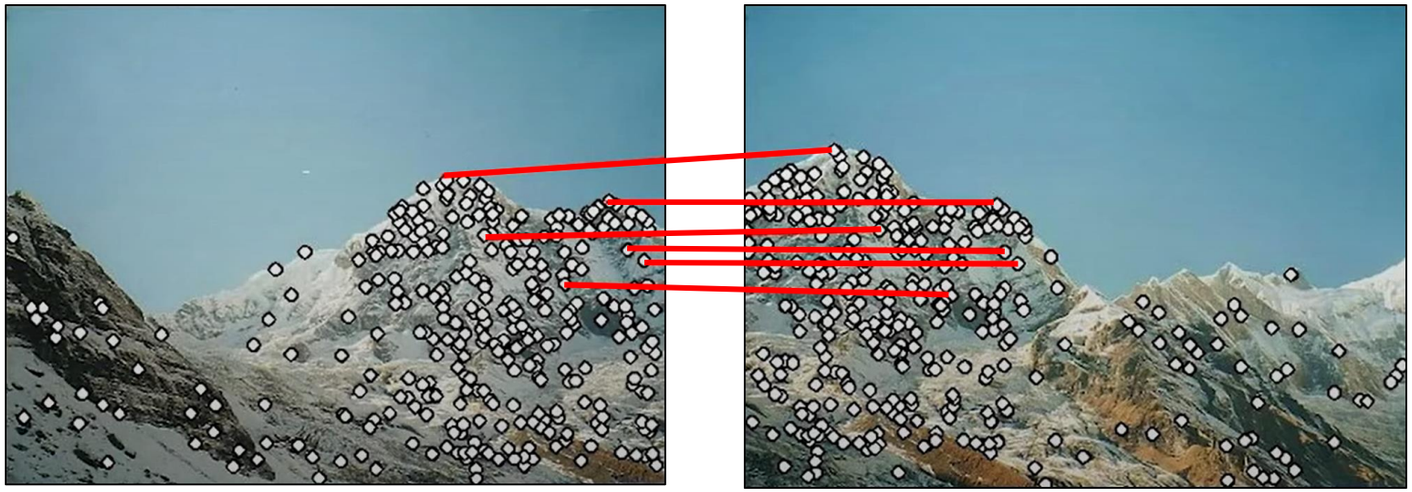
KENTECH
Korea Institute of Energy Technology

Experiment 2: Image Stitching

Experiment 2: Panorama Stitching Using SIFT + RANSAC

Image 1

Image 2



Match SIFT interest points.

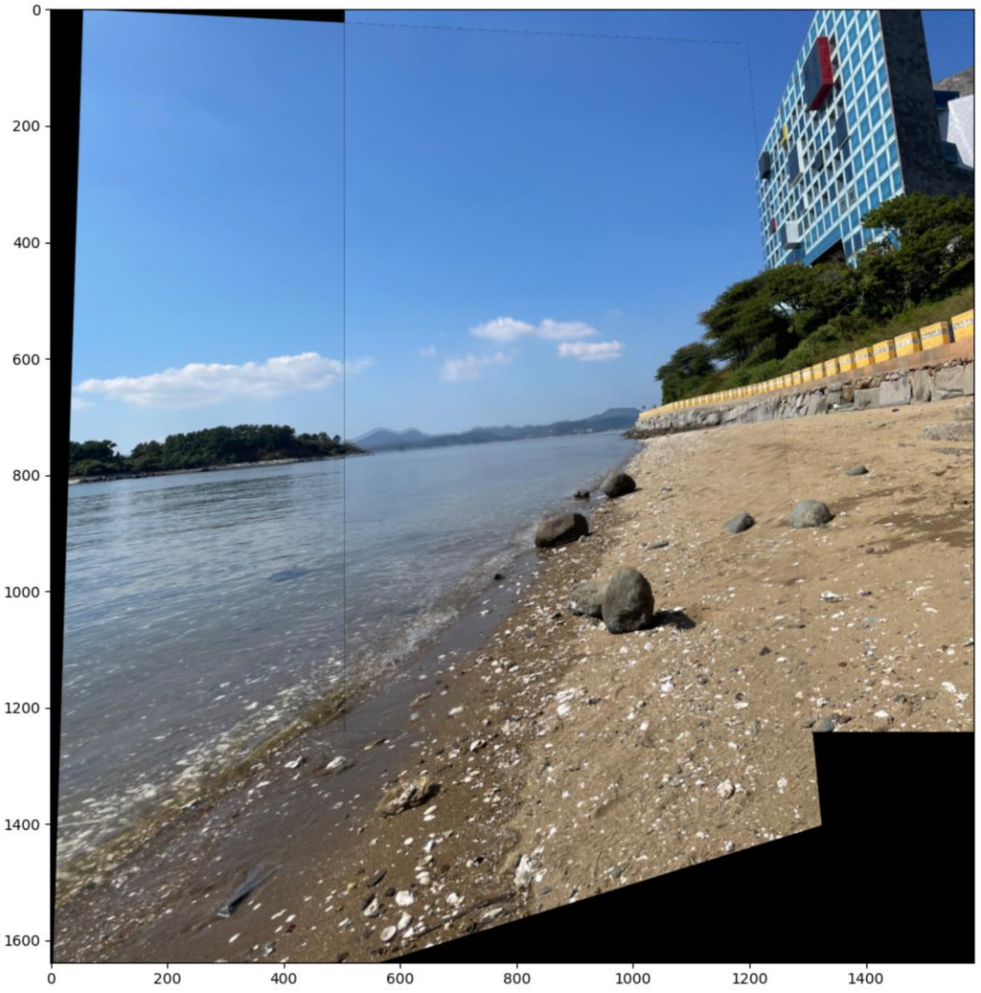
Experiment 2: Panorama Stitching Using SIFT + RANSAC

Warp and combine images to create a larger image.

Experiment 2: Panorama Stitching Using SIFT + RANSAC

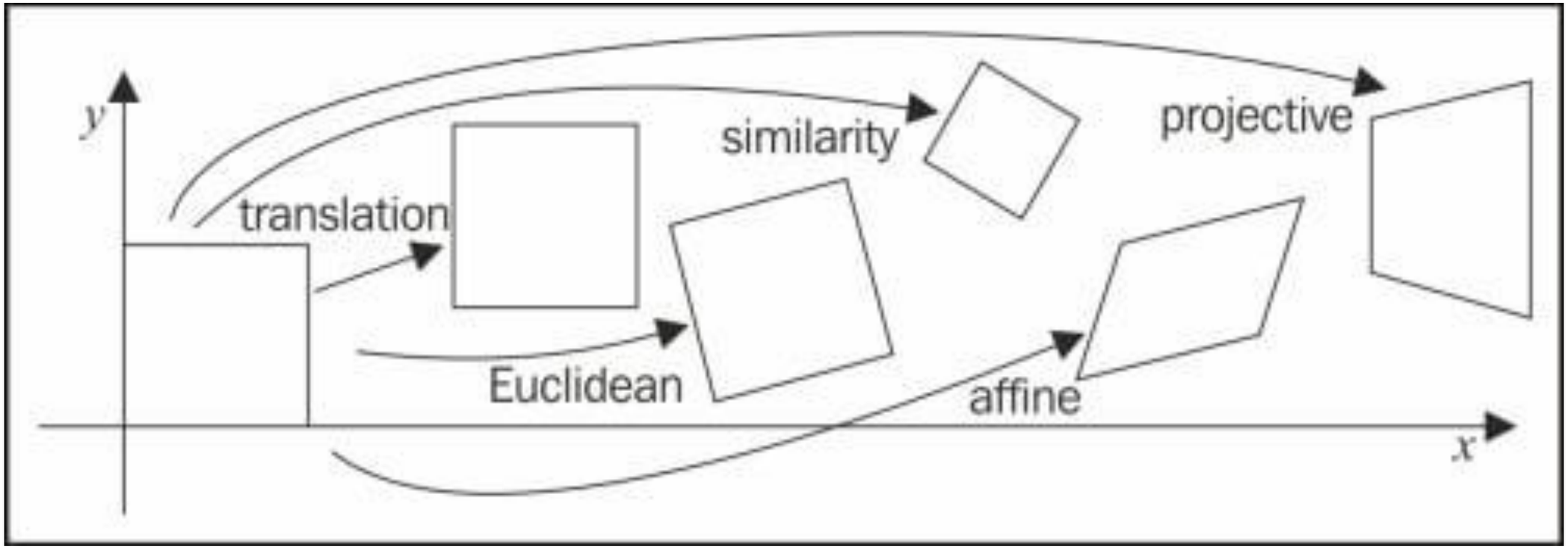
→ **Output results**

Matching result after SIFT matching + RANSAC



Warping result

Review: Homography



[1] Figure by Stanford Vision Lab

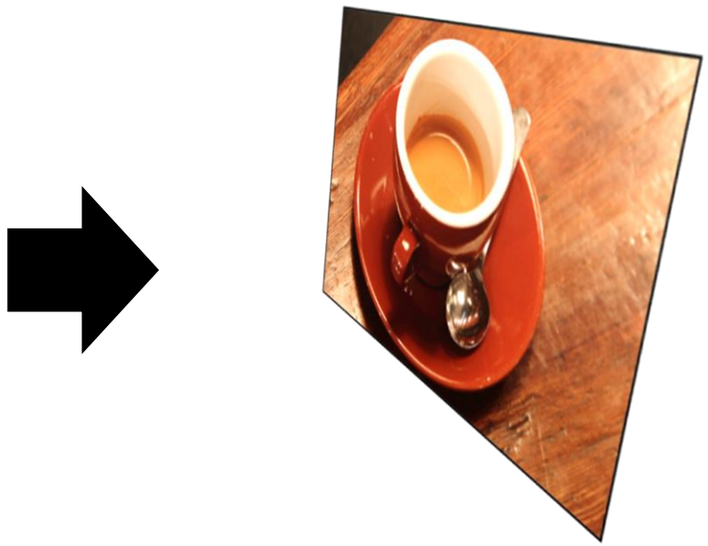
Review: Homography

- Any projective transformation of the form:

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

Also called **Homography**

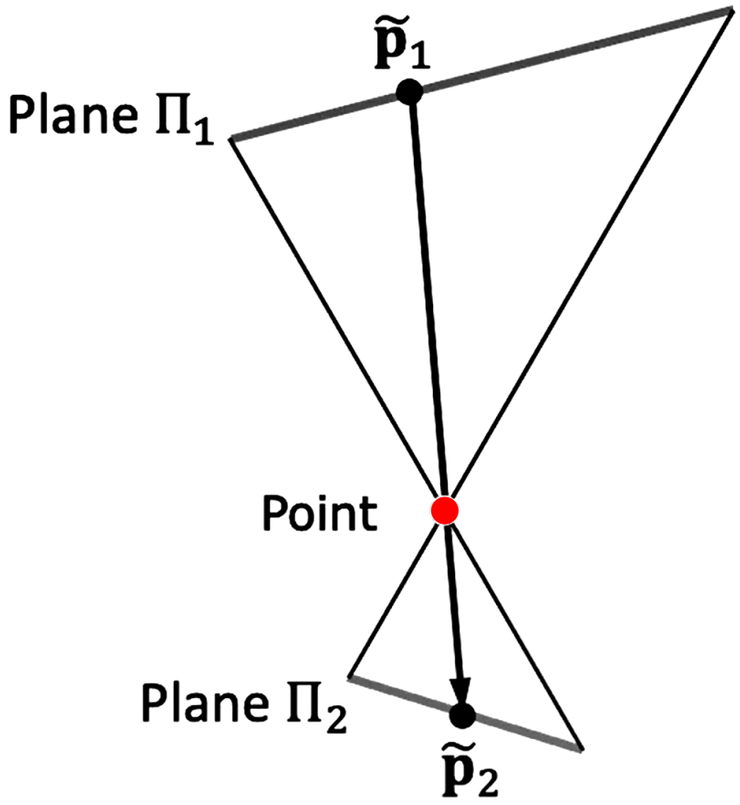
$$\tilde{\mathbf{p}}_2 = H\tilde{\mathbf{p}}_1$$



[1] Original slide by Shree Nayar

Review: Homography

- Mapping of one plane to another through a point.



$$\tilde{\mathbf{p}}_2 = H\tilde{\mathbf{p}}_1$$

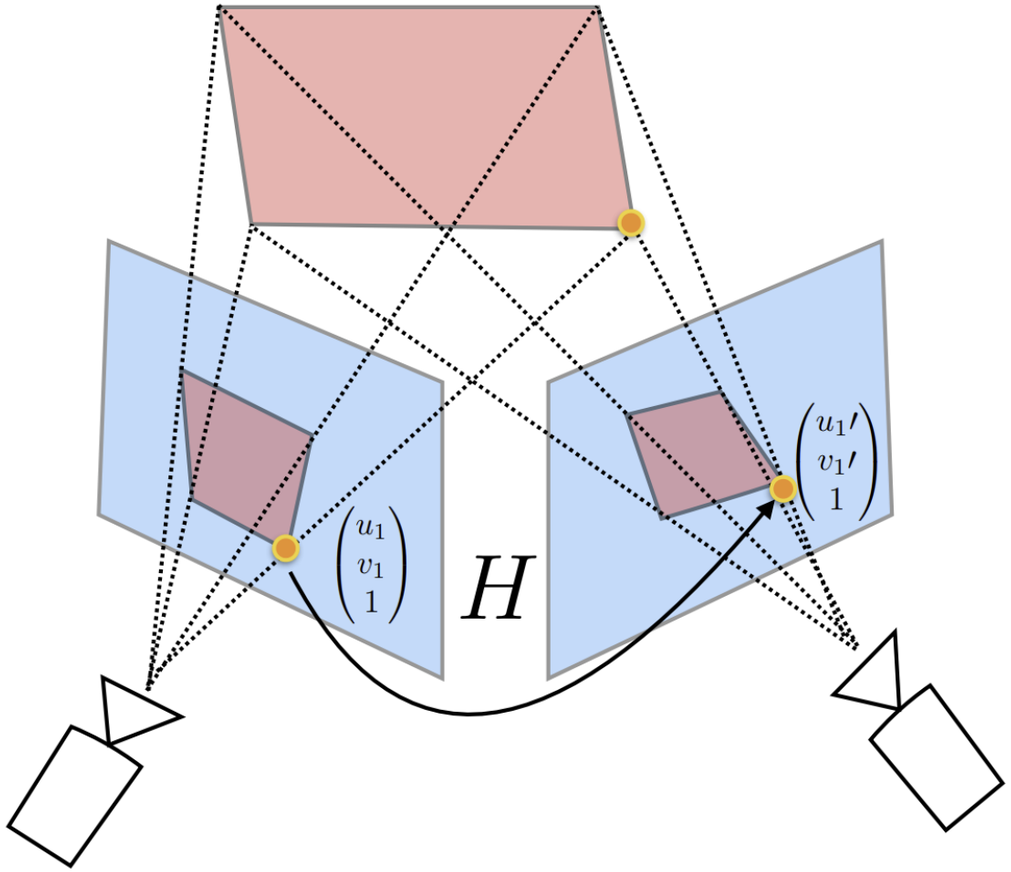
$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

Same as **imaging** a **plane** through a **pinhole**

- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Closed under composition

[1] Original slide by Shree Nayar

Finding Homography with Given 4 points Correspondence



$$H_{4point} = \begin{pmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \\ \Delta u_3 & \Delta v_3 \\ \Delta u_4 & \Delta v_4 \end{pmatrix}$$

1-to-1 mapping

$$H_{matrix} = \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{pmatrix}$$

[1] D. DeTone, et al., "Deep Image Homography Estimation", 2016.

Finding Homography with Given 4 points Correspondence

$$H_{projective} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \Rightarrow \begin{cases} x_i' = \dfrac{ax_i + by_i + c}{gx_i + hy_i + 1} \\ y_i' = \dfrac{dx_i + ey_i + f}{gx_i + hy_i + 1} \end{cases}$$

$$\Rightarrow \begin{cases} x_i'(gx_i + hy_i + 1) = ax_i + by_i + c \\ y_i'(gx_i + hy_i + 1) = dx_i + ey_i + f \end{cases} \Rightarrow \begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

→ Solve for $\mathbf{p}$, if $A\mathbf{p} = 0$ → Least square method (**LSM**) or singular value decomposition (**SVD**)

→ In $H_{projective}$, the **last** entry is defined only up to scales (**8-DoF**) → A 2D point has **2-DoF** to (x, y) components and each point-to-point **correspondence** accounts for **two constraints**. This is the reason why we need **4 points correspondences** for finding the homography.

[1] D. DeTone, et al., "Deep Image Homography Estimation", 2016.
[2] https://ai.stackexchange.com/questions/21042/how-do-you-find-the-homography-matrix-given-4-points-in-both-images

Python Codes

https://view.kentech.ac.kr/f088fa7f-874e-44bc-bd6d-6084b42dfdf7

- **Homography**

```
def homography(pairs):
    rows = []
    for i in range(pairs.shape[0]):
        p1 = np.append(pairs[i][0:2], 1)
        p2 = np.append(pairs[i][2:4], 1)
        row1 = [0, 0, 0, p1[0], p1[1], p1[2],
                -p2[1]*p1[0], -p2[1]*p1[1], -p2[1]*p1[2]]
        row2 = [p1[0], p1[1], p1[2], 0, 0, 0,
                -p2[0]*p1[0], -p2[0]*p1[1], -p2[0]*p1[2]]
        rows.append(row1)
        rows.append(row2)
    rows = np.array(rows)
    U, s, V = np.linalg.svd(rows)

    H = V[-1].reshape(3, 3)
    H = H/H[2, 2]
    return H
```

- **RANSAC**

```
def ransac(matches, threshold, iters):
    num_best_inliers = 0

    for i in range(iters):
        points = random_point(matches)
        H = homography(points)  # candidate

        if np.linalg.matrix_rank(H) < 3:
            continue  # Avoid dividing by zero.

        errors = get_error(matches, H)
        idx = np.where(errors < threshold)[0]
        inliers = matches[idx]

        num_inliers = len(inliers)
        if num_inliers > num_best_inliers:
            best_inliers = inliers.copy()
            num_best_inliers = num_inliers
            best_H = H.copy()

    return best_inliers, best_H
```