

Image Geometry



Sunglok Choi, Assistant Professor, Ph.D.
Computer Science and Engineering Department, SEOULTECH
sunglok@seoultech.ac.kr | <https://mint-lab.github.io/>

Review) Absolute Camera Pose Estimation

- Example) **Pose estimation (book) + camera calibration** – initially given K [pose_estimation_book3.py]

(due to wrong initial K)

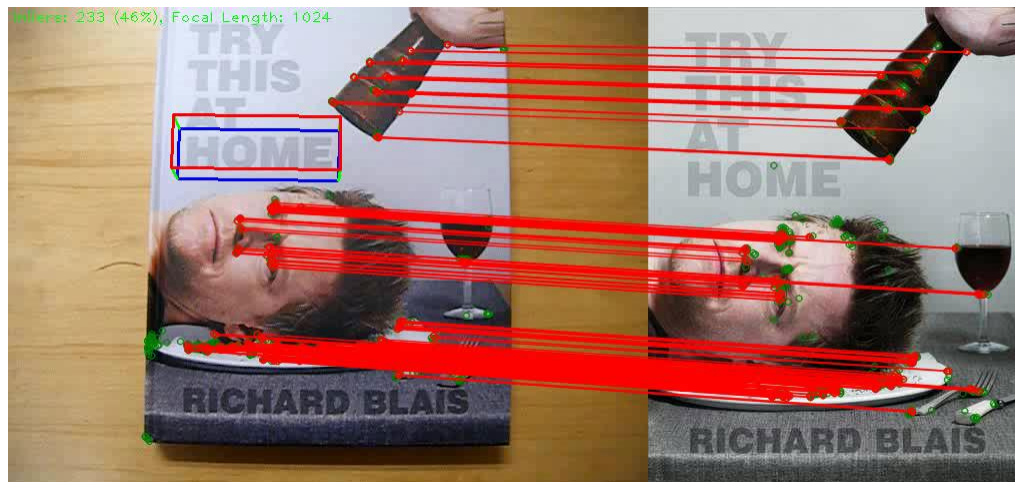
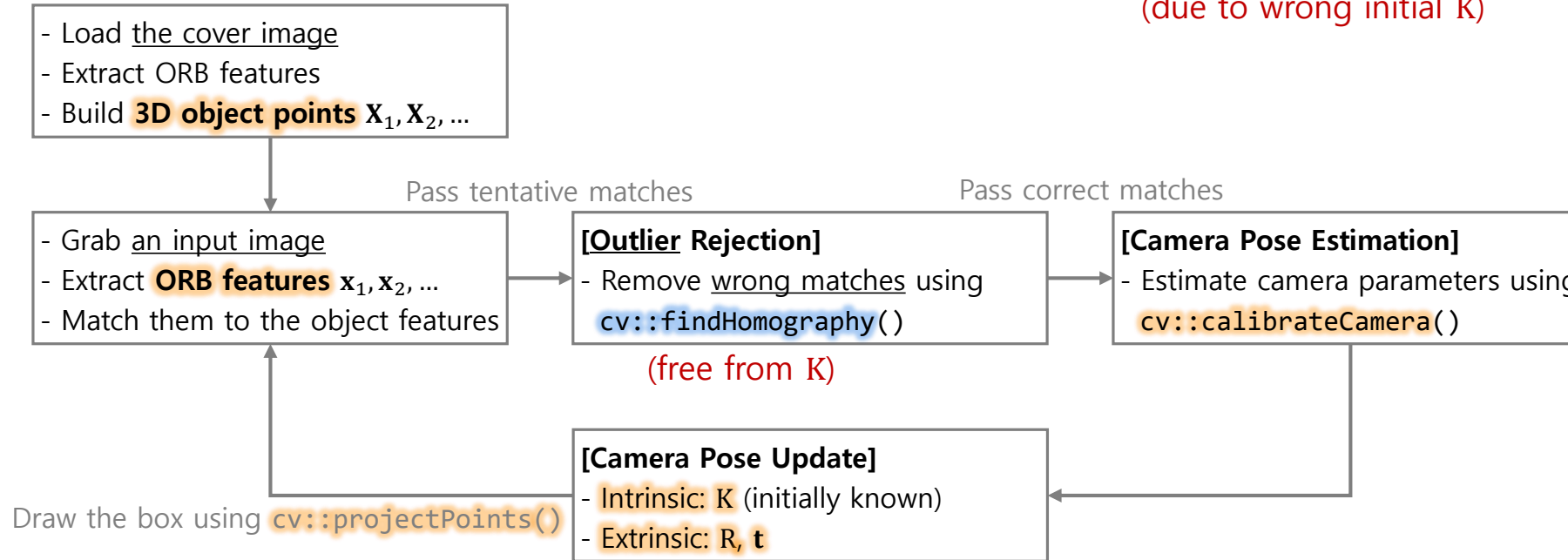
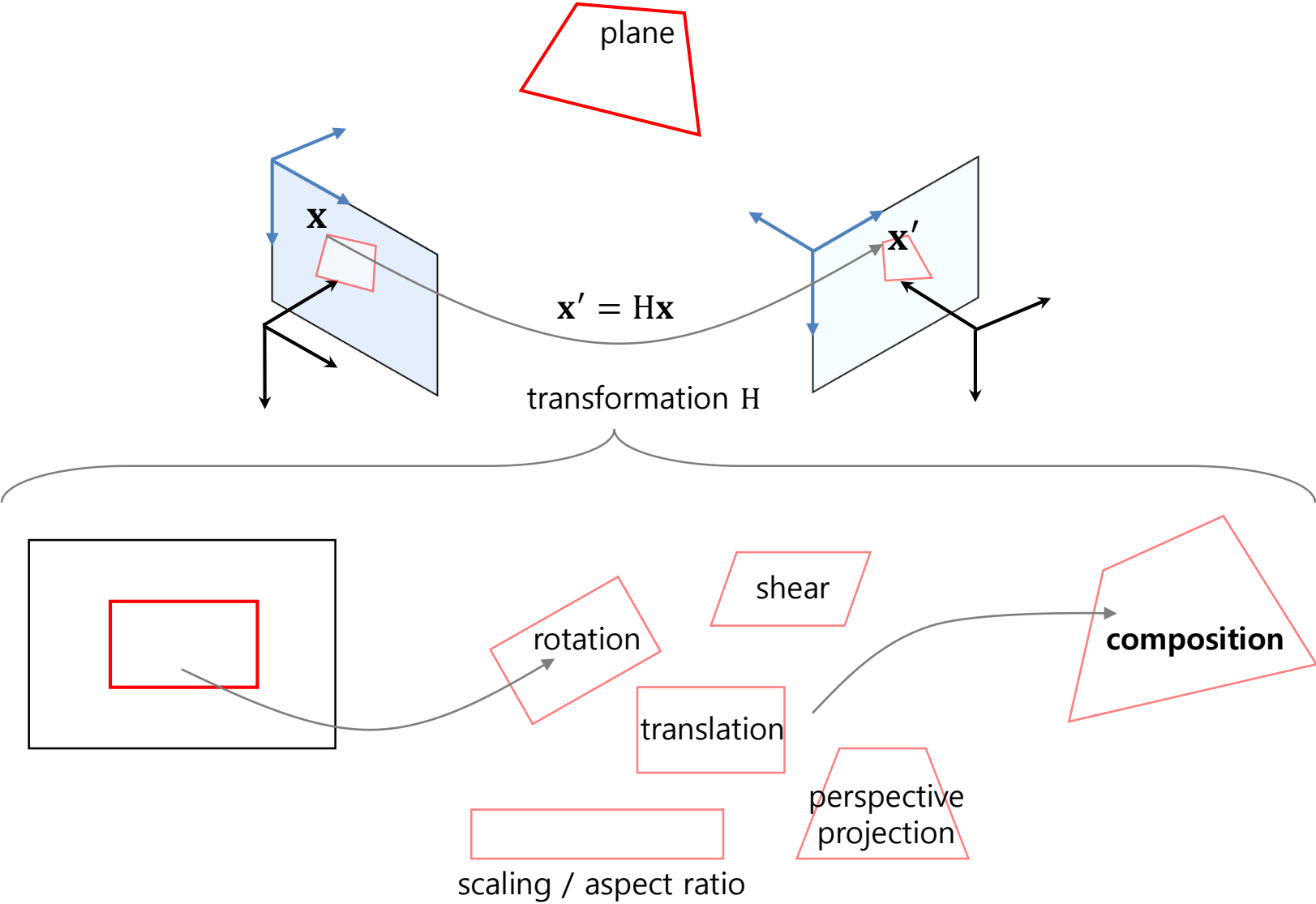



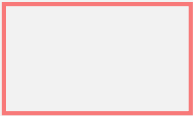

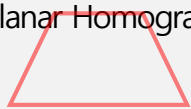
Table of Contents

- Planar Homography
- Triangulation

Planar Homography

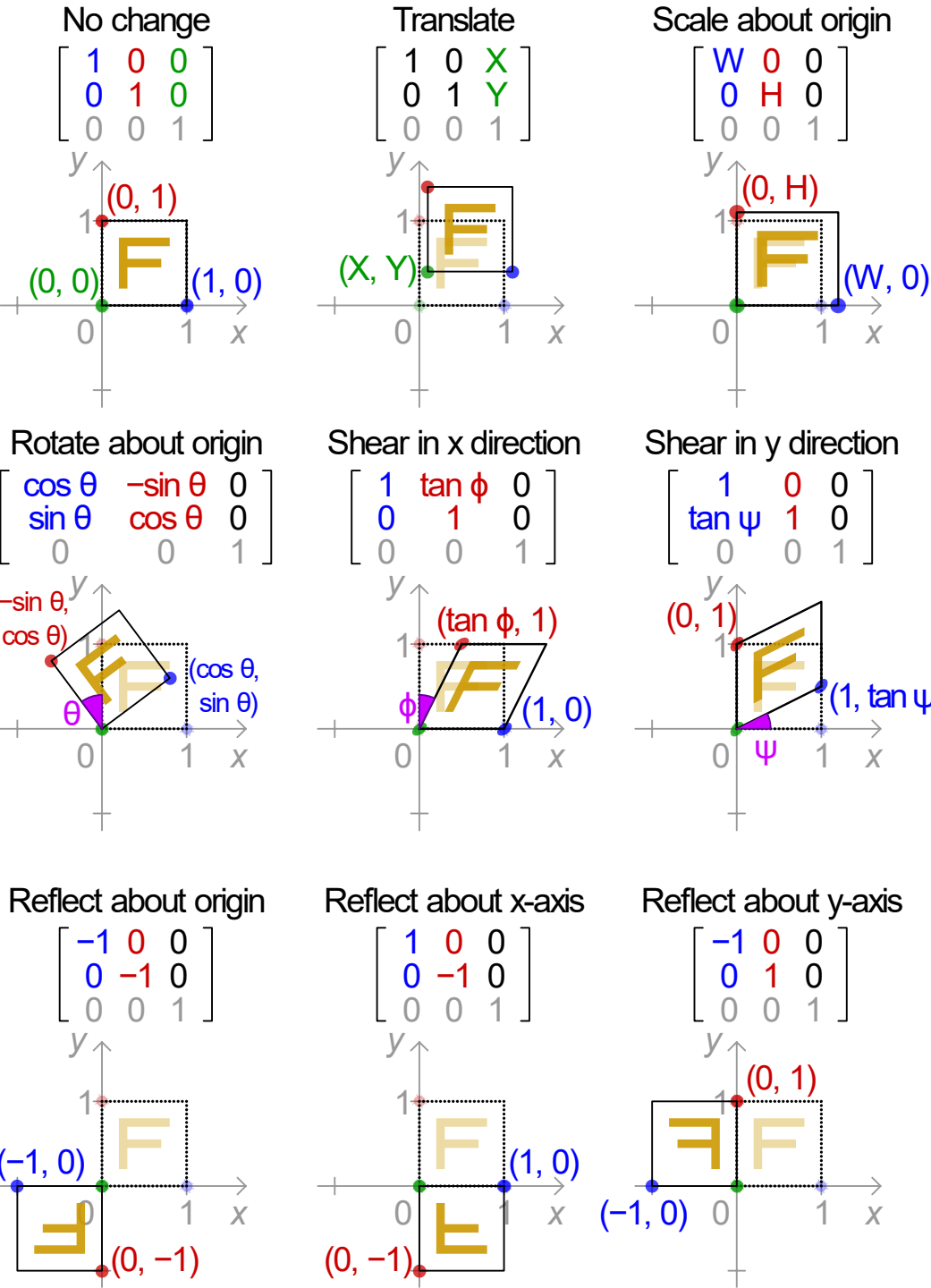


Planar Homography

	Euclidean Transformation (a.k.a. Rigid Transform)	Similarity Transformation	Affine Transformation	Projective Transformation (a.k.a. Planar Homography)
				
Matrix Forms H	$\begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s \cos \theta & -s \sin \theta & t_x \\ s \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & 1 \end{bmatrix}$
DOF	3	4	6	8
Transformations - rotation - translation - scaling - aspect ratio - shear - perspective projection	 ○ ○ X X X X	 ○ ○ ○ X X X	 ○ ○ ○ ○ ○ X	 ○ ○ ○ ○ ○ ○
Invariants - length - angle - ratio of lengths - parallelism - incidence - cross ratio	 ○ ○ ○ ○ ○ ○	 X ○ ○ ○ ○ ○	 X X X ○ ○ ○	 X X X X ○ ○
OpenCV APIs			cv.getAffineTransform() cv.estimateRigidTransform() - cv.warpAffine()	cv.getPerspectiveTransform() - cv.findHomography() cv.warpPerspective()

Note) Similarly **3D transformations** (3D-3D geometry) are represented as **4x4 matrices**.

Note) Affine Transformation



Note) Affine Transformation

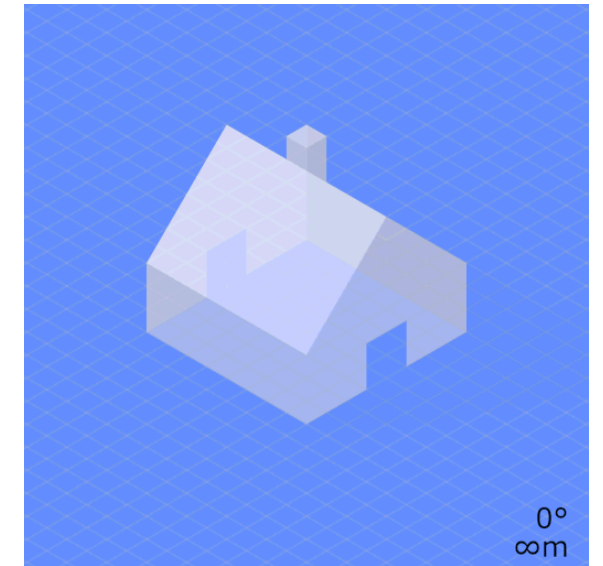


orthographic projection



perspective projection

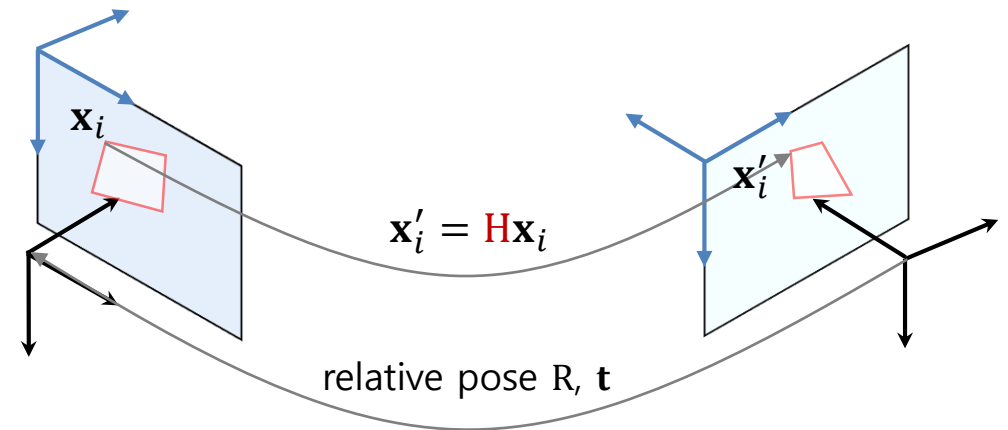
- Q) What is difference between two images?
 - [Parallel projection](#) vs. Perspective projection
 - Affine camera (a.k.a. weak perspective camera; $f = \infty$)
 - Less natural but (sometimes) useful in technical visualization



Planar Homography

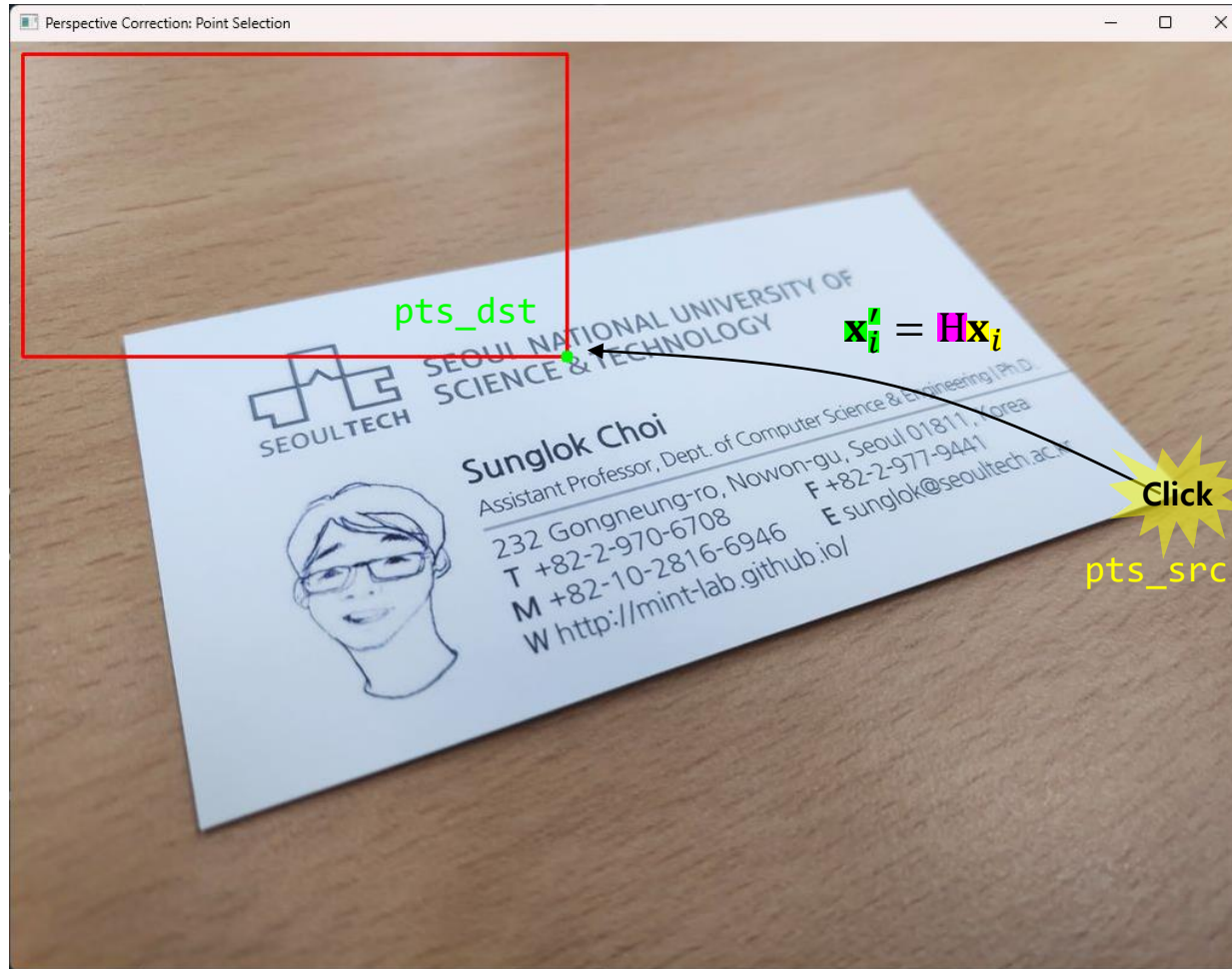
Planar homography estimation

- Unknown: Planar homography H (8 DOF)
- Given: Point correspondence $(\mathbf{x}_1, \mathbf{x}'_1), \dots, (\mathbf{x}_n, \mathbf{x}'_n)$
- Constraints: $n \times$ projective transformation $\mathbf{x}'_i = H\mathbf{x}_i$
- Solutions ($n \geq 4$) \rightarrow 4-point algorithm
 - OpenCV: `cv.getPerspectiveTransform()` and `cv.findHomography()`
 - Note) More simplified transformations need less number of minimal correspondence.
 - Affine ($n \geq 3$), similarity ($n \geq 2$), Euclidean ($n \geq 2$)
- Note) Planar homography can be decomposed as relative camera pose.
 - OpenCV: `cv.decomposeHomographyMat()`
 - The decomposition needs to know camera matrices.



Planar Homography

- Example) **Perspective distortion correction** [perspective_correction.py]



Planar Homography

- Example) **Perspective distortion correction** [perspective_correction.py]

```
def mouse_event_handler(event, x, y, flags, param):
    if event == cv.EVENT_LBUTTONDOWN:
        param.append((x, y))

if __name__ == '__main__':
    img_file = '../data/sunglok_card.jpg'
    card_size = (450, 250)
    offset = 10

    # Prepare the rectified points
    pts_dst = np.array([[0, 0], [card_size[0], 0], [0, card_size[1]], [card_size[0], card_size[1]]])

    # Load an image
    img = cv.imread(img_file)

    # Get the matched points from mouse clicks
    pts_src = []
    wnd_name = 'Perspective Correction: Point Selection'
    cv.namedWindow(wnd_name)
    cv.setMouseCallback(wnd_name, mouse_event_handler, pts_src)
    while len(pts_src) < 4:
        img_display = img.copy()
        cv.rectangle(img_display, (offset, offset), (offset + card_size[0], offset + card_size[1]), (0, 0, 255), 2)
        idx = min(len(pts_src), len(pts_dst))
        cv.circle(img_display, offset + pts_dst[idx], 5, (0, 255, 0), -1)
        cv.imshow(wnd_name, img_display)
```

Planar Homography

- Example) **Perspective distortion correction** [perspective_correction.py]

```
if __name__ == '__main__':
    img_file = '../data/sunglok_card.jpg'
    card_size = (450, 250)
    offset = 10

    # Prepare the rectified points
    pts_dst = np.array([[0, 0], [card_size[0], 0], [0, card_size[1]], [card_size[0], card_size[1]]])

    # Load an image
    img = cv.imread(img_file)

    # Get the matched points from mouse clicks
    pts_src = []
    ...

    if len(pts_src) == 4:
        # Calculate planar homography and rectify perspective distortion
        H, _ = cv.findHomography(np.array(pts_src), pts_dst)
        img_rectify = cv.warpPerspective(img, H, card_size)

        # Show the rectified image
        cv.imshow('Perspective Correction: Rectified Image', img_rectify)
        cv.waitKey(0)

    cv.destroyAllWindows()
```

Planar Homography

- Example) **Planar image stitching** [image_stitching.py]

```
# Load two images
```

```
img1 = cv.imread('../data/hill01.jpg')
```

```
img2 = cv.imread('../data/hill02.jpg')
```

```
# Retrieve matching points
```

```
brisk = cv.BRISK_create()
```

```
keypoints1, descriptors1 = brisk.detectAndCompute(img1, None)
```

```
keypoints2, descriptors2 = brisk.detectAndCompute(img2, None)
```

```
fmatcher = cv.DescriptorMatcher_create('BruteForce-Hamming')
```

```
match = fmatcher.match(descriptors1, descriptors2)
```

```
# Calculate planar homography and merge them
```

```
pts1, pts2 = [], []
```

```
for i in range(len(match)):
```

```
    pts1.append(keypoints1[match[i].queryIdx].pt)
```

```
    pts2.append(keypoints2[match[i].trainIdx].pt)
```

```
pts1 = np.array(pts1, dtype=np.float32)
```

```
pts2 = np.array(pts2, dtype=np.float32)
```

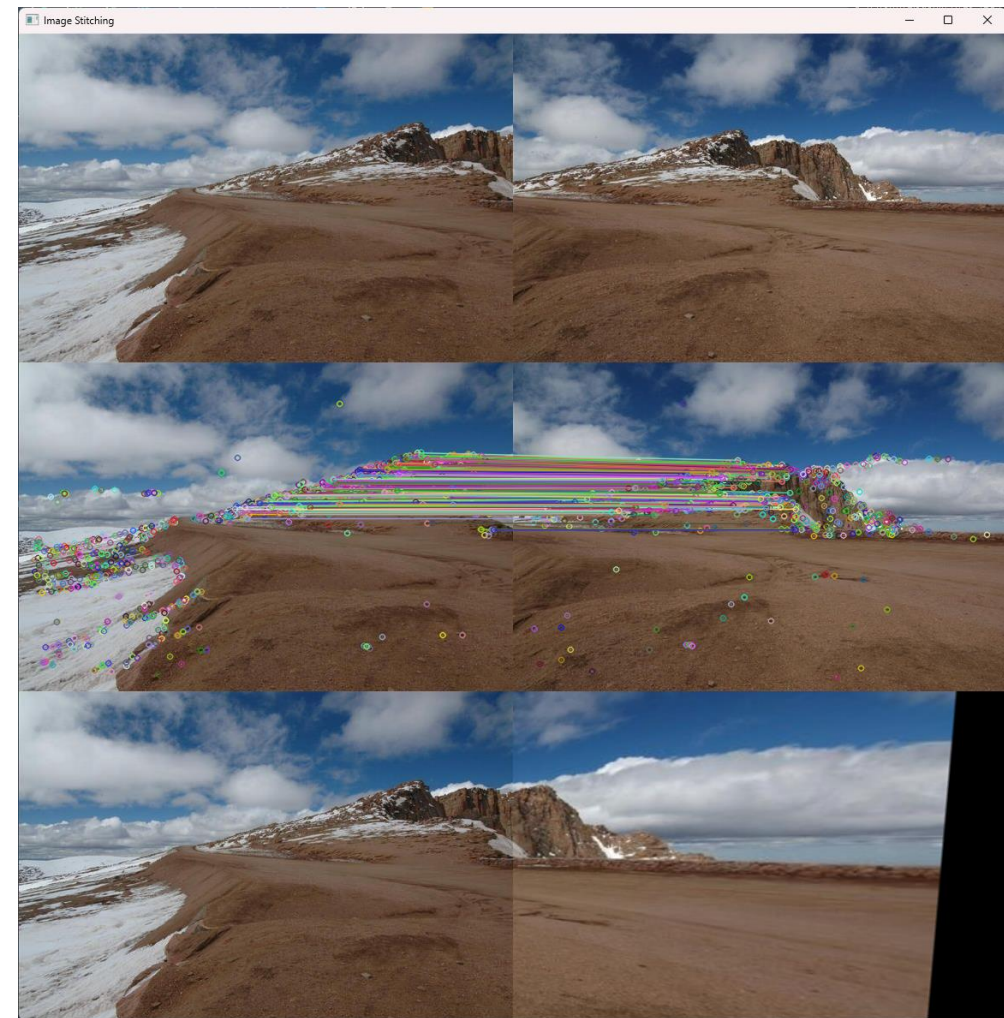
```
H, inlier_mask = cv.findHomography(pts2, pts1, cv.RANSAC)
```

```
img_merged = cv.warpPerspective(img2, H, (img1.shape[1]*2, img1.shape[0]))
```

```
img_merged[:, :img1.shape[1]] = img1 # Copy
```

```
# Show the merged image
```

```
img_matched = cv.drawMatches(img1, keypoints1, img2, keypoints2, match, None, None, None,
```



Planar Homography

- Example) **2D video stabilization** [video_stabilization.py]

```
# Open a video and get the reference image and feature points
```

```
video = cv.VideoCapture('../data/traffic.avi')
```

```
_, gray_ref = video.read()
```

```
if gray_ref.ndim >= 3:
```

```
    gray_ref = cv.cvtColor(gray_ref, cv.COLOR_BGR2GRAY)
```

```
pts_ref = cv.goodFeaturesToTrack(gray_ref, 2000, 0.01, 10)
```

```
# Run and show video stabilization
```

```
while True:
```

```
    # Read an image from `video`
```

```
    valid, img = video.read()
```

```
    if not valid:
```

```
        break
```

```
    if img.ndim >= 3:
```

```
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

```
    else:
```

```
        gray = img.copy()
```

```
# Extract optical flow and calculate planar homography
```

```
pts, status, err = cv.calcOpticalFlowPyrLK(gray_ref, gray, pts_ref, None)
```

```
H, inlier_mask = cv.findHomography(pts, pts_ref, cv.RANSAC)
```

```
# Synthesize a stabilized image
```

```
warp = cv.warpPerspective(img, H, (img.shape[1], img.shape[0]))
```

A shaking CCTV video



Planar Homography

- Assumption) **A plane** is observed by two views.
 - Perspective distortion correction: A complete plane
 - Planar image stitching: An approximated plane (\leftarrow distance \gg depth variation)
 - 2D video stabilization: An approximated plane (\leftarrow small motion)



Triangulation

- **Triangulation** (point localization)

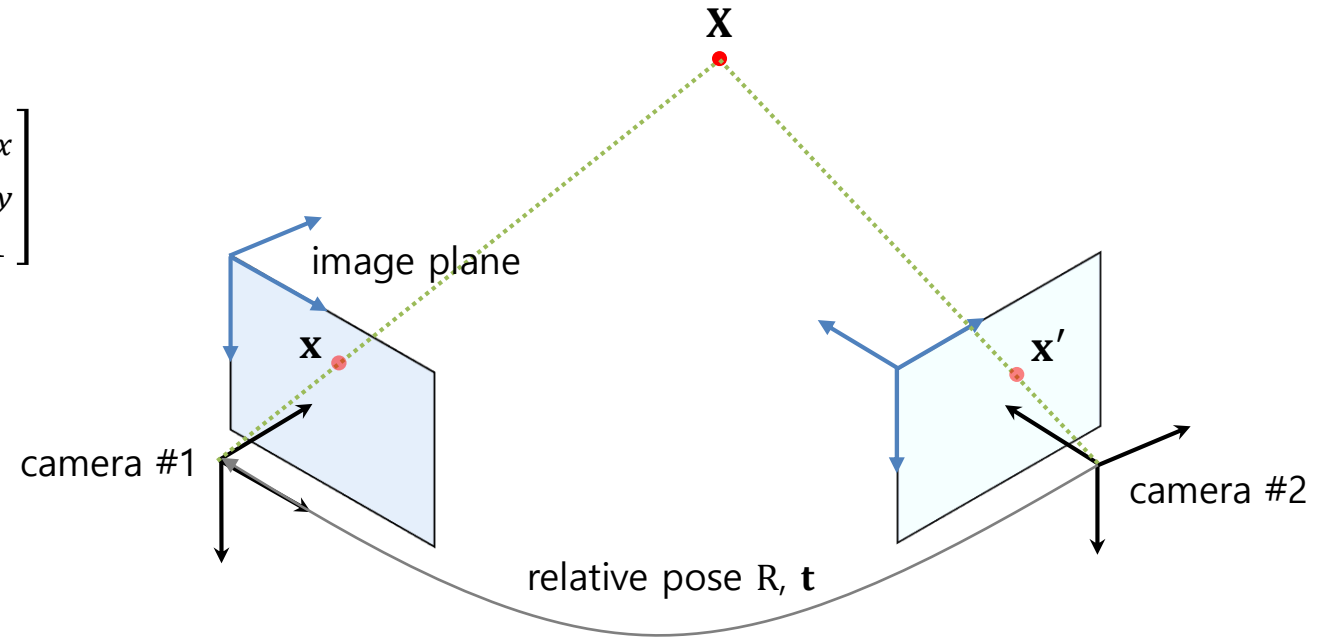
- Unknown: **Position of a 3D point \mathbf{X}** (3 DOF)
- Given: Point correspondence $(\mathbf{x}, \mathbf{x}')$, camera matrices (K, K') , and relative pose (R, \mathbf{t})
- Constraints: $\mathbf{x} = K [I | \mathbf{0}] \mathbf{X}$, $\mathbf{x}' = K' [R | \mathbf{t}] \mathbf{X}$
- Solution

- OpenCV `cv.triangulatePoints()`

- Special case) Stereo cameras

$$R = I_{3 \times 3}, \mathbf{t} = \begin{bmatrix} -b \\ 0 \\ 0 \end{bmatrix}, \text{ and } K = K' = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore Z = \frac{f}{x - x'} b$$



Triangulation

Example) Triangulation

```
f, cx, cy = 1000., 320., 240.
pts0 = np.loadtxt('../data/image_formation0.xyz')[::2]
pts1 = np.loadtxt('../data/image_formation1.xyz')[::2]
output_file = '../data/triangulation.xyz'
```

```
# Estimate relative pose of two view
```

```
F, _ = cv.findFundamentalMat(pts0, pts1, cv.FM_8POINT)
K = np.array([[f, 0, cx], [0, f, cy], [0, 0, 1]])
E = K.T @ F @ K
_, R, t, _ = cv.recoverPose(E, pts0, pts1)
```

```
# Reconstruct 3D points (triangulation)
```

```
P0 = K @ np.eye(3, 4, dtype=np.float32)
Rt = np.hstack((R, t))
P1 = K @ Rt
```

```
X = cv.triangulatePoints(P0, P1, pts0.T, pts1.T)   $\mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{X}$ 
```

```
X /= X[3]
```

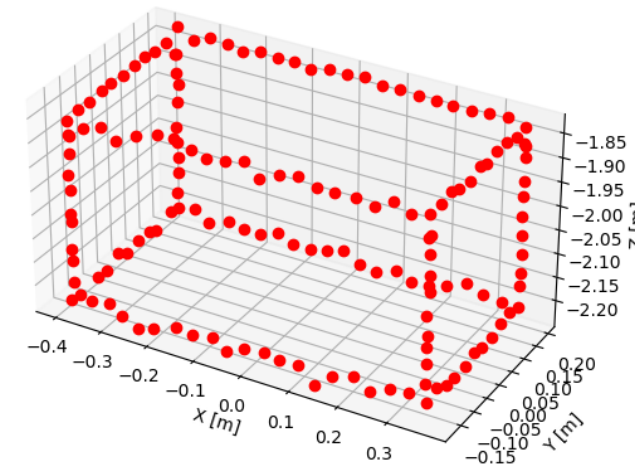
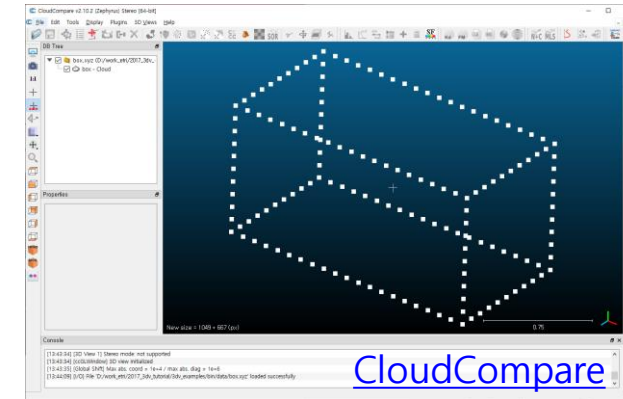
```
X = X.T
```

$$\mathbf{x}' = \mathbf{K}' \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X}$$

```
# Write the reconstructed 3D points
```

```
np.savetxt(output_file, X)
```

A point cloud: data/box.xyz



output_file: data/triangulation.xyz

Summary

- **Planar Homography:** $\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i$
 - Example) Perspective distortion correction
 - Example) Planar image stitching
 - Example) 2D video stabilization
- **Triangulation:** Finding \mathbf{X} (3 DOF)
 - Example) Triangulation