An Invitation to 3D Vision:
Two-View Geometry

Sunglok Choi, Assistant Professor, Ph.D.
Dept. of Computer Science and Engineering, 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]`
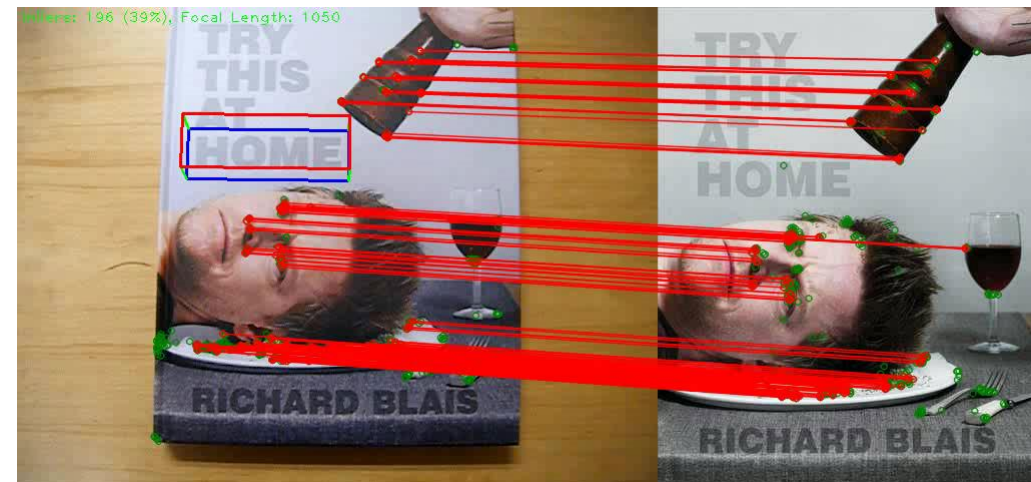
  (due to wrong initial K)

- Load the cover image
- Extract ORB features
- Build **3D object points** $X_1, X_2, \ldots$

Pass tentative matches

- Grab an input image
- Extract **ORB features** $x_1, x_2, \ldots$
- Match them to the object features

[Outlier Rejection]
- Remove wrong matches using
  `cv::findHomography()`

(free from K)

Pass correct matches

[Camera Pose Estimation]
- Estimate camera parameters using
  `cv::calibrateCamera()`

[Camera Pose Update]
- Intrinsic: K (initially known)
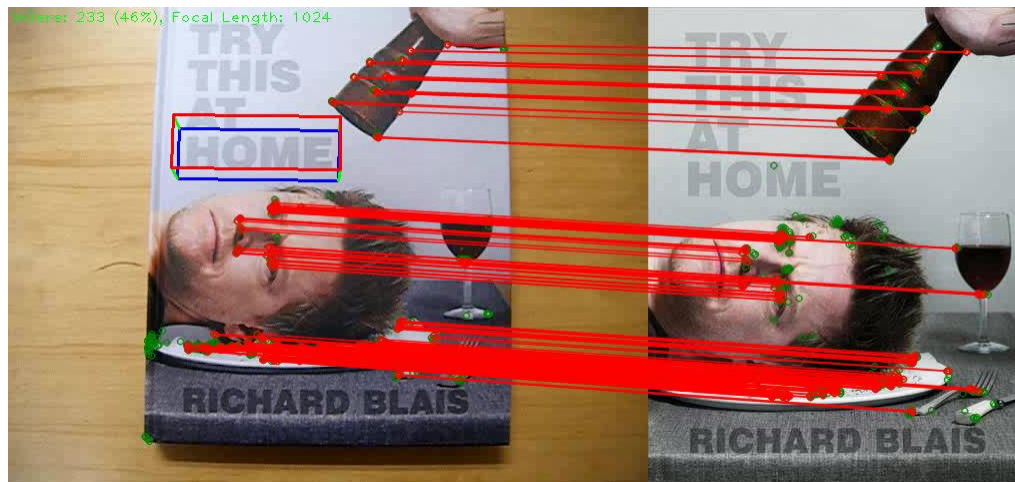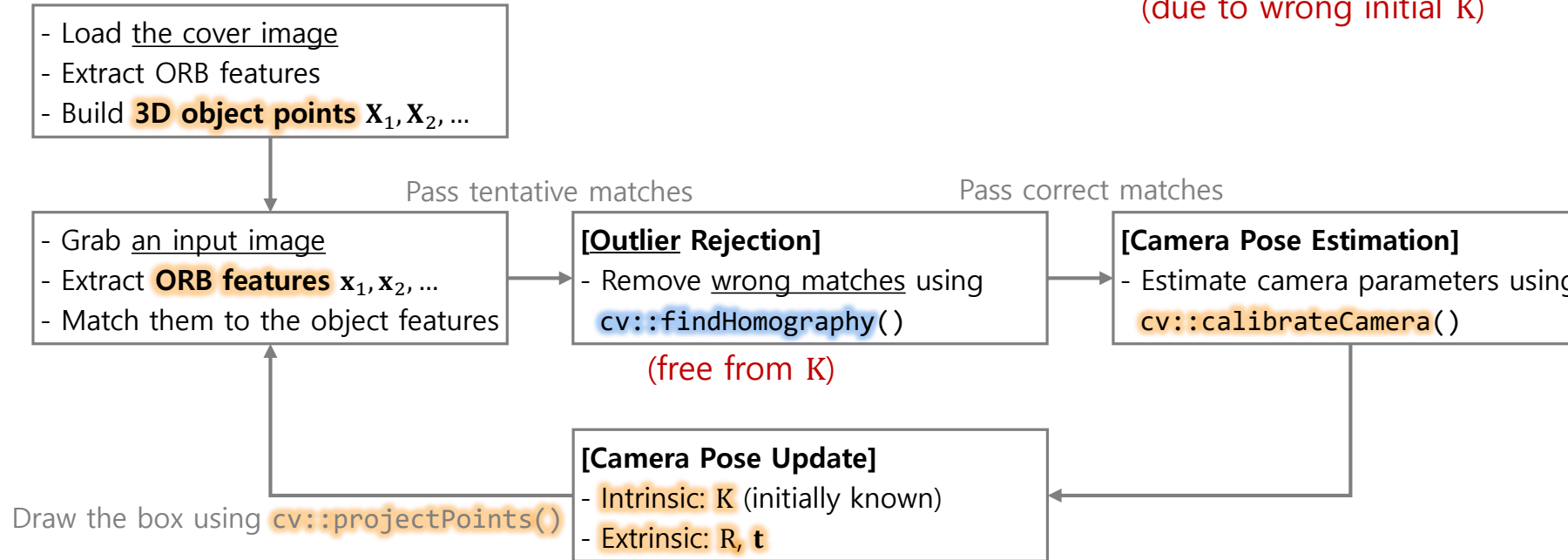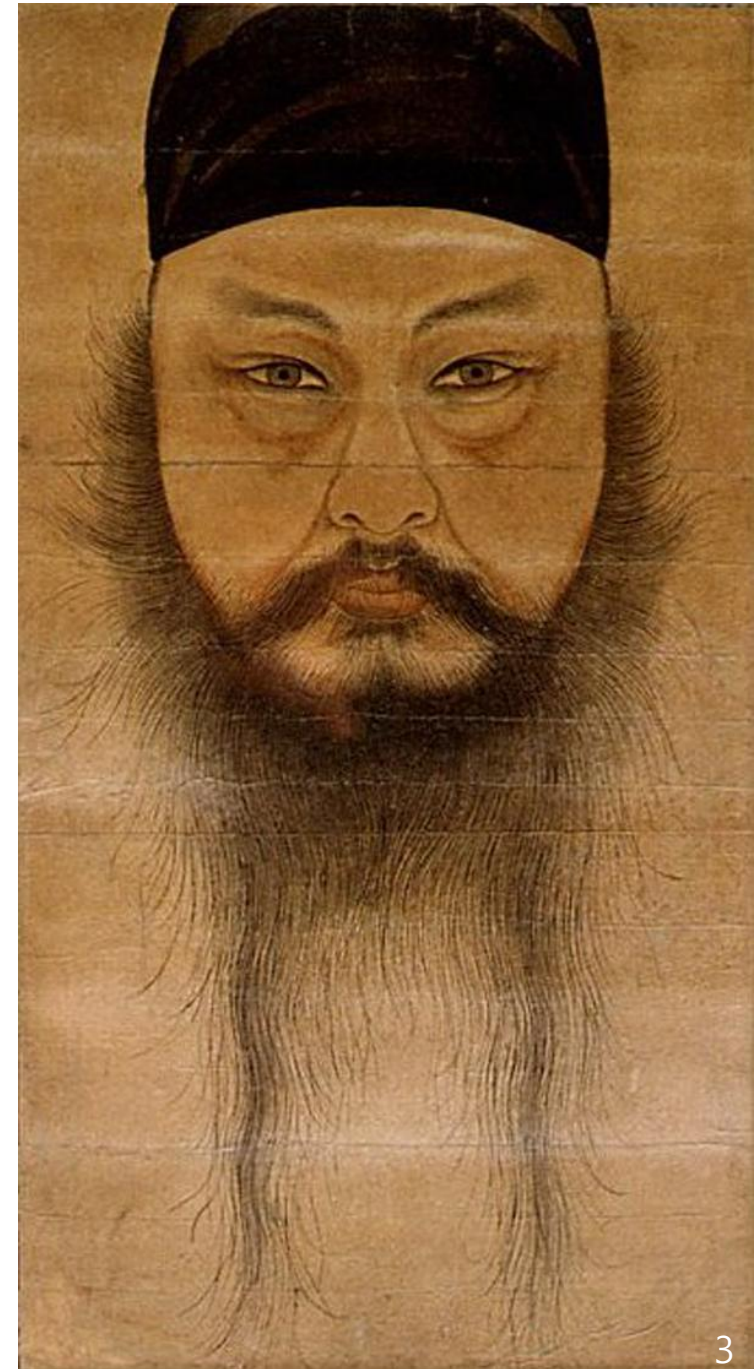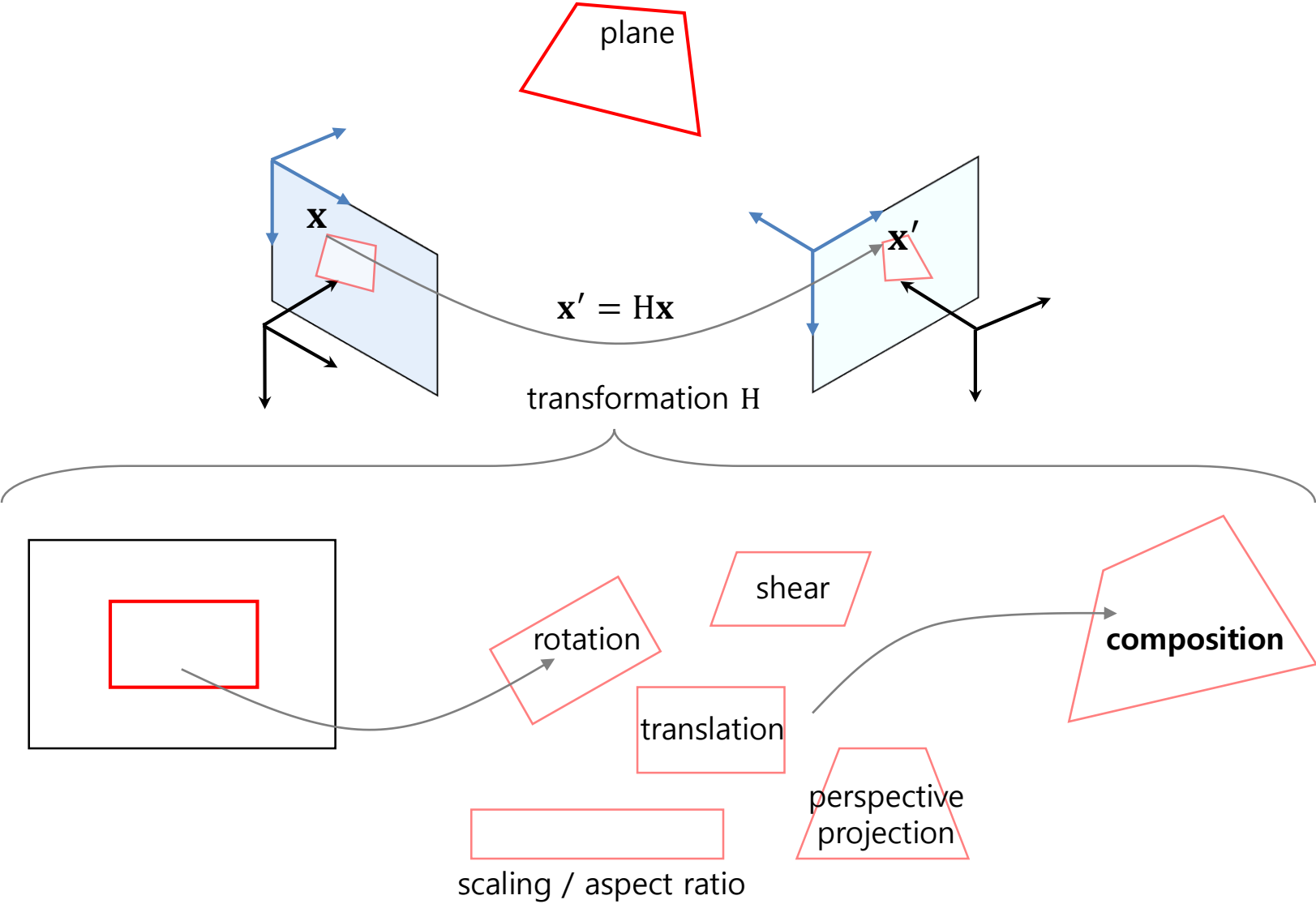- Extrinsic: R, t

Draw the box using `cv::projectPoints()`

Table of Contents: Two-view Geometry

- **Planar Homography**
- **Epipolar Geometry**
  - Epipolar constraint
  - Fundamental and essential matrix
- **Relative Camera Pose Estimation**
- **Triangulation**

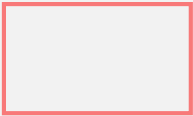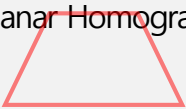윤두서(1668-1715) 자화상, 국보 제240호
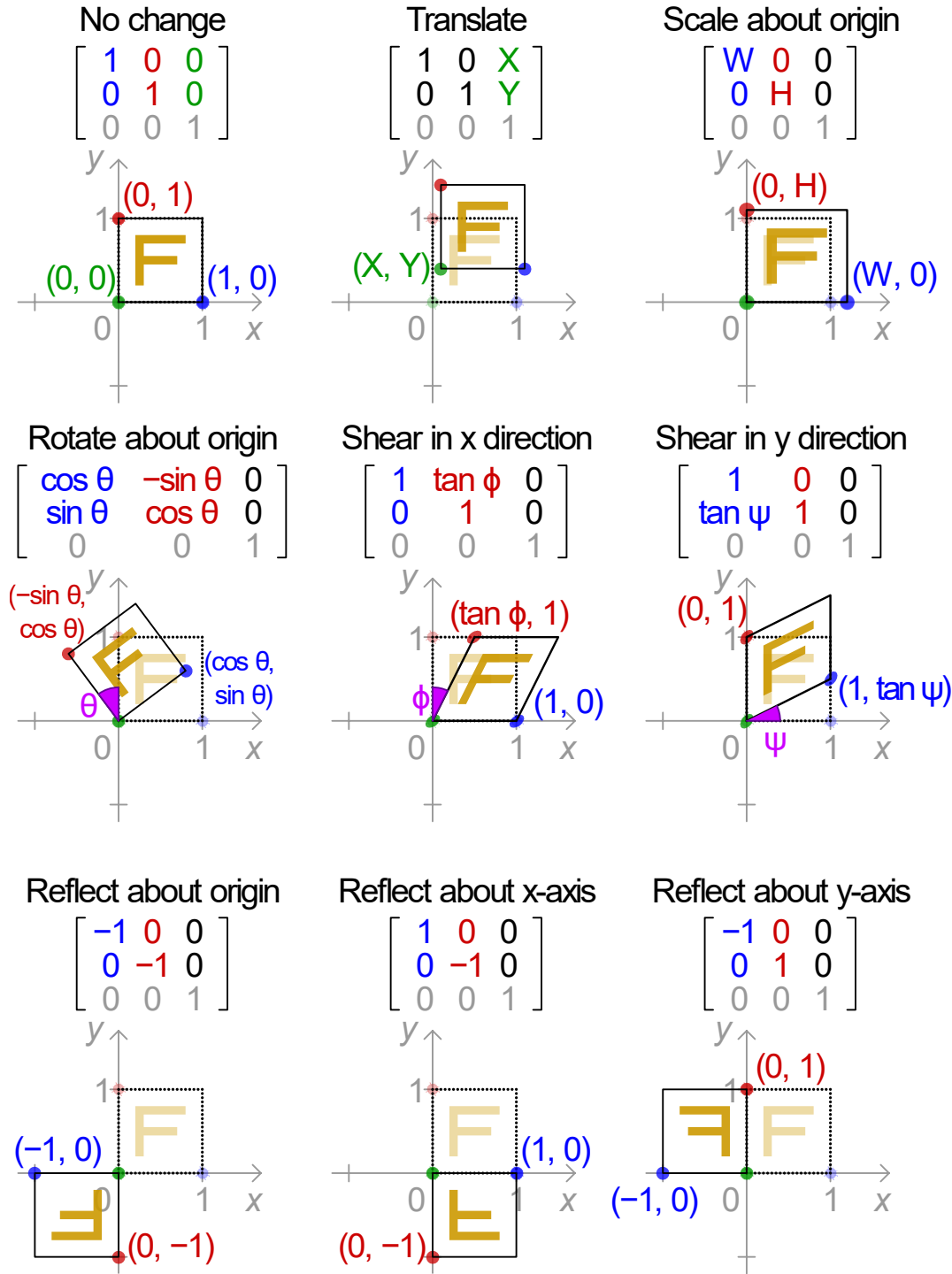Korean National Treasure No. 240

Planar Homography

plane

x

$$\mathbf{x}' = \mathbf{H}\mathbf{x}$$

x$'$

transformation H

rotation

shear

composition

translation

perspective projection

scaling / aspect ratio

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	O O X X X X	O O O X X X	O O O O O X	O O O O O O
Invariants - length - angle - ratio of lengths - parallelism - incidence - cross ratio	O O O O O O	X O O O O O	X X X O O O	X X X X O O

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

No change

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{bmatrix}$$

Scale about origin

$$\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotate about origin

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

Shear in x direction

$$\begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shear in y direction

$$\begin{bmatrix} 1 & 0 & 0 \\ \tan\psi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflect about origin

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflect about x-axis

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflect about y-axis

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Image: Wikipedia
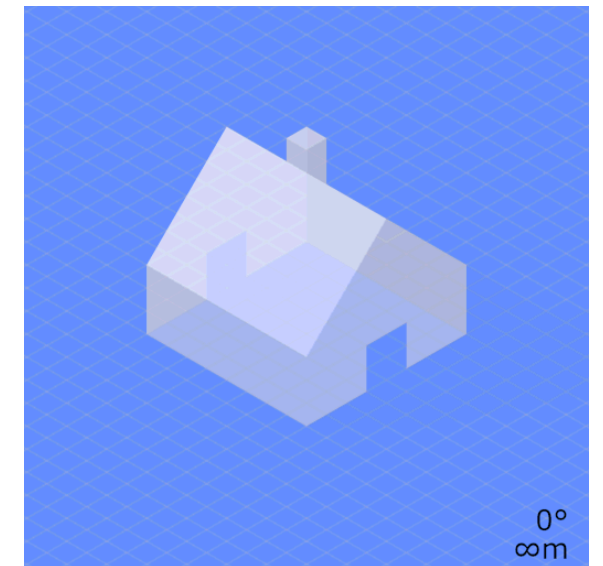
6

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 (e.g. CAD)

Image: Scatchpixel, Wikipedia

7

Planar Homography

▪ **Planar homography estimation**

- Unknown: Planar homography H (8 DOF)

- Given: Point correspondence $(\mathbf{x}_1, \mathbf{x}_1')$, ..., $(\mathbf{x}_n, \mathbf{x}_n')$

- Constraints: n x projective transformation $\mathbf{x}_i' = H\mathbf{x}_i$

- Solutions ($n \geq 4$) → 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.
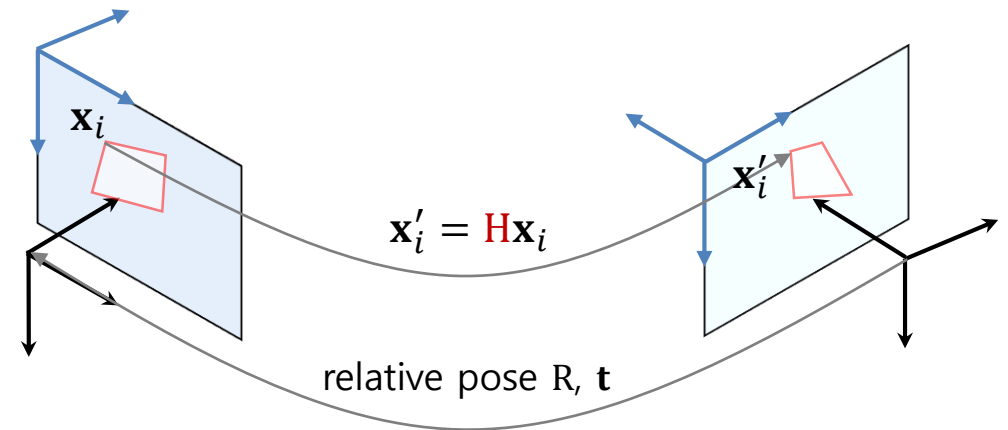
$\mathbf{x}_i$

$\mathbf{x}_i' = H\mathbf{x}_i$

$\mathbf{x}_i'$

relative pose R, **t**

Planar Homography

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

$$\mathbf{x}_i' = \mathbf{H}\mathbf{x}_i$$
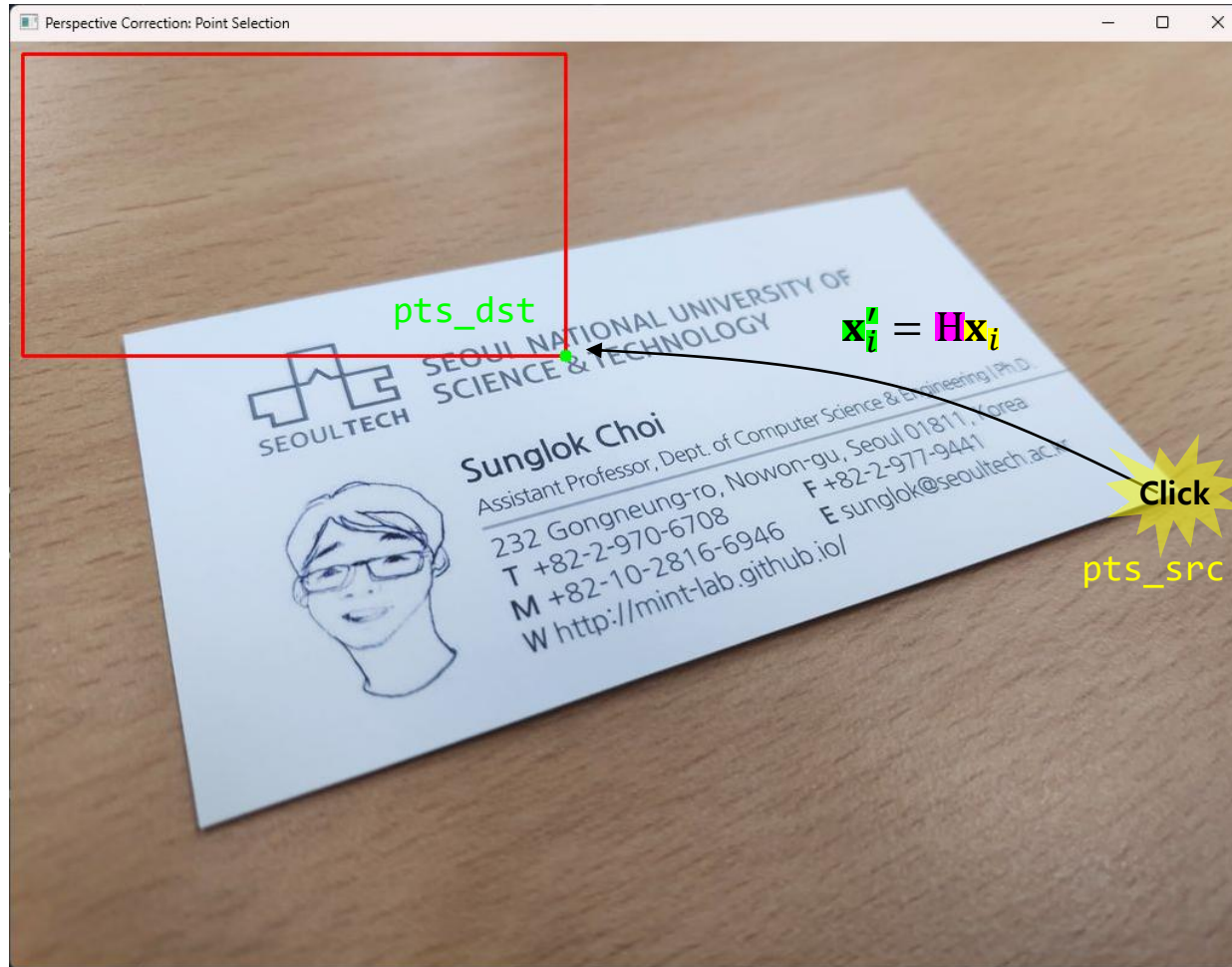
Planar Homography

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

```python
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]

```python
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]

```python
# Load two images
img1 = cv.imread('../data/hill01.jpg')
img2 = cv.imread('../data/hill02.jpg')

# Retrieve matching points
fdetector = cv.BRISK_create()
keypoints1, descriptors1 = fdetector.detectAndCompute(img1, None)
keypoints2, descriptors2 = fdetector.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,
```
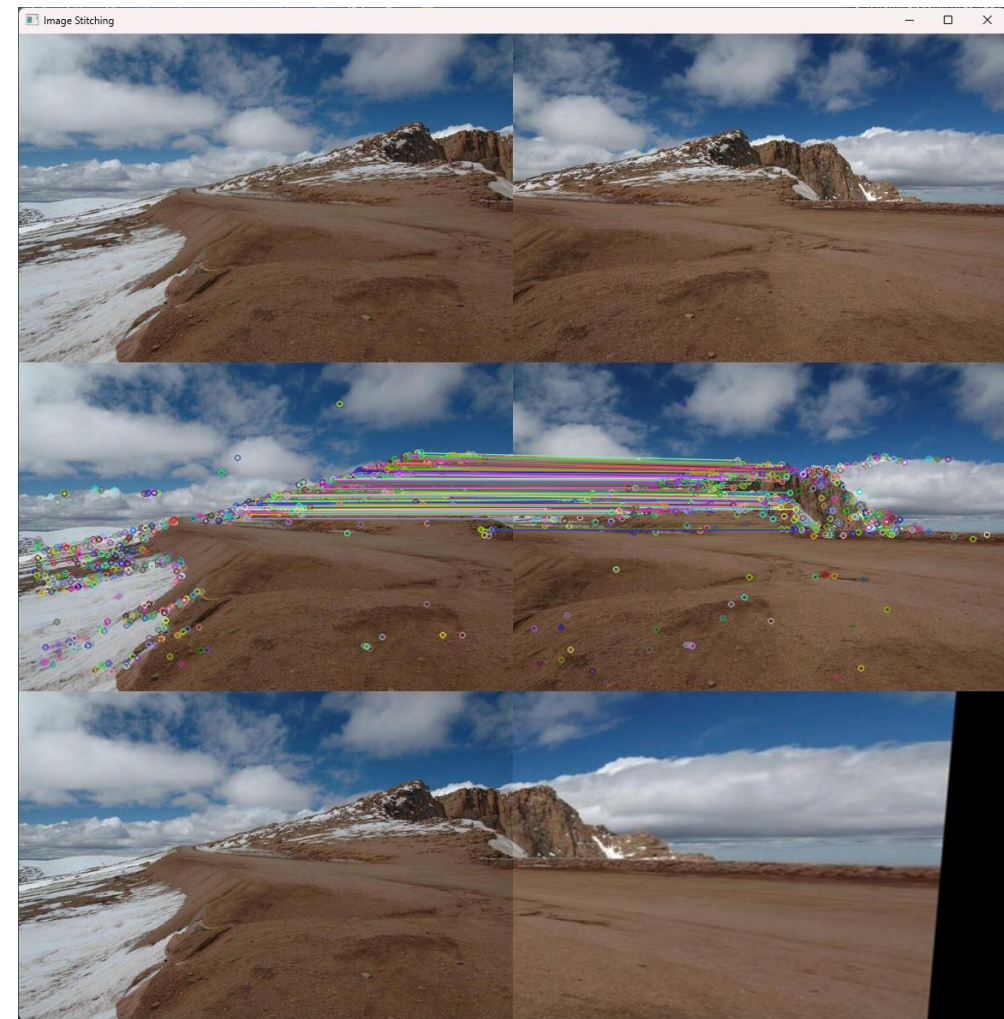
Planar Homography

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

```python
# 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, error = 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

(Red: Inlier, Green: Outlier)

13

Planar Homography

- Assumption) **A _plane_ is observed by two views.**
  - Perspective distortion correction: A complete plane

Planar Homography

- Assumption) **A <u>plane</u> is observed by two views.**

    – Perspective distortion correction: A complete plane

    – Planar image stitching: An approximated plane (← distance ≫ depth variation)

    – 2D video stabilization: An approximated plane (← small motion)
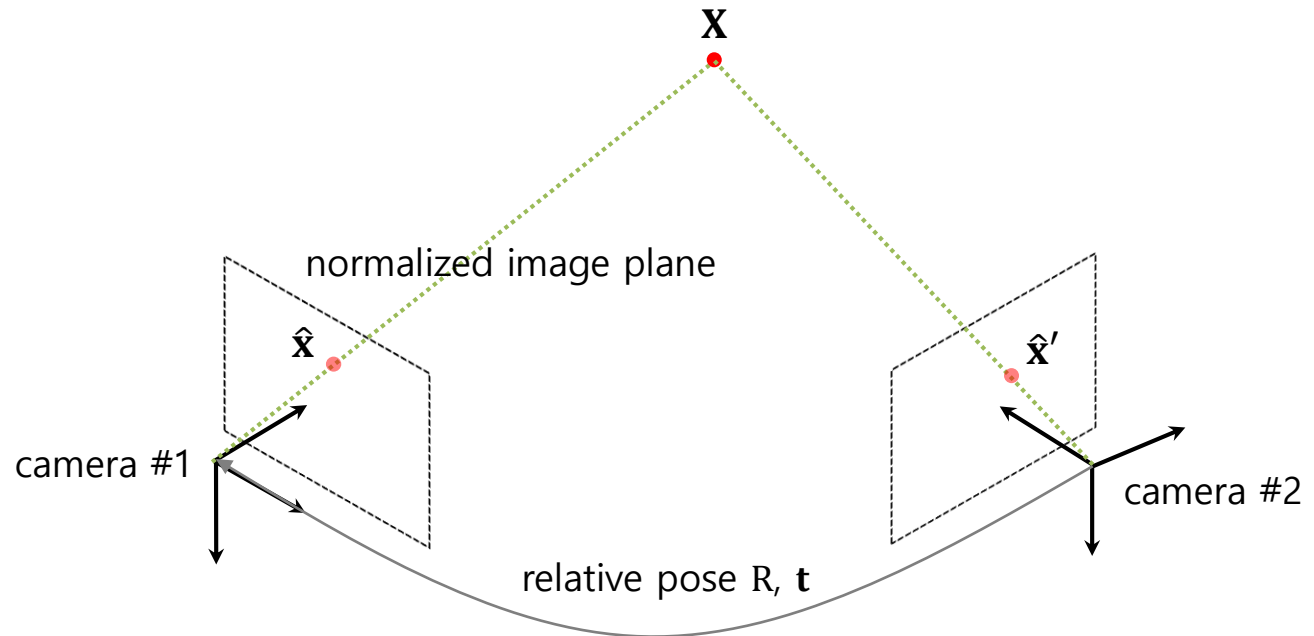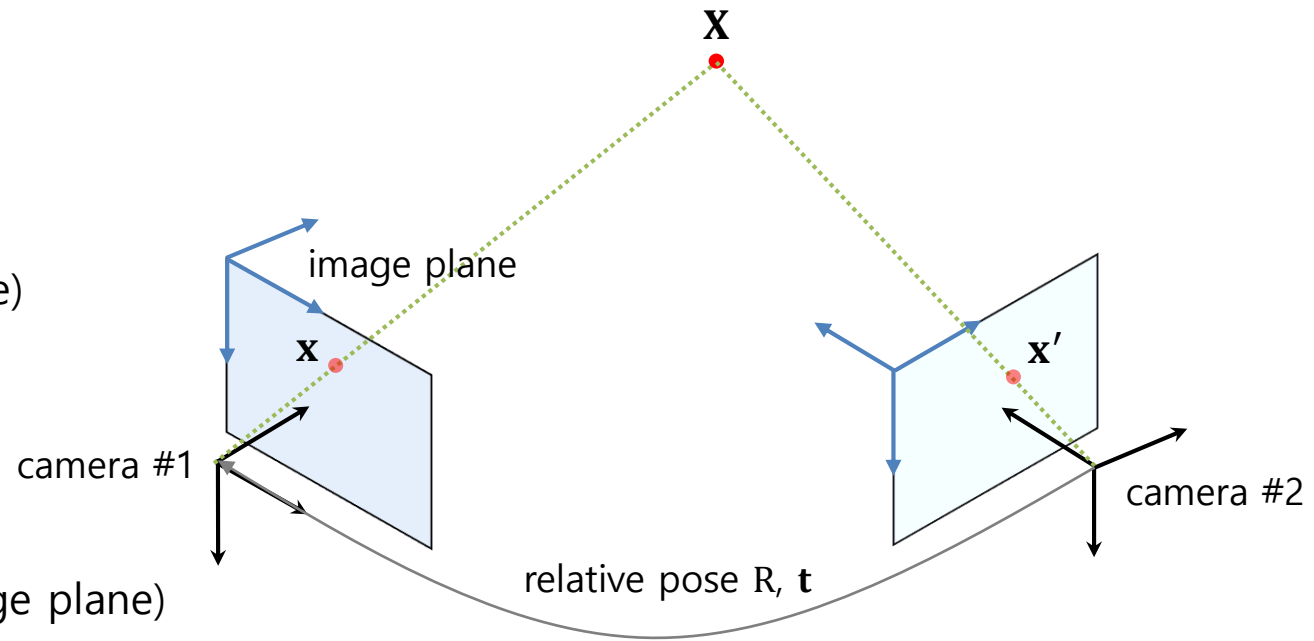
Epipolar Geometry

- **Epipolar constraint**

  $\mathbf{x'}^{\mathsf{T}}F\mathbf{x} = 0$ (F: Fundamental matrix on the image plane)

  $\updownarrow$ $(\mathbf{x} = K\hat{\mathbf{x}})$

  $\hat{\mathbf{x}}'^{\mathsf{T}}K'^{\mathsf{T}}FK\hat{\mathbf{x}} = 0$

  $\updownarrow$ $(E = K'^{\mathsf{T}}FK)$

  $\hat{\mathbf{x}}'^{\mathsf{T}}E\hat{\mathbf{x}} = 0$ (E: Essential matrix on the normalized image plane)

Epipolar Geometry

- **Epipolar constraint: Derivation**

$$\mathbf{X}' = R\mathbf{X} + \mathbf{t}$$

$$\downarrow \ (\mathbf{X} = \lambda \hat{\mathbf{x}})$$

$$\lambda' \hat{\mathbf{x}}' = \lambda R \hat{\mathbf{x}} + \mathbf{t}$$

$$\mathbf{t} \times \downarrow$$

$$\lambda' \mathbf{t} \times \hat{\mathbf{x}}' = \lambda \mathbf{t} \times R\hat{\mathbf{x}}$$

$$\hat{\mathbf{x}}' \cdot \downarrow$$
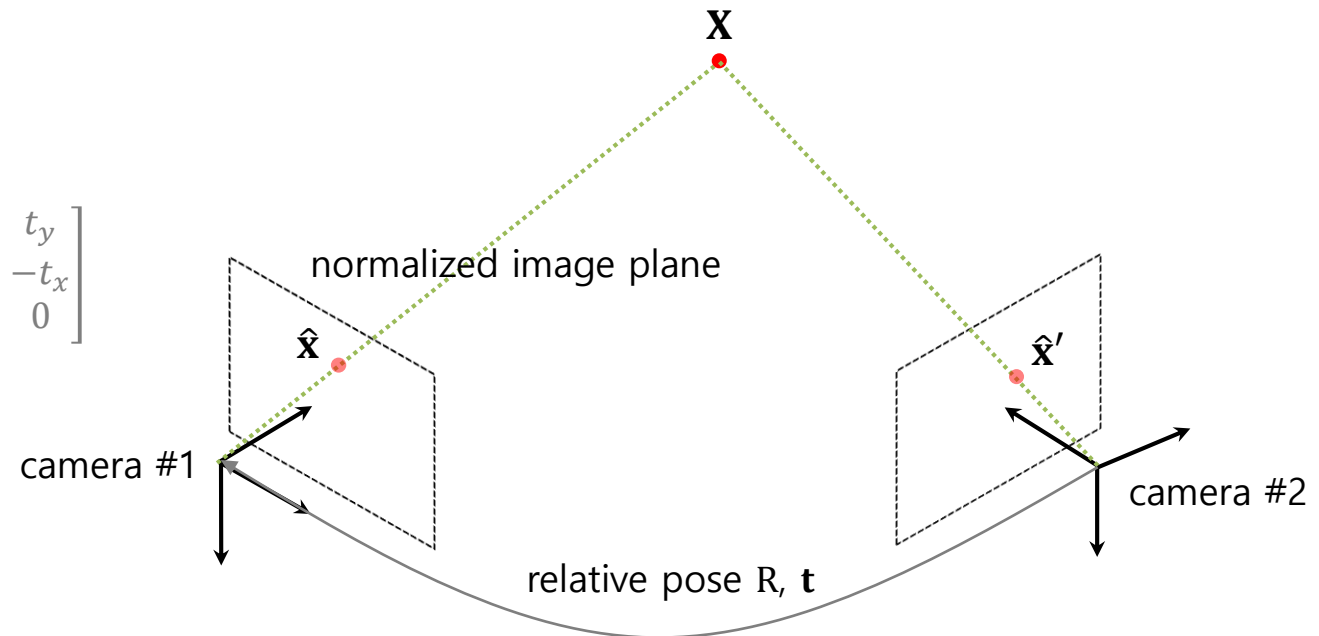
$$\lambda' \hat{\mathbf{x}}' \cdot (\mathbf{t} \times \hat{\mathbf{x}}') = \lambda \hat{\mathbf{x}}' \cdot \mathbf{t} \times R\hat{\mathbf{x}}$$

$$\downarrow \ (\hat{\mathbf{x}}' \cdot (\mathbf{t} \times \hat{\mathbf{x}}') = 0)$$

$$\lambda \hat{\mathbf{x}}' \cdot \mathbf{t} \times R\hat{\mathbf{x}} = 0$$

$$\downarrow \ (E = \mathbf{t} \times R \text{ or } E = [\mathbf{t}]_\times R) \quad \text{Note) } [\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

$$\hat{\mathbf{x}}' E \hat{\mathbf{x}} = 0$$

X

normalized image plane

$\hat{\mathbf{x}}$

$\hat{\mathbf{x}}'$

camera #1

camera #2

relative pose R, **t**

Epipolar Geometry

- **Epipolar geometry**
  - Baseline
    - Distance between two camera centers, $\mathbf{C}$ and $\mathbf{C}'$
  - Epipolar plane
    - Plane generated from three points, $\mathbf{X}$, $\mathbf{C}$, and $\mathbf{C}'$
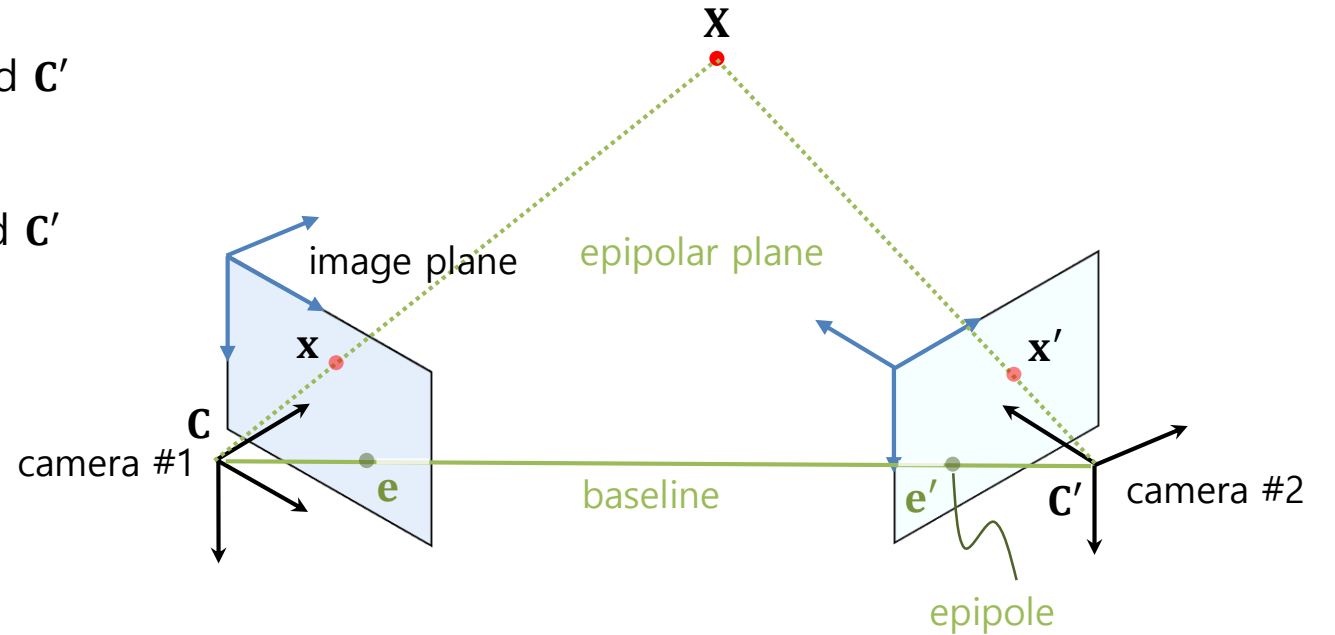 - Epipole (a.k.a. epipolar point)
    - Projection of other camera centers
    - $\mathbf{e} = \mathrm{P}\mathbf{C}'$ and $\mathbf{e}' = \mathrm{P}'\mathbf{C}$
      - e.g. $\mathrm{P} = \mathrm{K}[\mathrm{I}|0]$ and $\mathrm{P}' = \mathrm{K}'[\mathrm{R}|\mathbf{t}]$
        - $\mathbf{e} = -\mathrm{K}\mathrm{R}^{\mathsf{T}}\mathbf{t}$ and $\mathbf{e}' = \mathrm{K}'\mathbf{t}$

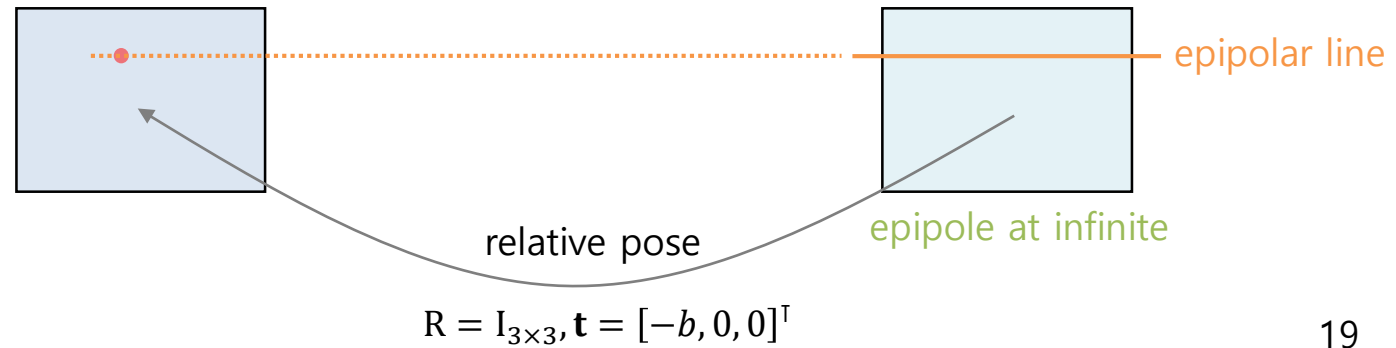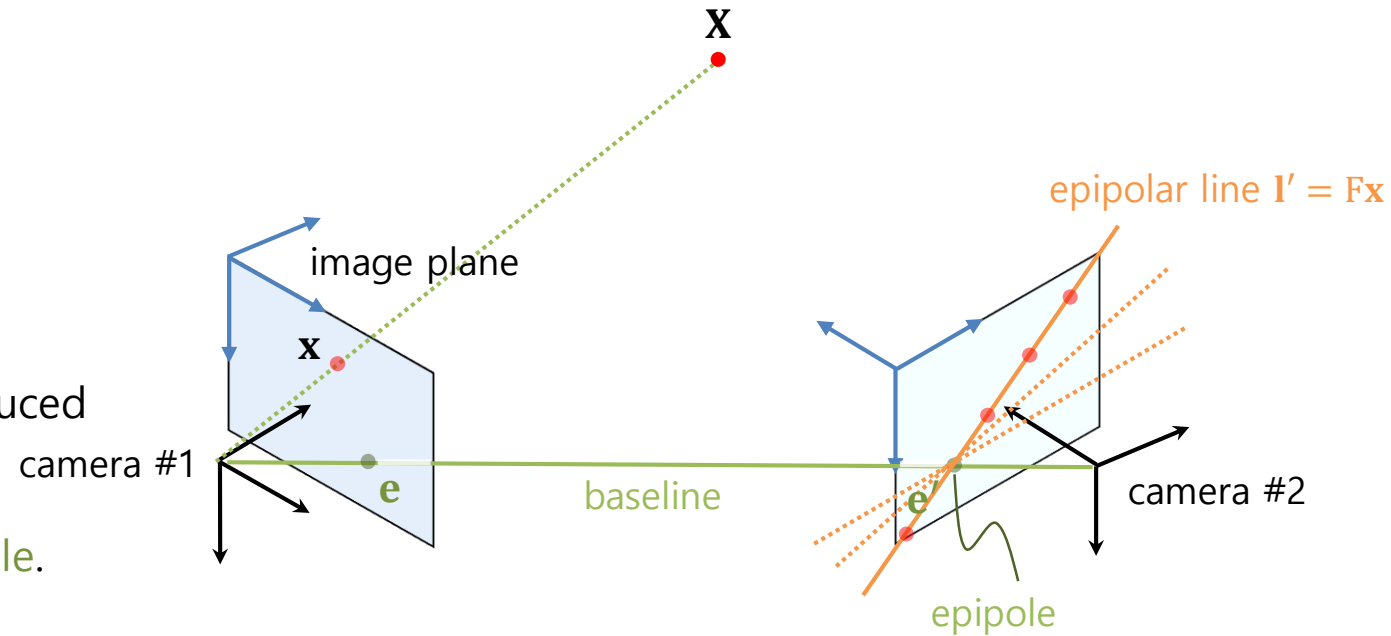# Epipolar Geometry

- **Epipolar geometry**
  - Epipolar line
    - $\mathbf{x'}^\top \mathrm{F}\mathbf{x} = 0 \rightarrow \mathbf{x'}^\top \mathbf{l'} = 0$ where $\mathbf{l'} = \mathrm{F}\mathbf{x}$
      - $\mathbf{x'}$ will lie on the line $\mathbf{l'}$.
    - $\mathbf{x'}^\top \mathrm{F}\mathbf{x} = 0 \rightarrow \mathbf{l}^\top \mathbf{x} = 0$ where $\mathbf{l} = \mathrm{F}^\top \mathbf{x'}$
      - $\mathbf{x}$ will lie on the line $\mathbf{l}$.
    - The search space of feature matching is reduced from a image plane to the epipolar line.
  - Note) Every epipolar line intersects at the epipole.
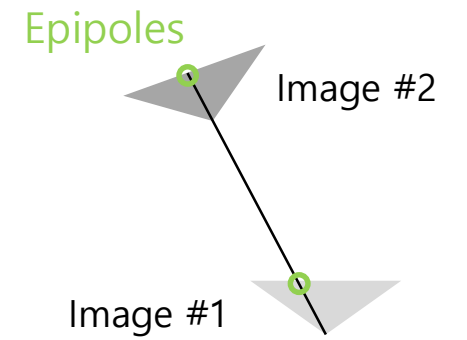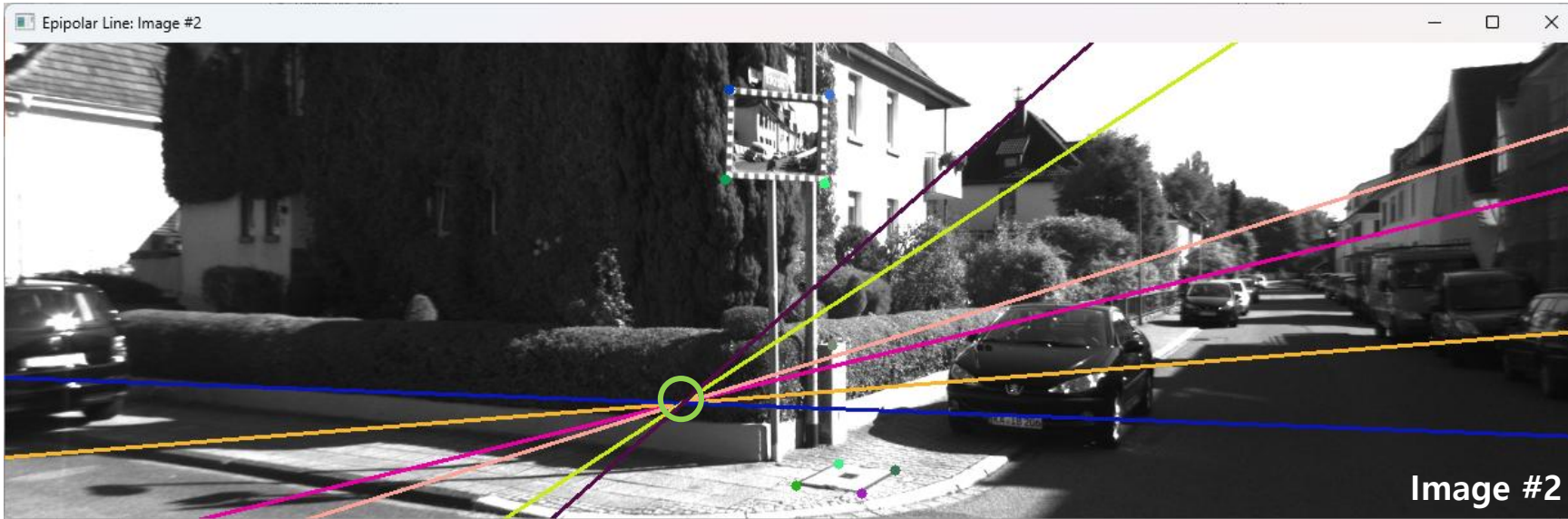   - $\mathrm{F}\mathbf{e} = 0$
    - $\mathbf{e}$ is the null space of F.

Special case) Stereo cameras

$$\mathrm{R} = \mathrm{I}_{3\times3}, \mathbf{t} = [-b, 0, 0]^\top$$

Epipolar Geometry

- Example) **Epipolar line visualization** `[epipolar_line_visualization.py]`
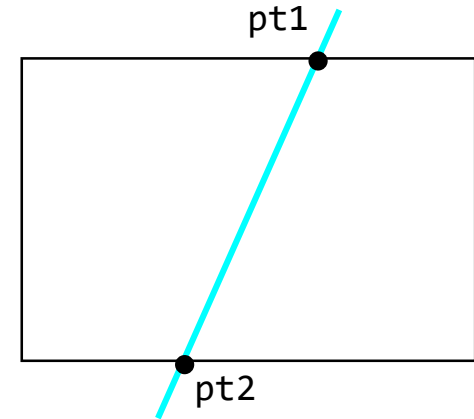
Epipolar Geometry

- Example) **Epipolar line visualization** [epipolar_line_visualization.py]

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

def draw_straight_line(img, line, color, thickness=1):
    h, w, *_ = img.shape
    a, b, c = line # Line: ax + by + c = 0
    if abs(a) > abs(b):
        pt1 = (int(c / -a), 0)
        pt2 = (int((b*h + c) / -a), h)
    else:
        ...
    cv.line(img, pt1, pt2, color, thickness)

if __name__ == '__main__':
    # Load two images
    img1 = cv.imread('../data/KITTI07/image_0/000000.png', cv.IMREAD_COLOR)
    img2 = cv.imread('../data/KITTI07/image_0/000023.png', cv.IMREAD_COLOR)
    # Note) `F` is derived from `fundamental_mat_estimation.py`.
    F = np.array([[ 3.34638533e-07,  7.58547151e-06, -2.04147752e-03], ...])

    # Register event handlers and show images
    wnd1_name, wnd2_name = 'Epipolar Line: Image #1', 'Epipolar Line: Image #2'
    img1_pts, img2_pts = [], []
    cv.namedWindow(wnd1_name)
    cv.setMouseCallback(wnd1_name, mouse_event_handler, img1_pts)
```


21

Epipolar Geometry

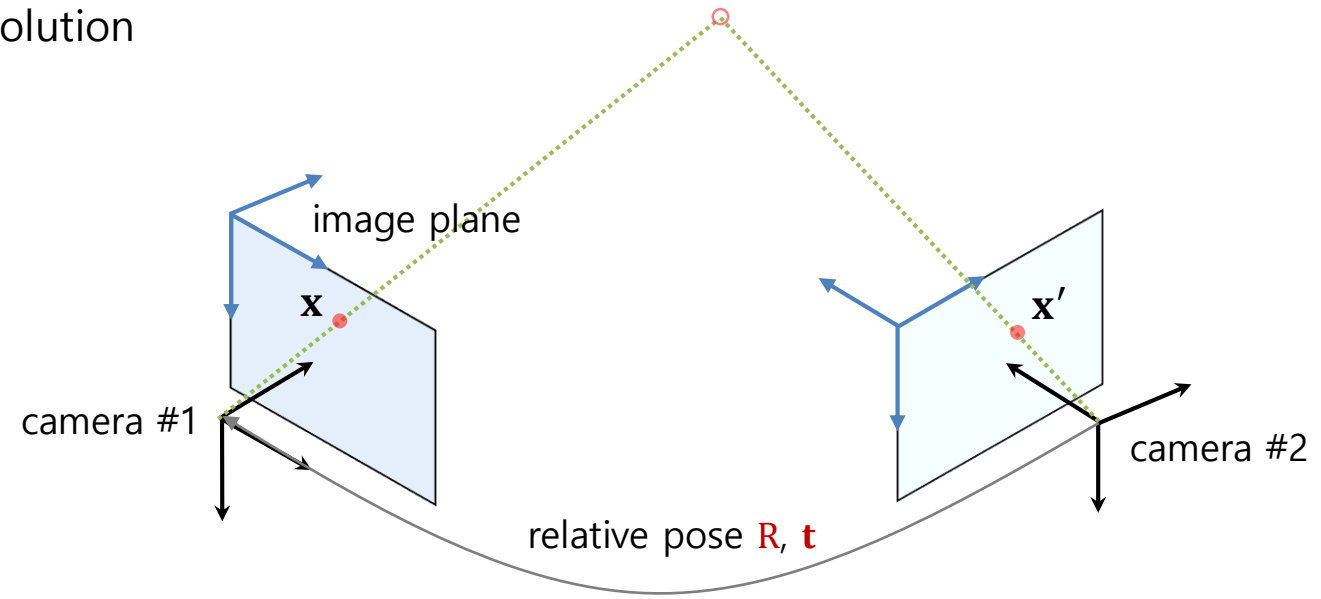▪ Example) **Epipolar line visualization** [epipolar_line_visualization.py]

```python
if __name__ == '__main__':
    # Load two images
    img1 = cv.imread('../data/KITTI07/image_0/000000.png', cv.IMREAD_COLOR)
    img2 = cv.imread('../data/KITTI07/image_0/000023.png', cv.IMREAD_COLOR)
    F = np.array([[ 3.34638533e-07,   7.58547151e-06, -2.04147752e-03], ...])

    # Register event handlers and show images
    ...

    # Get a point from a image and draw its correponding epipolar line on the other image
    while True:
        if len(img1_pts) > 0:
            for x, y in img1_pts:
                color = (random.randrange(256), random.randrange(256), random.randrange(256))
                cv.circle(img1, (x, y), 4, color, -1)
                epipolar_line = F @ [[x], [y], [1]]
                draw_straight_line(img2, epipolar_line, color, 2)
            img1_pts.clear()
        if len(img2_pts) > 0:
            for x, y in img2_pts:
                color = (random.randrange(256), random.randrange(256), random.randrange(256))
                cv.circle(img2, (x, y), 4, color, -1)
                epipolar_line = F.T @ [[x], [y], [1]]
                draw_straight_line(img1, epipolar_line, color, 2)
            img2_pts.clear()
        cv.imshow(wnd2_name, img2)
```

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

$$\mathbf{l} = F^\top \mathbf{x'}$$

22

Relative Camera Pose Estimation

- **Relative camera pose estimation** (~ fundamental matrix estimation)

  - Unknown: Rotation and translation R, **t** (**5 DOF**; up-to scale **"scale ambiguity"**)

  - Given: Point correspondence $(\mathbf{x}_1, \mathbf{x}_1')$, ..., $(\mathbf{x}_n, \mathbf{x}_n')$ and camera matrices K, K′

  - Constraints: n x epipolar constraint ($\mathbf{x}'^\top F \mathbf{x} = 0$ or $\hat{\mathbf{x}}'^\top E \hat{\mathbf{x}} = 0$)

  - Solutions) **Fundamental matrix**: 7/8-point algorithm (**7 DOF**; intrinsic+extrinsic)

    - **Properties**: $\det(F) = 0$ (or $\text{rank}(F) = 2$)

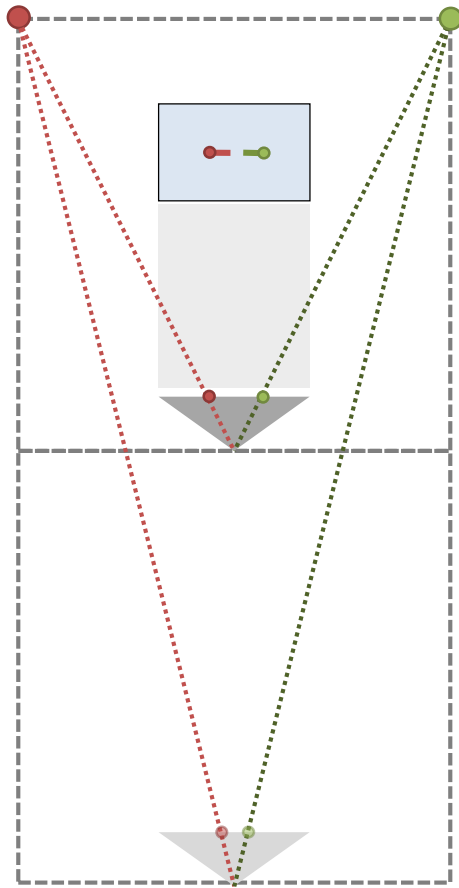   - **Estimation**: `cv.findFundamentalMat()` → 1 solution

camera #1

image plane

$\mathbf{x}$

$\mathbf{x}'$

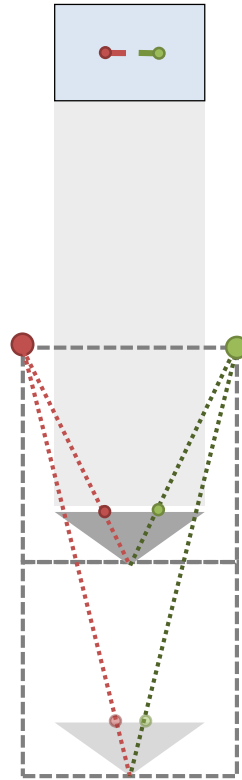camera #2

relative pose R, **t**

Relative Camera Pose Estimation

- **Relative camera pose estimation** (~ fundamental/essential matrix estimation)

  - Unknown: Rotation and translation R, **t** (**5 DOF**; up-to scale **"scale ambiguity"**)

  - Given: Point correspondence $(\mathbf{x}_1, \mathbf{x}'_1)$, ..., $(\mathbf{x}_n, \mathbf{x}'_n)$ and camera matrices K, K′

  - Constraints: n x epipolar constraint ($\mathbf{x}'^\top F \mathbf{x} = 0$ or $\hat{\mathbf{x}}'^\top E \hat{\mathbf{x}} = 0$)

  - Solutions) **Fundamental matrix**: 7/8-point algorithm (**7 DOF**; intrinsic+extrinsic)

    - **Properties**: $\det(F) = 0$ (or $\text{rank}(F) = 2$)

    - **Estimation**: `cv.findFundamentalMat()` → 1 solution

    - **Conversion to E**: $E = K'^\top F K$

  - Solutions) **Essential matrix**: 5-point algorithm (**5 DOF**; extrinsic)

    - **Properties**: $\det(E) = 0$ and $2EE^\top E - \text{tr}(EE^\top) E = 0$

    - **Estimation**: `cv.findEssentialMat()` → k solutions

    - **Decomposition**: `cv.decomposeEssentialMat()` → 4 solutions **"relative pose ambiguity"**

    - **Decomposition with positive-depth check**: `cv.recoverPose()` → 1 solution

R, **t**
$\updownarrow$ $(E = [\mathbf{t}]_\times R)$
E
$\updownarrow$ $(E = K'^\top F K)$
F

Relative Camera Pose Estimation: Scale Ambiguity

(a) 2-meter-wide tunnel

(b) 1-meter-wide tunnel

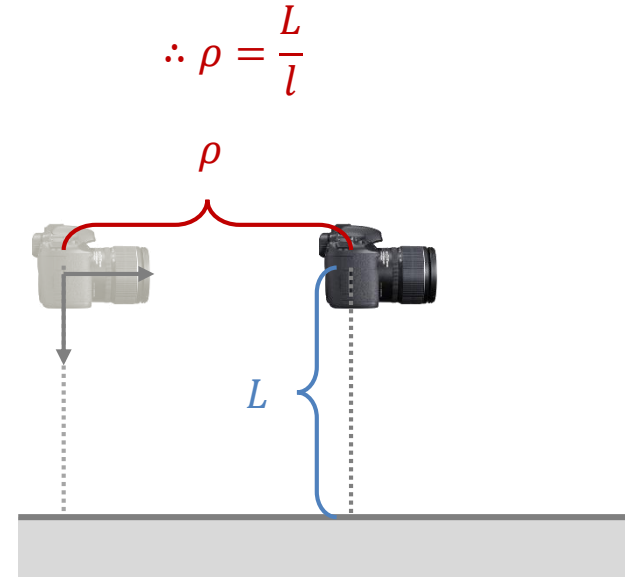The Sandcrawler @ Star Wars IV: A New Hope (1977)

Reference: Choi et al., "Resolving Scale Ambiguity in Monocular Visual Odometry", URAI, 2013

Relative Camera Pose Estimation: Scale Ambiguity

- **How to resolve scale ambiguity**
    - **Additional sensors**: Speedometers (odometers), IMUs, GPSs, depth/distance (stereo, RGB-D, LiDAR, …)
    - **Motion constraints**: Known initial translation, Ackerman's steering kinematics
    - **Observation constraints**: Known size of objects, known and constant height of camera

$$\therefore \rho = \frac{L}{l}$$
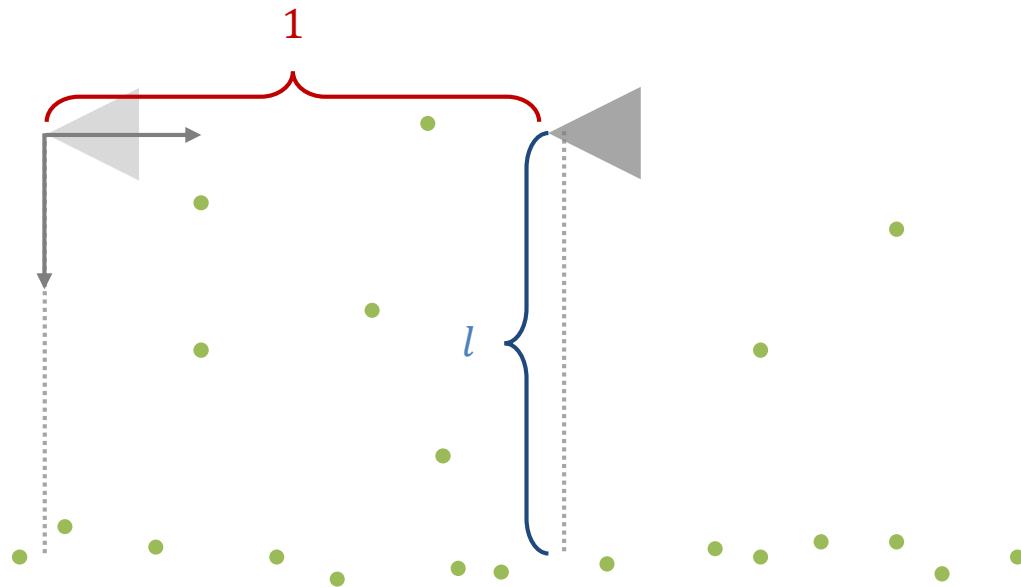
(a) The reconstructed world

(b) The real world

Reference: Choi et al., "Resolving Scale Ambiguity in Monocular Visual Odometry", URAI, 2013
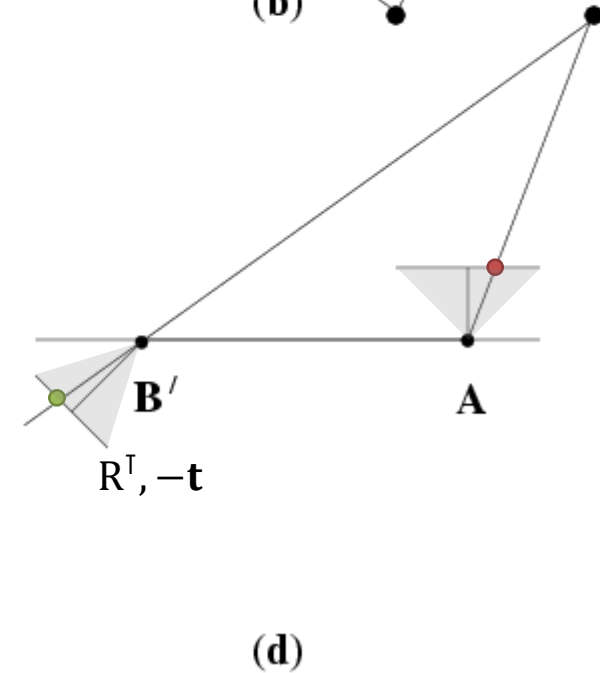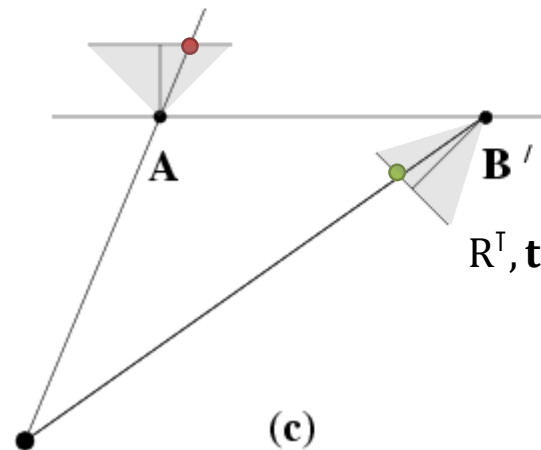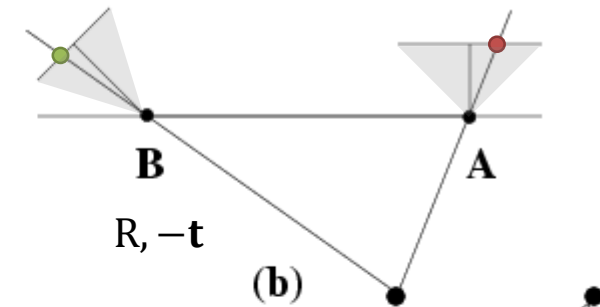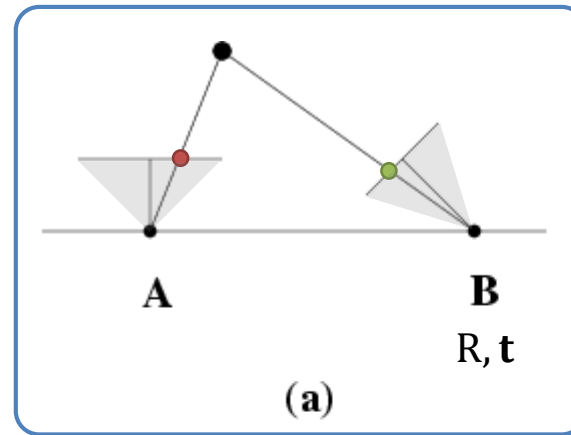
Relative Camera Pose Estimation: Relative Pose Ambiguity

▪ **How to resolve pose ambiguity**

   – Positive depth constraint

Image: Hartley and Zisserman, "Multiple View Geometry in Computer Vision" (2nd Edition), Cambridge University Press, 2004

Relative Camera Pose Estimation

Example) **Fundamental matrix estimation** [fundamental_mat_estimation.py]

```python
# Load two images
img1 = cv.imread('../data/KITTI07/image_0/000000.png')
img2 = cv.imread('../data/KITTI07/image_0/000023.png')
f, cx, cy = 707.0912, 601.8873, 183.1104 # From the KITTI dataset
K = np.array([[f, 0, cx], [0, f, cy], [0, 0, 1]])

# Retrieve matching points
fdetector = cv.BRISK_create()
keypoints1, descriptors1 = fdetector.detectAndCompute(img1, None)
keypoints2, descriptors2 = fdetector.detectAndCompute(img2, None)

fmatcher = cv.DescriptorMatcher_create('BruteForce-Hamming')
match = fmatcher.match(descriptors1, descriptors2)
```
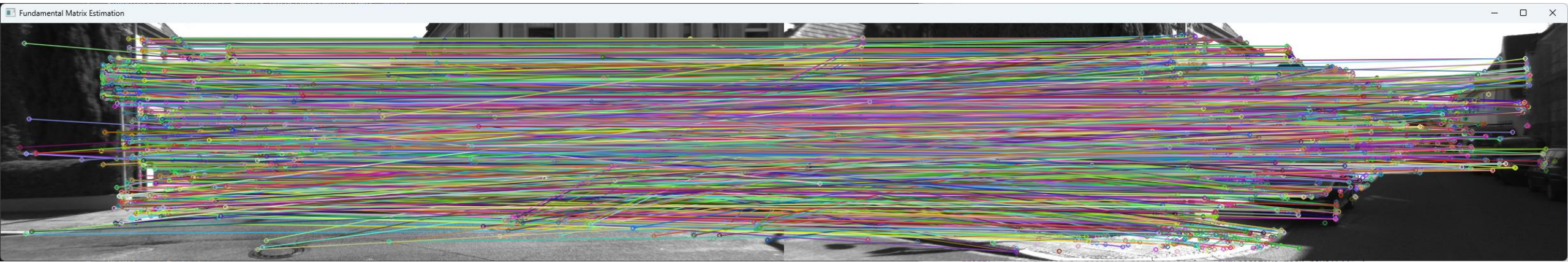

Fundamental Matrix Estimation

Relative Camera Pose Estimation

Example) **Fundamental matrix estimation** [fundamental_mat_estimation.py]

```python
# Load two images
...

# Retrieve matching points
...

# Calculate the fundamental matrix
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)
F, inlier_mask = cv.findFundamentalMat(pts1, pts2, cv.FM_RANSAC, 0.5, 0.999)
```
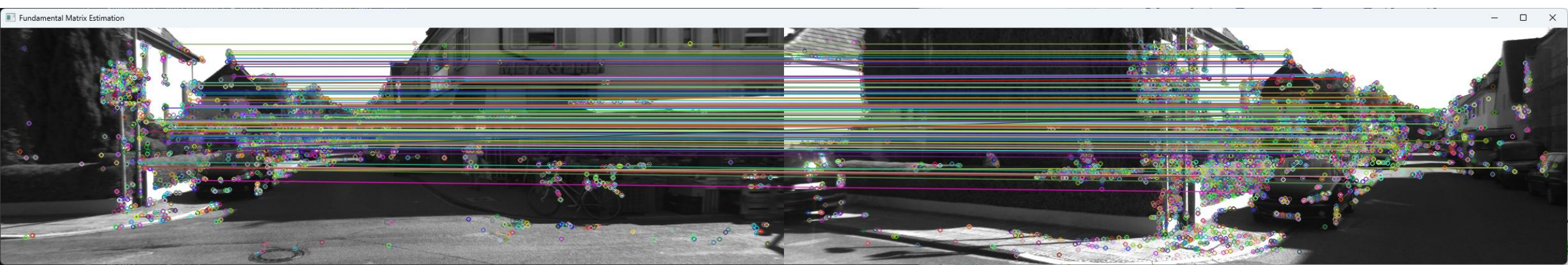
Relative Camera Pose Estimation

Example) **Fundamental matrix estimation** [fundamental_mat_estimation.py]

```python
# Load two images
...
f, cx, cy = 707.0912, 601.8873, 183.1104 # From the KITTI dataset
K = np.array([[f, 0, cx], [0, f, cy], [0, 0, 1]])

# Retrieve matching points
...

# Calculate the fundamental matrix
...
F, inlier_mask = cv.findFundamentalMat(pts1, pts2, cv.FM_RANSAC, 0.5, 0.999)

# Extract relative camera pose between two images
E = K.T @ F @ K
positive_num, R, t, positive_mask = cv.recoverPose(E, pts1, pts2, K, mask=inlier_mask)
...
print(f'* The position of Image #2 = {-R.T @ t}') # [-0.57, 0.09, 0.82]
```
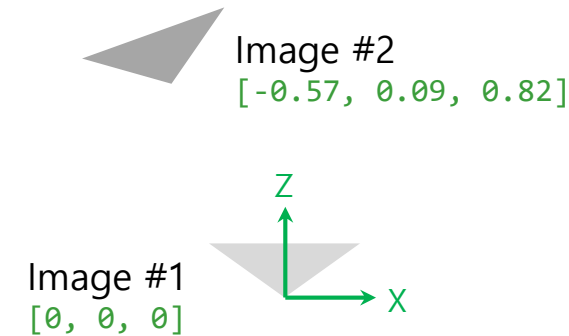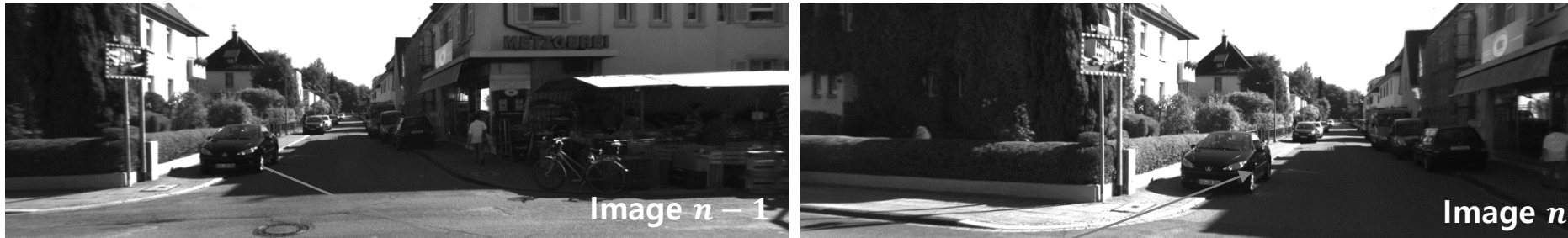
Image #2
[-0.57, 0.09, 0.82]

Image #1
[0, 0, 0]

Z

X

Relative Camera Pose Estimation

▪ Example) **Monocular Visual Odometry (Epipolar Version)** `[vo_epipolar.cpp]`

Image $n-1$

Image n

Relative pose R, **t**

Given K   Use_5pt = true

[Tentative Matching]	**[Outlier Rejection]**	**[Relative Pose Estimation]**
- Extract optical flow between adjacent images	- Remove wrong matches using `cv::findEssentialMat()`	- Decompose camera pose using `cv::recoverPose()`

Pass tentative matches   Pass the essential matrix E

[Relative Pose Accumulation]

$$T_n^0 = T_{n-1}^0 \Delta_n^{n-1} = T_{n-1}^0 \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix}^{-1}$$

Pass the relative pose R, **t** (Note: $\| \mathbf{t} \| = 1$)
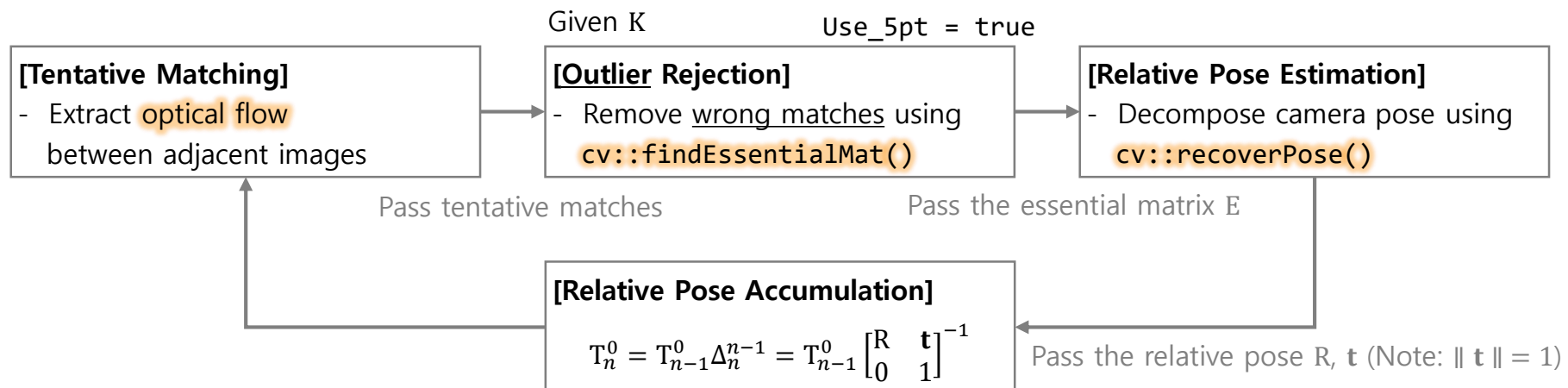
Relative Camera Pose Estimation

- Example) **Monocular Visual Odometry (Epipolar Version)** `[vo_epipolar.cpp]`

(Red: Inlier, Green: Outlier)

Given K   Use_5pt = true

[Tentative Matching]
- Extract optical flow between adjacent images

→ Pass tentative matches →

[Outlier Rejection]
- Remove wrong matches using `cv::findEssentialMat()`

→ Pass the essential matrix E →

[Relative Pose Estimation]
- Decompose camera pose using `cv::recoverPose()`

[Relative Pose Accumulation]

$$T_n^0 = T_{n-1}^0 \Delta_n^{n-1} = T_{n-1}^0 \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix}^{-1}$$

Pass the relative pose R, $\mathbf{t}$ (Note: $\| \mathbf{t} \| = 1$)

Relative Camera Pose Estimation

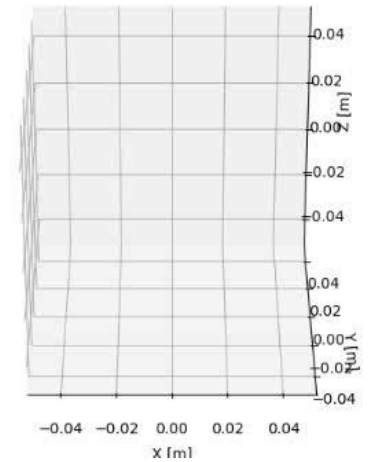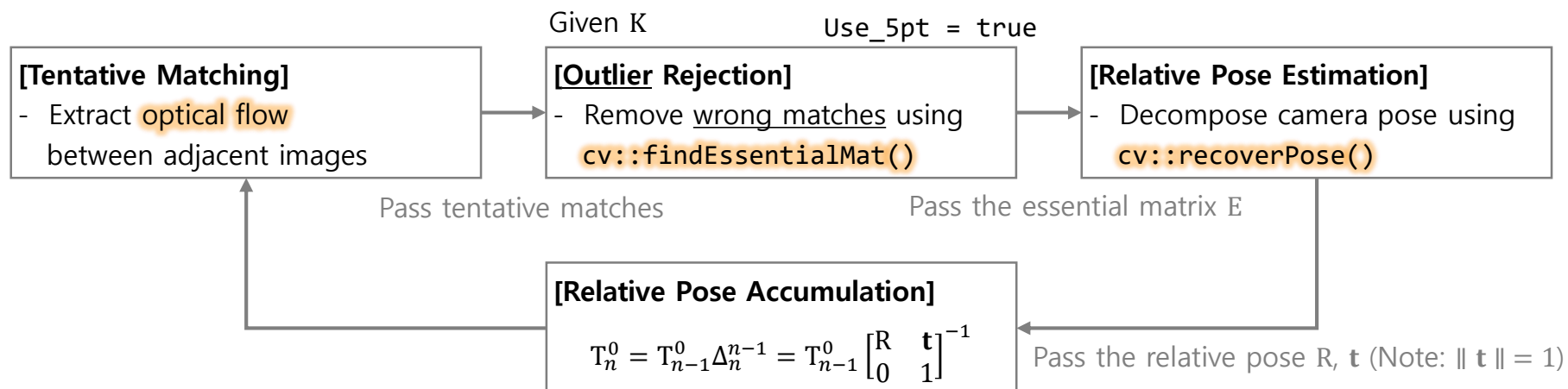- Example) **Monocular Visual Odometry (Epipolar Version)** [vo_epipolar.cpp]

```python
video_file = '../data/KITTI07/image_0/%06d.png'
f, cx, cy = 707.0912, 601.8873, 183.1104
use_5pt = True
min_inlier_num = 100
min_inlier_ratio = 0.2
traj_file = '../data/vo_epipolar.xyz'

# Open a video and get an initial image
video = cv.VideoCapture(video_file)


_, gray_prev = video.read()
if gray_prev.ndim >= 3 and gray_prev.shape[2] > 1:
    gray_prev = cv.cvtColor(gray_prev, cv.COLOR_BGR2GRAY)

# Prepare a plot to visualize the camera trajectory
...


# Run the monocular visual odometry
K = np.array([[f, 0, cx], [0, f, cy], [0, 0, 1]])
camera_traj = np.zeros((1, 3))
camera_pose = np.eye(4)
while True:
    # Grab an image from the video
    valid, img = video.read()
    if not valid:
        break
```

Relative Camera Pose Estimation

- Example) **Monocular Visual Odometry (Epipolar Version)** [vo_epipolar.cpp]

```python
# Run the monocular visual odometry
K = np.array([[f, 0, cx], [0, f, cy], [0, 0, 1]])
camera_traj = np.zeros((1, 3))
camera_pose = np.eye(4)
while True:
    # Grab an image from the video
    valid, img = video.read()
    if not valid:
        break
    if img.ndim >= 3 and img.shape[2] > 1:
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    else:
        gray = img.copy()

    # Extract optical flow
    pts_prev = cv.goodFeaturesToTrack(gray_prev, 2000, 0.01, 10)
    pts, status, error = cv.calcOpticalFlowPyrLK(gray_prev, gray, pts_prev, None)
    gray_prev = gray

    # Calculate relative pose
    if use_5pt:
        E, inlier_mask = cv.findEssentialMat(pts_prev, pts, f, (cx, cy), cv.FM_RANSAC, 0.99, 1)
    else:
        F, inlier_mask = cv.findFundamentalMat(pts_prev, pts, cv.FM_RANSAC, 1, 0.99)
        E = K.T @ F @ K
    inlier_num, R, t, inlier_mask = cv.recoverPose(E, pts_prev, pts, focal=f, pp=(cx, cy), mask=inlier_mask)
```

34

Relative Camera Pose Estimation

- Example) **Monocular Visual Odometry (Epipolar Version)** [vo_epipolar.cpp]

```python
# Run the monocular visual odometry
while True:
    # Grab an image from the video
    ...

    # Extract optical flow
    ...

    # Calculate relative pose
    if use_5pt:
        E, inlier_mask = cv.findEssentialMat(pts_prev, pts, f, (cx, cy), cv.FM_RANSAC, 0.99, 1)
    else:
        F, inlier_mask = cv.findFundamentalMat(pts_prev, pts, cv.FM_RANSAC, 1, 0.99)
        E = K.T @ F @ K
    inlier_num, R, t, inlier_mask = cv.recoverPose(E, pts_prev, pts, focal=f, pp=(cx, cy), mask=inlier_mask)
    inlier_ratio = inlier_num / len(pts)

    # Accumulate relative pose if result is reliable
    info_color = (0, 255, 0)
    if inlier_num > min_inlier_num and inlier_ratio > min_inlier_ratio:
        T = np.eye(4)
        T[:3, :3] = R
        T[:3, 3] = t.flatten()
        camera_pose = camera_pose @ np.linalg.inv(T)
        info_color = (0, 0, 255)
```

$$T_n^0 = T_{n-1}^0 \Delta_n^{n-1} = T_{n-1}^0 \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1}$$

Relative Camera Pose Estimation

- **Relative camera pose estimation** (~ fundamental/essential matrix estimation)

  - Unknown: Rotation and translation R, **t** (**5 DOF**; up-to scale **"scale ambiguity"**)

  - Given: Point correspondence $(\mathbf{x}_1, \mathbf{x}_1')$, ..., $(\mathbf{x}_n, \mathbf{x}_n')$ and camera matrices K, K'

  - Constraints: n x epipolar constraint ($\mathbf{x}'^\mathsf{T} F \mathbf{x} = 0$ or $\hat{\mathbf{x}}'^\mathsf{T} E \hat{\mathbf{x}} = 0$)

  - Solutions) **Fundamental matrix**: 7/8-point algorithm (**7 DOF**; intrinsic+extrinsic)

    - **Properties**: $\det(F) = 0$ (or $\text{rank}(F) = 2$)

    - **Estimation**: `cv.findFundamentalMat()` $\rightarrow$ 1 solution

    - **Conversion to E**: $E = K'^\mathsf{T} F K$

    - **Degenerate cases**: No translation, correspondence from a single plane

  - Solutions) **Essential matrix**: 5-point algorithm (**5 DOF**; extrinsic)

    - **Properties**: $\det(E) = 0$ and $2EE^\mathsf{T}E - \text{tr}(EE^\mathsf{T}) E = 0$

    - **Estimation**: `cv.findEssentialMat()` $\rightarrow$ k solutions

    - **Decomposition**: `cv.decomposeEssentialMat()` $\rightarrow$ 4 solutions **"relative pose ambiguity"**

    - **Decomposition with positive-depth check**: `cv.recoverPose()` $\rightarrow$ 1 solution

    - **Degenerate cases**: No translation ($\because E = [\mathbf{t}]_\times R$)

R, **t**
$\updownarrow$ ($E = [\mathbf{t}]_\times R$)
E
$\updownarrow$ ($E = K'^\mathsf{T} F K$)
F

Relative Camera Pose Estimation

- **Relative camera pose estimation** (~ fundamental/essential matrix estimation + homography estimation)
  - – Solutions) **Fundamental matrix**: 7/8-point algorithm (**7 DOF**; intrinsic+extrinsic)
    - **Estimation**: `cv.findFundamentalMat()` → 1 solution
    - **Conversion to E**: $E = K'^{\top} F K$
    - **Degenerate cases**: No translation, correspondence from a single plane
  - – Solutions) **Essential matrix**: 5-point algorithm (**5 DOF**; extrinsic)
    - **Estimation**: `cv.findEssentialMat()` → k solutions
    - **Decomposition**: `cv.decomposeEssentialMat()` → 4 solutions **"relative pose ambiguity"**
    - **Decomposition with positive-depth check**: `cv.recoverPose()` → 1 solution
    - **Degenerate cases**: No translation ($\because E = [\mathbf{t}]_\times R$)
  - – Solutions) **Planar homography**: 4-point algorithm (**8 DOF**; up-to scale **"scale ambiguity"**)
    - **Estimation**: `cv.findHomography()` → 1 solutions
    - **Conversion to calibrated H**: $\hat{H} = K'^{-1} H K$
    - **Decomposition**: `cv.decomposeHomographyMat()` → 4 solutions **"relative pose ambiguity"**
    - **Degenerate cases**: Correspondence **not** from a single plane

complementary

Relative Camera Pose Estimation

- **Relative camera pose estimation**
  - Solutions) **Planar homography**: 4-point algorithm (**8 DOF**; up-to scale **"scale ambiguity"**)
    - **Estimation**: `cv.findHomography()` → 1 solutions
    - **Derivation**

      $$\lambda' \hat{\mathbf{x}}' = \lambda R \hat{\mathbf{x}} + \mathbf{t}$$

      $$\downarrow \quad \frac{1}{d} \mathbf{n}^\mathsf{T} \hat{\mathbf{x}} = 1 \ (\because n_x \hat{x} + n_y \hat{y} + n_z \hat{z} - d = 0)$$

      $$\hat{\mathbf{x}}' = \lambda'' \left(R + \frac{1}{d'} \mathbf{t}\, \mathbf{n}^\mathsf{T} \right) \hat{\mathbf{x}}$$

      $$\downarrow \quad \boxed{\widehat{H} = R + \frac{1}{d'} \mathbf{t}\, \mathbf{n}^\mathsf{T}}$$

      $$\hat{\mathbf{x}}' = \widehat{H}\, \hat{\mathbf{x}}$$

      $$\downarrow \quad \hat{\mathbf{x}} = K^{-1}\mathbf{x} \ \text{ and } \ \hat{\mathbf{x}}' = K'^{-1}\mathbf{x}'$$

      $$\mathbf{x}' = H\, \mathbf{x}$$

    - **Conversion to calibrated H**: $\boxed{\widehat{H} = K'^{-1} H K}$
    - **Decomposition**: `cv.decomposeHomographyMat()` → 4 solutions **"relative pose ambiguity"**
    - **Degenerate cases**: Correspondence **not** from a single plane

Relative Camera Pose Estimation: Overview

	Items	General 2D-2D Geometry	Planar 2D-2D Geometry
On Image Planes	Model	**Fundamental Matrix** (7 DOF)	**Planar Homography** (8 DOF)
	Formulation	$F = K'^{-\top}EK^{-1}$ $\qquad E = K'^{\top}FK$	$H = K'\widehat{H}K^{-1}$ $\qquad \widehat{H} = K'^{-1}HK$
	Estimation	- **7-point algorithm** ($n \geq 7$) $\rightarrow k$ solution - **(normalized) 8-point algorithm** $\rightarrow$ 1 solution - `cv.findFundamentalMat()`	- **4-point algorithm** ($n \geq 4$) $\rightarrow$ 1 solution - `cv::findHomography()`
	Input	- $(\mathbf{x}_i, \mathbf{x}_i')$ [px] on the image plane	- $(\mathbf{x}_i, \mathbf{x}_i')$ [px] on a plane in the image plane
	Degenerate Cases	- No translational motion - Correspondence from a single plane	- Correspondence <u>not</u> from a single plane
	Decomposition to R and $\mathbf{t}$	- Convert to an essential matrix and decompose it	- `cv.decomposeHomographyMat()`
On Normalized Image Planes	Model	**Essentials Matrix** (5 DOF)	**(Calibrated) Planar Homgraphy** (8 DOF)
	Formulation	$E = [\mathbf{t}]_{\times}R$	$\widehat{H} = R + \dfrac{1}{d}\mathbf{t}\mathbf{n}^{\top}$
	Estimation	- **5-point algorithm** ($n \geq 5$) $\rightarrow k$ solution - `cv.findEssentialMat()`	- **4-point algorithm** ($n \geq 4$) $\rightarrow$ 1 solution - `cv::findHomography()`
	Input	- $(\widehat{\mathbf{x}}_i, \widehat{\mathbf{x}}_i')$ [m] on the normalized image plane	- $(\widehat{\mathbf{x}}_i, \widehat{\mathbf{x}}_i')$ [m] on a plane in the normalized image plane
	Degenerate Cases	- No translational motion	- Correspondence not from a single plane
	Decomposition to R and $\mathbf{t}$	- `cv.decomposeEssentialMat()` - `cv.recoverPose()`	- `cv.decomposeHomographyMat()` with $K = I_{3\times 3}$

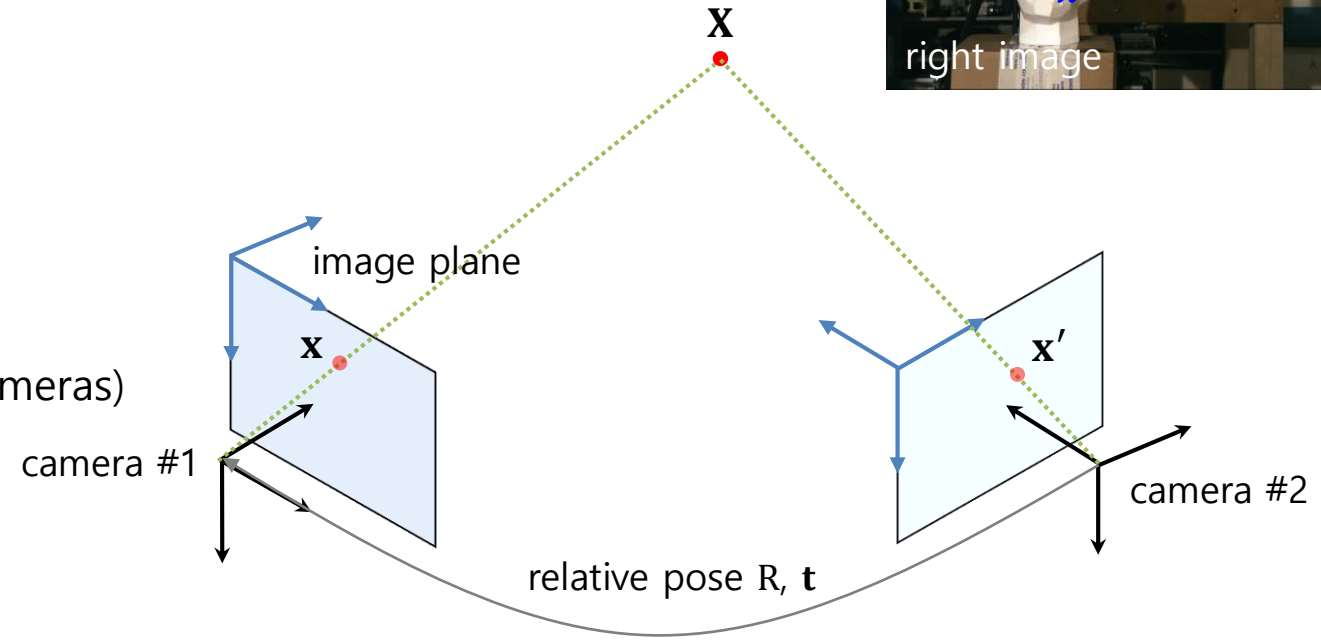# Triangulation

- **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: $\mathbf{x} = K\,[\,I\,|\,\mathbf{0}\,]\,\mathbf{X} = P\,\mathbf{X}$, $\mathbf{x}' = K'\,[\,R\,|\,\mathbf{t}\,]\,\mathbf{X} = P'\,\mathbf{X}$
  - Solution
    - OpenCV `cv.triangulatePoints()`

  - Special case) Stereo cameras

$$R = I_{3\times3}, \ \mathbf{t} = \begin{bmatrix} -b \\ 0 \\ 0 \end{bmatrix}, \text{ and } K = K' = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

  where b is the baseline (distance between two cameras)

$$\therefore Z = \frac{f}{x-x'}b, \ X = \frac{x-c_x}{f}Z, \text{ and } Y = \frac{y-c_y}{f}Z$$

left image
disparity
right image

$\mathbf{X}$
image plane
$\mathbf{x}$
$\mathbf{x}'$
camera #1
camera #2
relative pose R, $\mathbf{t}$

Image: Middlebury Stereo Vision Page

41

Triangulation

- Example) Triangulation

```python
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)
X /= X[3]
X = X.T

# Write the reconstructed 3D points
np.savetxt(output_file, X)
```
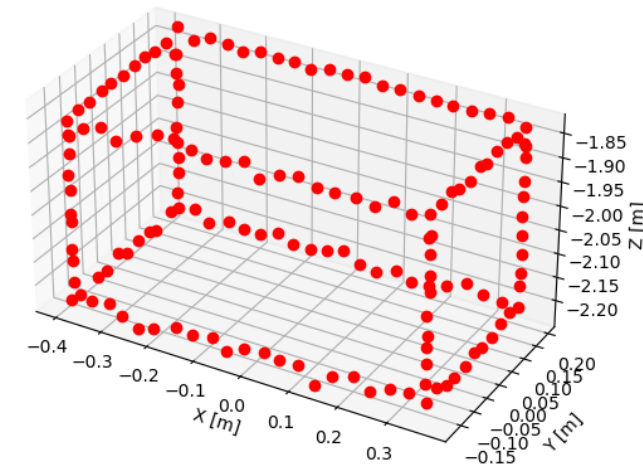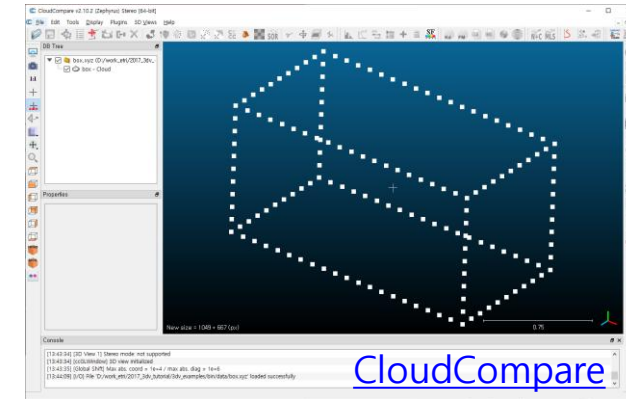
$$x = K\,[\,I\,|\,0\,]\,X = P\,X$$

$$x' = K'\,[\,R\,|\,t\,]\,X = P'\,X$$

A point cloud: data/box.xyz

CloudCompare

output_file: data/triangulation.xyz

Summary) Two-view Geometry

- **Planar Homography**: $\mathbf{x}'_i = \mathrm{H}\mathbf{x}_i$

  - Example) Perspective distortion correction

  - Example) Planar image stitching

  - Example) 2D video stabilization

- **Epipolar Geometry**: $\mathbf{x}'^\top \mathrm{F}\mathbf{x} = 0$ (on the image plane), $\hat{\mathbf{x}}'^\top \mathrm{E}\hat{\mathbf{x}} = 0$ (on the *normalized* image plane)

  - Example) Epipolar line visualization

- **Relative Camera Pose Estimation**: Finding R and $\mathbf{t}$ (5 DOF)

  - Solutions) **Fundamental matrix**: 7/8-point algorithm (7 DOF)
  - Solutions) **Essential matrix**: 5-point algorithm (5 DOF)
  - Solutions) **Planar homography**: 4-point algorithm (8 DOF)

  - Example) Fundamental matrix estimation

  - Example) Monocular Visual Odometry (Epipolar Version)

- **Triangulation**: Finding $\mathbf{X}$ (3 DOF)

  - Example) Triangulation

$$
\begin{array}{l}
\mathrm{R},\ \mathbf{t} \\
\updownarrow \quad (\mathrm{E} = [\mathbf{t}]_\times \mathrm{R}) \\
\mathrm{E} \\
\updownarrow \quad (\mathrm{E} = \mathrm{K}'^\top \mathrm{F}\mathrm{K}) \\
\mathrm{F}
\end{array}
$$