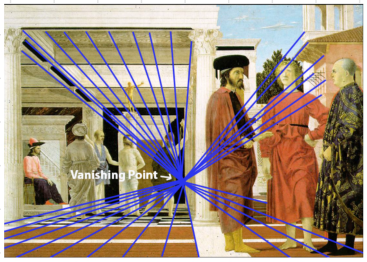
Assignment4 notes

Lec 14. perspective

① motivation: recover structure from { multi-view geometry ~ in general
single will ~ with certain assumption

② properties of projection:



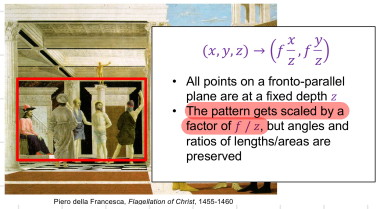
Piero della Francesca, *Flagellation of Christ*, 1455-1460

~ lines: parallel lines meets on vanishing point
(not all, just some direction)

~ planes: ① patterns on non-fronto-parallel planes
are distorted by a 2D homography.
and have vanishing lines


Vanishing line of the ground plane
Piero della Francesca, *Flagellation of Christ*, 1455-1460

② pattern on fronto-parallel plane: →



Piero della Francesca, *Flagellation of Christ*, 1455-1460

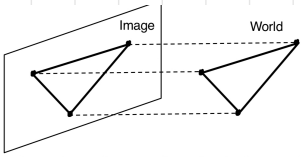
Tip: horizon tells us relative height of scene points and camera.

③ Perspective projection matrix

projection is a matrix multiplication using homogeneous coordinates:

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} \Rightarrow \left(f\frac{x}{z}, f\frac{y}{z}\right)$$

— for Orthographic projection:



- Assuming projection along the z axis, what's the matrix?

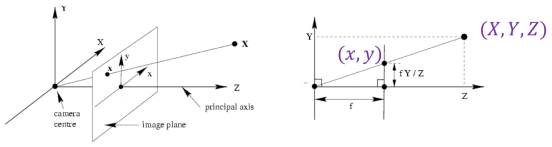
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Lec 15. calibration and triangulation

↳ first taste for 3D reconstruction (Lo6)

① camera calibration → figure out the transformation from world coordinate to image's.

assume in a normalized coordinates (camera center is on the origin, principle axis is the z-axis. x, y are parallel for image and world coordinate)



$$x = f\frac{X}{Z}, y = f\frac{Y}{Z}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cong \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$x \cong PX$ (Equality up to scale)

- x: Homogeneous coord. vec. of image point
- P: Camera projection matrix
- X: Homogeneous coord. vec. of 3D point

To be specific:

$$\begin{pmatrix} 2D\ point \\ 3\times1 \\ x \end{pmatrix} = \begin{pmatrix} Camera\ to\ pixel\ coordinate\ trans.\ matrix \\ K\ (3\times3) \end{pmatrix} \cdot \begin{pmatrix} canonical\ projection\ matrix \\ [I|0]\ (3\times4) \end{pmatrix} \cdot \begin{pmatrix} world\ to\ camera\ coord.\ trans.\ matrix \\ \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}(4\times4) \end{pmatrix} \cdot \begin{pmatrix} 3D\ point \\ 4\times1 \\ X \end{pmatrix}$$

(intrinsic matrix (principal points, scaling ...))

↑ extrinsic matrix: rotation, translation

$x = P \cdot X$, $P = k[R|t]$

1. intrinsic matrix:

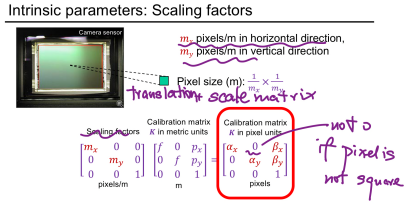
$$x = f\frac{X}{Z} + P_x,\ y = f\cdot\frac{Y}{Z} + P_y$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cong \begin{pmatrix} f\cdot X + Z\cdot P_x \\ f\ Y + Z\cdot P_y \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & P_x & 0 \\ 0 & f & P_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$P = k[I|0] \longrightarrow \begin{bmatrix} f & 0 & P_x \\ 0 & f & P_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

intrinsic matrix (calibration matrix k) · canonical projection matrix $[I|0]$

advanced



2. extrinsic matrix:

$$\tilde{x}_{cam} = R\cdot(\tilde{X} - \tilde{C}) = R\cdot\tilde{X} + t$$

(R: (3×3); $\tilde{X}$: coord. of a point in world frame; $\tilde{C}$: coord. of camera center in world frame)

In non-homo coordinates: $\tilde{X}_{cam} = R\cdot\tilde{X}_{world} + t$

In homo coordinates:

$$\begin{bmatrix} \tilde{x}_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \cdot \begin{pmatrix} \tilde{x}_{world} \\ 1 \end{pmatrix}$$

(4×1) (4×4) (4×1)

∴ Projection of camera center $= R\cdot(C - C) = 0$ ∴ camera center is the null space of projection matrix

so, first need to find projection matrix (P2-c)

```python
def camera_matrix(projection_matrix):
    # matrix A's null space means v that make AV=0
    camera_center = np.zeros(shape = (3,1))
    U, S, V = np.linalg.svd(projection_matrix)
    homography_camera_center = V[len(V) - 1].reshape(1,4)
    for i in range(3):
        camera_center[i] = homography_camera_center[0][i] / homography_camera_center[0][3]
    return camera_center.transpose()

if __name__ == '__main__':
    lab_3d,lab1_2d,lab2_2d = load_data()
    lab_target = lab1_2d
    actual_projection_matrix = find_camera_projection_matrix(lab_target, lab_3d,iterations = 100)
    points_3d_proj, residual = evaluate_points(actual_projection_matrix, lab_target, lab_3d)
    visualize_3dprojects(lab_target,points_3d_proj)
    print("residual:",residual)
    print("projection_matrix:",actual_projection_matrix)
```

2. estimation :

$$\because x \cong k[R|t] \cdot X = P \cdot X$$

$$\therefore \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cong \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

given n points with known 3D X_i and image projection x_i, estimate camera parameter.

- **The method:** $\because x \cong P \cdot X \quad \therefore x \times PX = 0 \quad \therefore x_i \times PX_i = 0$

Camera calibration: Linear method

- Final linear system:

$$\begin{bmatrix} 0^T & X_1^T & -y_1 X_1^T \\ X_1^T & 0^T & -x_1 X_1^T \\ \dots & \dots & \dots \\ 0^T & X_n^T & -y_n X_n^T \\ X_n^T & 0^T & -x_n X_n^T \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0 \qquad Ap = 0$$

$$\begin{bmatrix} 0^T & -X_i^T & y_i X_i^T \\ X_i^T & 0^T & -x_i X_i^T \\ -y_i X_i^T & x_i X_i^T & 0^T \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0$$

3x12 ↑ 3x4

One match gives two linearly independent constraints

- One 2D/3D correspondence gives two linearly independent equations
  - The projection matrix has 11 degrees of freedom
  - 6 correspondences needed for a minimal solution
- Homogeneous least squares: find p minimizing $\|Ap\|^2$
  - Solution is eigenvector of A^TA corresponding to smallest eigenvalue

∴ for coding:

```
def Projection_matrix(data_2d,data_3d):
    sample_index = random.sample(range(0, 20), 6)
    a = np.zeros(shape=(12,12))
    for i in range(6):
        a[2 * i][0] = 0
        a[2 * i][1] = 0
        a[2 * i][2] = 0
        a[2 * i][3] = 0
        a[2 * i][4] = -1 * data_3d[sample_index[i]][0]
        a[2 * i][5] = -1 * data_3d[sample_index[i]][1]
        a[2 * i][6] = -1 * data_3d[sample_index[i]][2]
        a[2 * i][7] = -1
        a[2 * i][8] = data_2d[sample_index[i]][1] * data_3d[sample_index[i]][0]
        a[2 * i][9] = data_2d[sample_index[i]][1] * data_3d[sample_index[i]][1]
        a[2 * i][10] = data_2d[sample_index[i]][1] * data_3d[sample_index[i]][2]
        a[2 * i][11] = data_2d[sample_index[i]][1]
        a[2 * i + 1][0] = data_3d[sample_index[i]][0]
        a[2 * i + 1][1] = data_3d[sample_index[i]][1]
        a[2 * i + 1][2] = data_3d[sample_index[i]][2]
        a[2 * i + 1][3] = 1
        a[2 * i + 1][4] = 0
        a[2 * i + 1][5] = 0
        a[2 * i + 1][6] = 0
        a[2 * i + 1][7] = 0
        a[2 * i + 1][8] = -1 *data_2d[sample_index[i]][0] * data_3d[sample_index[i]][0]
        a[2 * i + 1][9] = -1 *data_2d[sample_index[i]][0] * data_3d[sample_index[i]][1]
        a[2 * i + 1][10] = -1 *data_2d[sample_index[i]][0] * data_3d[sample_index[i]][2]
        a[2 * i + 1][11] = -1 *data_2d[sample_index[i]][0]
    U, S, V = np.linalg.svd(a)
    projection_matrix = V[len(V) - 1].reshape(3,4)
    projection_matrix = projection_matrix/projection_matrix[2][3]
    return projection_matrix
```

compare: given x, X, figure out P ↑

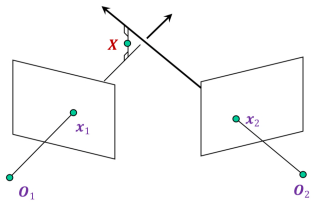
given x, P, figure out X ↓

② Triangulation:

given projection of a 3-D point on 2/more images, with known projection matrix, find the coordinate of this point in 3-D space.

1) geometry approach

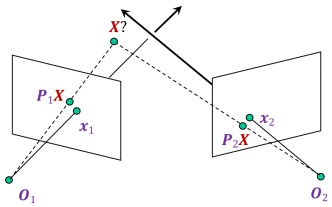
- Find shortest segment connecting the two viewing rays and let X be the midpoint of that segment



2) Nonlinear approach

- Find X that minimizes

$$\|\text{proj}(P_1 X) - x_1\|_2^2 + \|\text{proj}(P_2 X) - x_2\|_2^2$$



↳ like structure from motion

codes: C (原理还没找到, 网上找的)

```
def find_3d_point(x1, y1, x2, y2, actual_projection_matrix1, actual_projection_matrix2):
    a = np.zeros(shape=(4,4))
    a[0][0] = y1 * actual_projection_matrix1[2][0] - actual_projection_matrix1[1][0]
    a[0][1] = y1 * actual_projection_matrix1[2][1] - actual_projection_matrix1[1][1]
    a[0][2] = y1 * actual_projection_matrix1[2][2] - actual_projection_matrix1[1][2]
    a[0][3] = y1 * actual_projection_matrix1[2][3] - actual_projection_matrix1[1][3]
    a[1][0] = actual_projection_matrix1[0][0] - x1 * actual_projection_matrix1[2][0]
    a[1][1] = actual_projection_matrix1[0][1] - x1 * actual_projection_matrix1[2][1]
    a[1][2] = actual_projection_matrix1[0][2] - x1 * actual_projection_matrix1[2][2]
    a[1][3] = actual_projection_matrix1[0][3] - x1 * actual_projection_matrix1[2][3]
    a[2][0] = y2 * actual_projection_matrix2[2][0] - actual_projection_matrix2[1][0]
    a[2][1] = y2 * actual_projection_matrix2[2][1] - actual_projection_matrix2[1][1]
    a[2][2] = y2 * actual_projection_matrix2[2][2] - actual_projection_matrix2[1][2]
    a[2][3] = y2 * actual_projection_matrix2[2][3] - actual_projection_matrix2[1][3]
    a[3][0] = actual_projection_matrix2[0][0] - x2 * actual_projection_matrix2[2][0]
    a[3][1] = actual_projection_matrix2[0][1] - x2 * actual_projection_matrix2[2][1]
    a[3][2] = actual_projection_matrix2[0][2] - x2 * actual_projection_matrix2[2][2]
    a[3][3] = actual_projection_matrix2[0][3] - x2 * actual_projection_matrix2[2][3]
    U, S, V = np.linalg.svd(a)
    point3d_homo = V[len(V) - 1].reshape(4, 1)
    x = point3d_homo[0] / point3d_homo[3]
    y = point3d_homo[1] / point3d_homo[3]
    z = point3d_homo[2] / point3d_homo[3]
    return x,y,z
```

Lec16. single view : from a single image build the 3-D structure

① camera calibration using vanish points

- Consider a scene with three orthogonal vanishing directions:

- Note: v_1, v_2 are *finite* vanishing points and v_3 is an *infinite* vanishing point

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = p_3$$

p_1 p_2 p_3 p_4

Vanishing points in x, y, z directions (i.e., v_1, v_2, v_3)

$$v_i \cong K[R|t] \begin{pmatrix} e_i \\ 0 \end{pmatrix}$$

- Let us align the world coordinate system with three orthogonal vanishing directions in the scene:

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$v_i \cong KRe_i$$

$$e_i \cong R^T K^{-1} v_i$$

- Orthogonality constraint: $e_i^T e_j = 0$

$$v_i^T K^{-T} K^{-1} v_j = 0$$

$e_i \cong R^T \cdot K^{-1} \cdot v_i$

$\therefore e_i^T \cdot e_j =$

- Extrinsic parameter matrix (R) disappears and we are left with constraints on the calibration matrix!

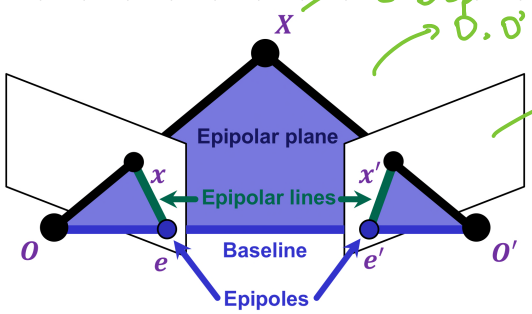
$v_i^T \cdot (K^{-1})^T \cdot R \cdot R^T \cdot K^{-1} \cdot v_j = 0$

$\therefore v_i^T \cdot (K^{-1})^T \cdot K^{-1} \cdot v_j = 0$ $\quad K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$

codes:

```
def get_camera_parameters(vpts):
    """
    Computes the camera parameters. Hint: The SymPy package is suitable for this.
    """
    vp1 = vpts[:, 0].reshape(3,1)
    vp2 = vpts[:, 1].reshape(3,1)
    vp3 = vpts[:, 2].reshape(3,1)

    f, px, py = symbols('f, px, py')
    K_i_T = Matrix([[1/f, 0, 0], [0, 1/f, 0], [-px/f, -py/f, 1]])
    K_i = Matrix([[1/f, 0, -px/f], [0, 1/f, -py/f], [0, 0, 1]])

    eq1 = Eq((vp1.T @ K_i_T @ K_i @ vp2)[0], 0)
    eq2 = Eq((vp1.T @ K_i_T @ K_i @ vp3)[0], 0)
    eq3 = Eq((vp2.T @ K_i_T @ K_i @ vp3)[0], 0)

    f, u, v = solve((eq1,eq2,eq3), (f,px,py))[0]

    return f, u, v
```

Lec 17 epipolar (对极几何)



3-D space point (4×1)

O, O' is camera center

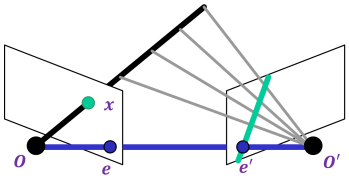
camera plate

① motivation: Given a 2D point in one view, where can we find the corresponding point in the other view?

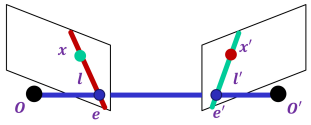
without explicit 3D reasoning

Given only 2D correspondences, how can we calibrate the two cameras, i.e., estimate their relative position and orientation and the intrinsic parameters?

② epipolar constraint



- Where can we find the x' corresponding to x in the other image?
- Along the **epipolar line** corresponding to x (projection of visual ray connecting O with x into the second image plane)



- Potential matches for x have to lie on the matching epipolar line l'
- Potential matches for x' have to lie on the matching epipolar line l

③ Essential matrix :

$$x'^T E \cdot x = 0 \longrightarrow (x', y', 1) \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$$

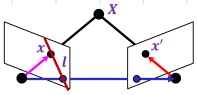
↳ essential matrix

Assume the intrisic and extrinsic matrix of camera are given.

world coordinate is set same to the first camera:

1) First: $k[I|0]$
2) second: $k'[R|t]$

$$\therefore \begin{cases} x_{norm} \cong k \cdot [I|0] X \\ x'_{norm} \cong k' \cdot [R|t] \cdot X \end{cases}$$



$x'^T E x = 0$

- Ex is the epipolar line associated with x ($l' = Ex$)
- $E^T x'$ is the epipolar line associated with x' ($l = E^T x'$)
- $Ee = 0$ and $E^T e' = 0$
- E is singular (rank two) and has five degrees of freedom

④ Fundamental matrix: calibration matrix

compare ↑ $x'^T_{norm} \cdot E \cdot x_{norm} = 0$

where $x_{norm} = k^{-1} x$, $x'_{norm} = k'^{-1} \cdot x'$

k, k' are unknown.

$$x'^T \cdot F \cdot x^T = 0 \rightarrow F = k'^{-T} E \cdot k^{-1}$$

- Fx is the epipolar line associated with x ($l' = Fx$)
- $F^T x'$ is the epipolar line associated with x' ($l = F^T x'$)
- $Fe = 0$ and $F^T e' = 0$
- F is singular (rank two) and has seven degrees of freedom

⑤ estimating the Fundamental matrix

$$(x', y', 1) \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0 \Rightarrow (x'x, x'y, x', y'x, y'y, y', x, y, 1) \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = 0$$

We know F needs to be singular/rank 2. How do we force it to be singular?

Solution: take SVD of the initial estimate and throw out the smallest singular value

$F_{init} = U \Sigma V^T$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \times \end{bmatrix} \rightarrow \Sigma' = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$F = U \Sigma' V^T$

codes for that:

rank-2 F estimate

```
def fundamental_matrix(matches):
    N = len(matches)
    sample_index = random.sample(range(0, N), 8)
    A = np.zeros(shape=(8,9))
    for i in range(8):
        A[i][0] = matches[sample_index[i]][0] * matches[sample_index[i]][2]
        A[i][1] = matches[sample_index[i]][1] * matches[sample_index[i]][2]
        A[i][2] = matches[sample_index[i]][2]
        A[i][3] = matches[sample_index[i]][0] * matches[sample_index[i]][3]
        A[i][4] = matches[sample_index[i]][1] * matches[sample_index[i]][3]
        A[i][5] = matches[sample_index[i]][3]
        A[i][6] = matches[sample_index[i]][0]
        A[i][7] = matches[sample_index[i]][1]
        A[i][8] = 1
    U, S, V = la.svd(A)
    F_unnormalized = V[len(V) - 1].reshape(3, 3)
    _U, _sigma, _V = la.svd(F_unnormalized)
    _sigma[2] = 0
    F_normalized = _U @ (np.diag(_sigma)) @ _V
    return F_unnormalized,F_normalized
```