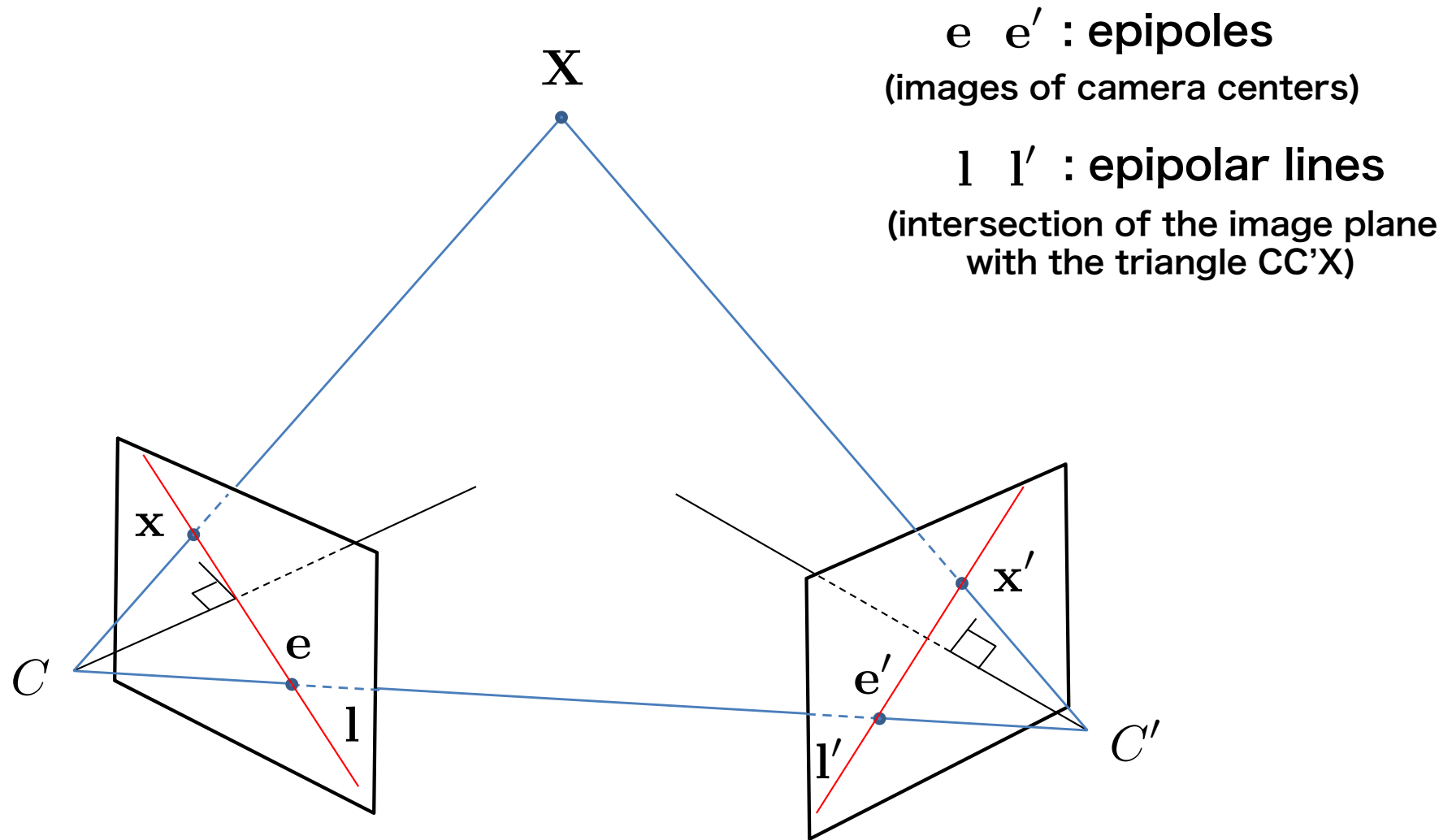


コンピュータビジョン

多視点幾何

Epipolar geometry

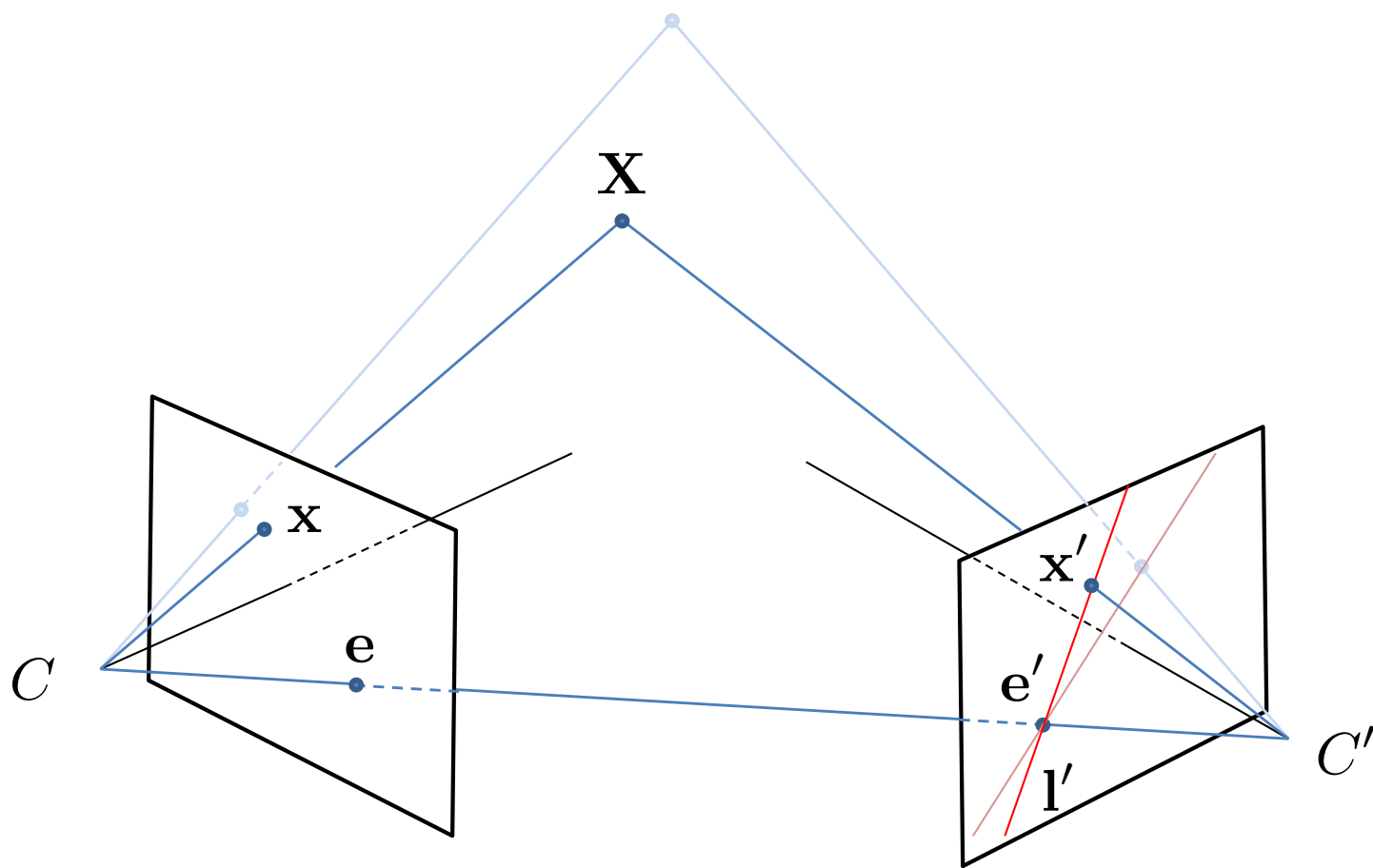
(Two-view geometry)



Epipolar geometry

(Two-view geometry)

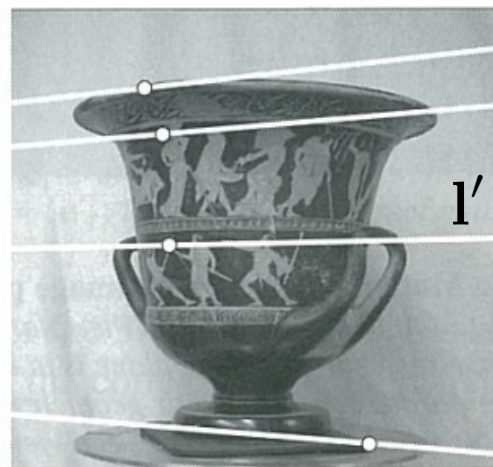
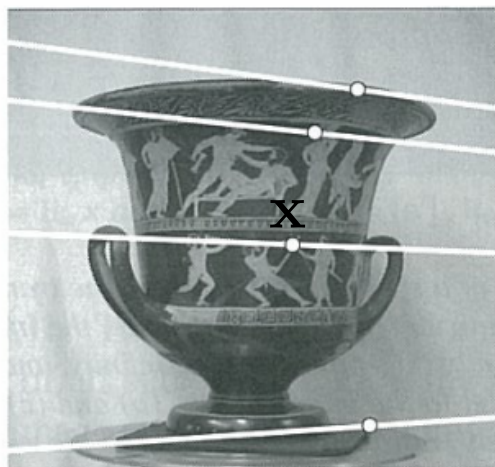
- $\mathbf{x}$ を決めると $\mathbf{l}'$ が決まる



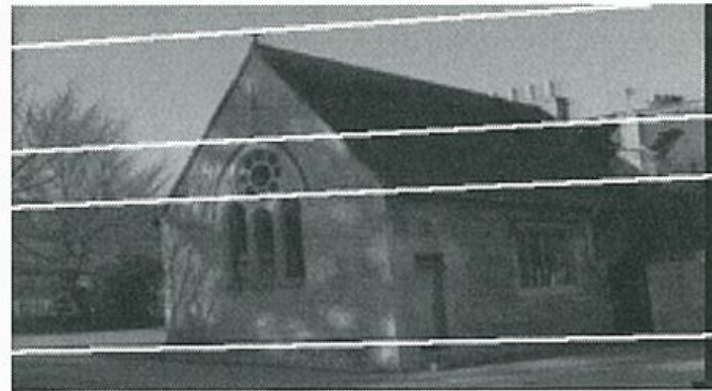
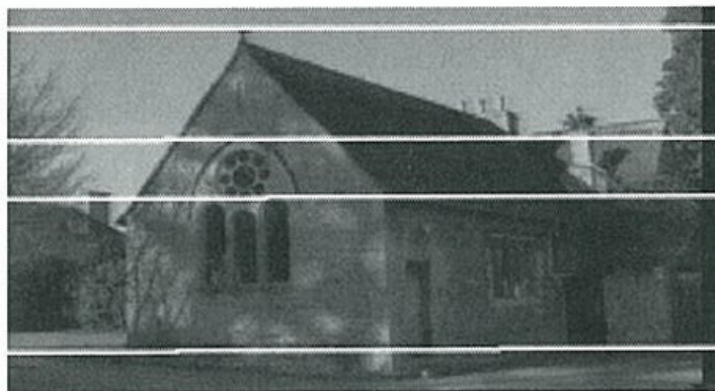
Epipolar geometry

(Two-view geometry)

- x を決めると l' が決まる



- epipoleが無限遠にあるとき



[Hartley-Zisserman03]

Essential matrix

$$\mathbf{E} = [\mathbf{t}]_{\times} \mathbf{R}$$

$\mathbf{E}$ は自由度5

$$R(3)+t(3)-\text{scale}(1)=5$$

スケール倍には意味がない

座標変換: $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t}$

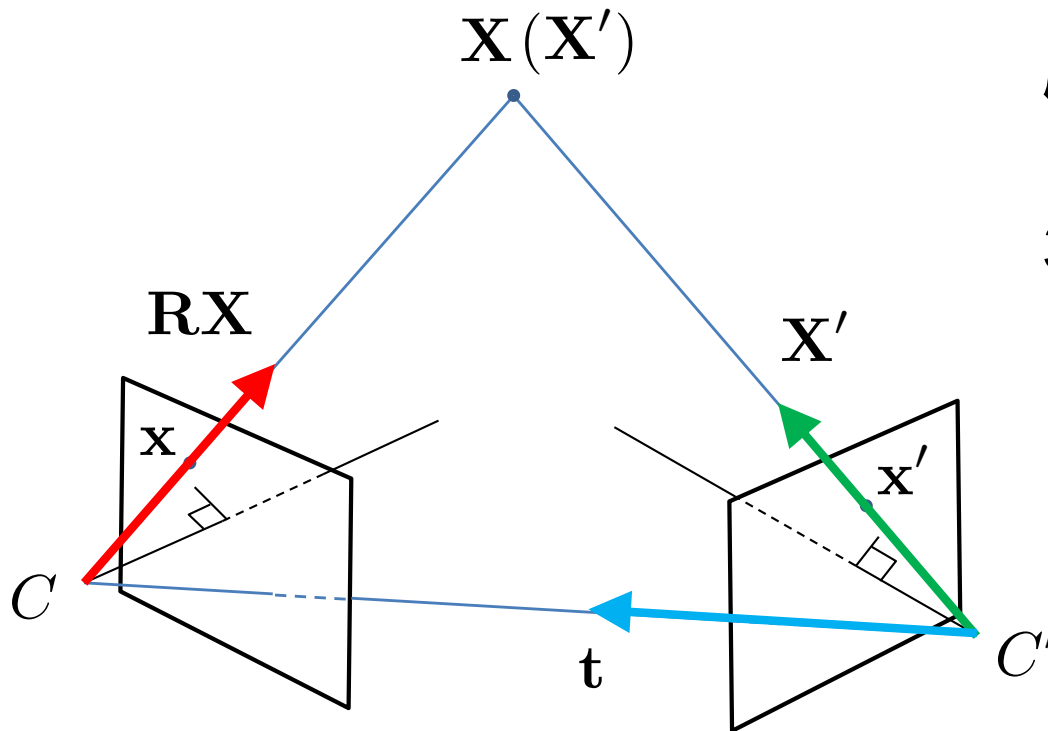
3つのベクトルが同一平面上

$$\rightarrow \mathbf{X}'^{\top} (\mathbf{t} \times \mathbf{R}\mathbf{X}) = 0$$

$$\rightarrow \mathbf{X}'^{\top} ([\mathbf{t}]_{\times} \mathbf{R}\mathbf{X}) = 0$$

$$\rightarrow \mathbf{X}'^{\top} ([\mathbf{t}]_{\times} \mathbf{R}) \mathbf{X} = 0$$

$$\rightarrow \mathbf{X}'^{\top} \mathbf{E} \mathbf{X} = 0$$



カメラモデル (revisited)

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

Image coordinates

$$\mathbf{x} \propto \mathbf{P}\mathbf{X}$$

$$\underbrace{\mathbf{K} [\mathbf{R} \mid \mathbf{t}]}$$



Internal parameters
(focal length etc.)

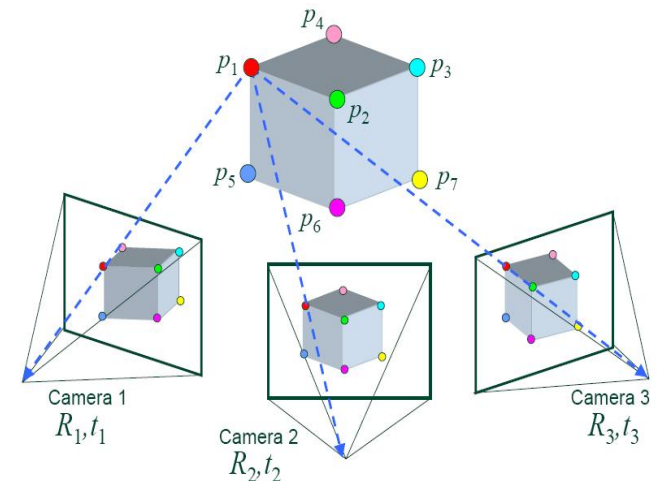
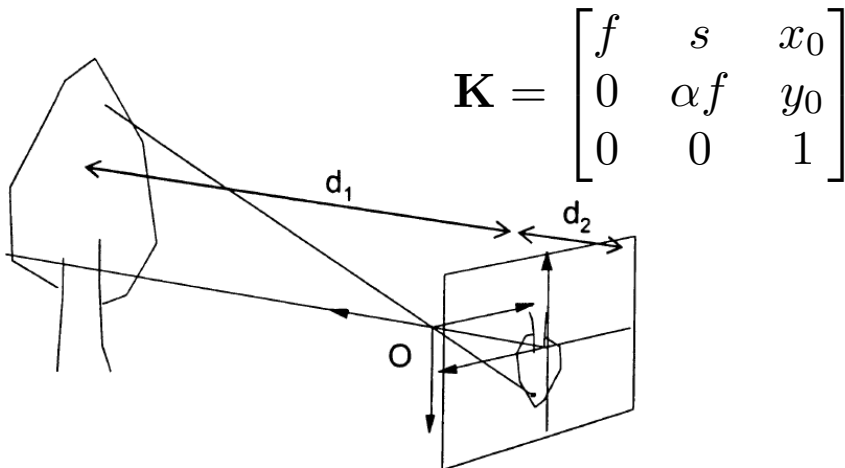
“What camera?”

External parameters
(= camera poses)

“From where?”

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

3D coordinates



Fundamental matrix

$$\mathbf{F} = \mathbf{K}'^{-\top} \mathbf{E} \mathbf{K}^{-1}$$

essential mat.の「画像座標版」

$\mathbf{F}$ は自由度7

3x3 – scale(1) – “det=0”(1) = 7

スケール倍には意味がない

$\mathbf{X}'^{\top} \mathbf{E} \mathbf{X} = 0$ に以下を代入

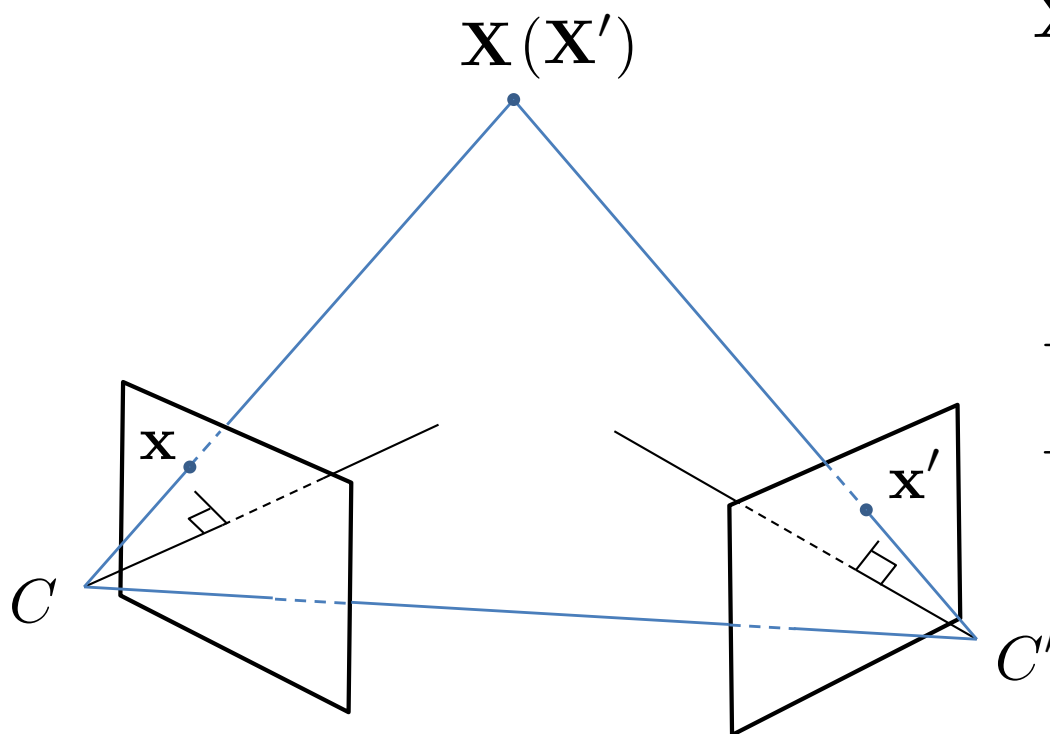
$$\mathbf{x} \propto \mathbf{K} \mathbf{X} \rightarrow \mathbf{X} \propto \mathbf{K}^{-1} \mathbf{x}$$

$$\mathbf{x}' \propto \mathbf{K}' \mathbf{X}' \rightarrow \mathbf{X}' \propto \mathbf{K}'^{-1} \mathbf{x}'$$

$$\rightarrow (\mathbf{K}'^{-1} \mathbf{x}')^{\top} \mathbf{E} (\mathbf{K}^{-1} \mathbf{x}) = 0$$

$$\rightarrow \mathbf{x}'^{\top} (\mathbf{K}'^{-\top} \mathbf{E} \mathbf{K}^{-1}) \mathbf{x} = 0$$

$$\rightarrow \mathbf{x}'^{\top} \mathbf{F} \mathbf{x} = 0$$

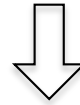


Fundamental matrixの推定

8-point algorithm

$$\mathbf{x}'^\top \mathbf{F} \mathbf{x} = 0$$

$$x'x f_{00} + x'y f_{01} + x' f_{02} + y'x f_{10} + y'y f_{11} + y' f_{12} + x f_{20} + y f_{21} + f_{22} = 0$$



9

$$n \begin{bmatrix} x'_0 x_0 & x'_0 y_0 & x'_0 & y'_0 x_0 & y'_0 y_0 & y'_0 & x_0 & y_0 & 1 \\ x'_1 x_1 & x'_1 y_1 & x'_1 & y'_1 x_1 & y'_1 y_1 & y'_1 & x_1 & y_1 & 1 \\ & & & & & & \vdots & & \\ x'_{n-1} x_{n-1} & x'_{n-1} y_{n-1} & x'_{n-1} & y'_{n-1} x_{n-1} & y'_{n-1} y_{n-1} & y'_{n-1} & x_{n-1} & y_{n-1} & 1 \end{bmatrix} \begin{bmatrix} f_{00} \\ f_{01} \\ f_{02} \\ f_{10} \\ f_{11} \\ f_{12} \\ f_{20} \\ f_{21} \\ f_{22} \end{bmatrix} = \mathbf{0}$$

$$\mathbf{A} \mathbf{f} = \mathbf{0}$$

Fundamental matrixの推定

fundM.py

```
if desc0 != None:
    # Match the saved and the latest kp's
    matches = bf.match(desc0, desc)
    src_pts = numpy.float32([ kp0[m.queryIdx].pt ¥
                              for m in matches ]).reshape(-1,1,2)
    dst_pts = numpy.float32([ kp[m.trainIdx].pt ¥
                              for m in matches ]).reshape(-1,1,2)
    F, mask = cv2.findFundamentalMat(src_pts, dst_pts,
                                     ¥cv2.FM_RANSAC)

    ...
key = cv2.waitKey(10)

if key == 0x1b: # ESC
    cv2.imwrite('img1.png', image)
    numpy.savetxt('fundM.txt', F)
    break
elif key == 0x20:
    desc0 = desc.copy()
    kp0 = kp[:]
    cv2.imwrite('img0.png', image)
    print ('Target updated!')
```

Space → 第1視点・ESC → 第2視点, fundM.txt にセーブ

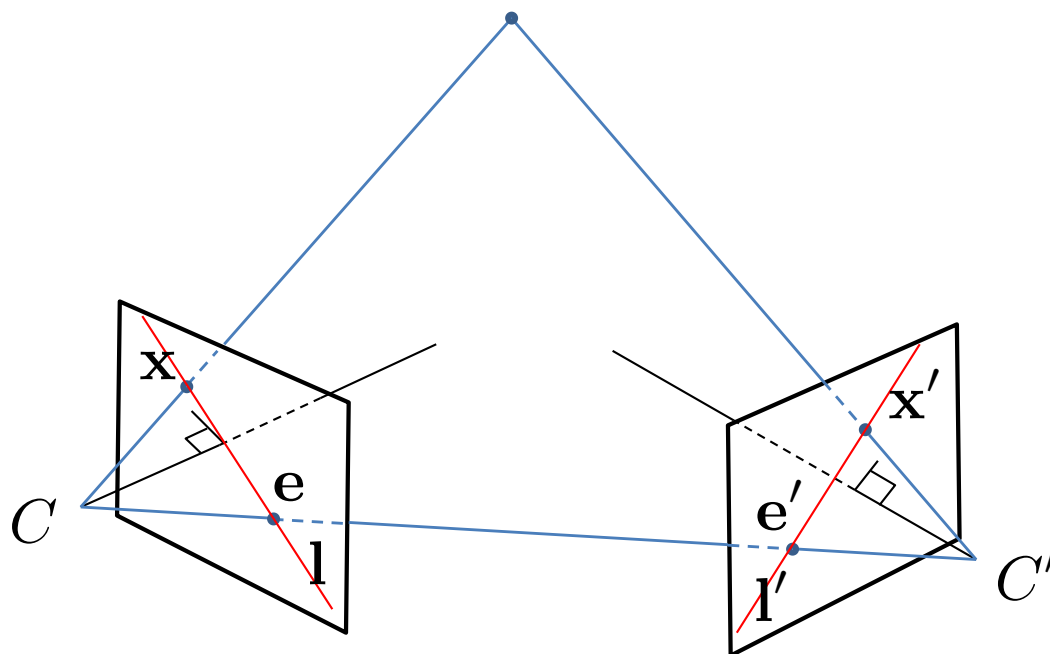
Fundamental matrix

$\mathbf{x}$ に対する epipolar line $\mathbf{l}'$ は $\mathbf{l}' \propto \mathbf{F}\mathbf{x}$ ($\mathbf{x}'^\top \mathbf{l}' = \mathbf{x}'^\top (\mathbf{F}\mathbf{x}) = 0$)

($\mathbf{x}'$ に対する epipolar line $\mathbf{l}$ は $\mathbf{l} \propto \mathbf{F}^\top \mathbf{x}'$ ($\mathbf{x}^\top \mathbf{l} = \mathbf{x}^\top (\mathbf{F}^\top \mathbf{x}') = 0$))

$\mathbf{e}$ に対する epipolar line $\mathbf{l}'$ は定まらない: $\mathbf{F}\mathbf{e} = \mathbf{F}^\top \mathbf{e} = 0$

どの epipolar line $\mathbf{l}$ も $\mathbf{e}$ を通る: $\mathbf{l}^\top \mathbf{e} = \mathbf{x}'^\top \mathbf{F}\mathbf{e} = \mathbf{x}^\top (\mathbf{F}\mathbf{e}) = 0$



直線の式: $\mathbf{l}^\top \mathbf{x} = 0$

$$\mathbf{l} = [a, b, c]^\top$$

$$\mathbf{x} = [x, y, 1]^\top$$

$$ax + by + c = 0$$

Epipolar line

epi_line.py

```
def on_mouse(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE:
        l = F.dot(numpy.array((x,y,1)))
        w = img1.shape[1]
        img1c = img1.copy()
        cv2.line(img1c, (0,int(-l[2]/l[1])),
                (w,int((-l[2]-l[0]*w)/l[1])), (0,0,255))
        cv2.imshow('Image 1', img1c)

cv2.namedWindow('Image 0')
cv2.setMouseCallback('Image 0', on_mouse)

img0 = cv2.imread('img0.png')
img1 = cv2.imread('img1.png')
img1c = img1.copy()
F = numpy.loadtxt('fundM.txt')
cv2.imshow('Image 0', img0)
cv2.imshow('Image 1', img1c)

while 1:
    key = cv2.waitKey(10)

    if key == 0x1b: # ESC
        break
```

Essential matrixの分解

- E を分解して R と t を取り出せる

E の特異値分解は必ず次の形に：

$$\mathbf{E} = \mathbf{U} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{V}^\top$$

次の4通りの (R, t) がある：

$$(\mathbf{R}, \mathbf{t}) = (\mathbf{U}\mathbf{W}\mathbf{V}^\top, \mathbf{u}_3)$$

$$(\mathbf{R}, \mathbf{t}) = (\mathbf{U}\mathbf{W}\mathbf{V}^\top, -\mathbf{u}_3)$$

$$(\mathbf{R}, \mathbf{t}) = (\mathbf{U}\mathbf{W}^\top \mathbf{V}^\top, \mathbf{u}_3)$$

$$(\mathbf{R}, \mathbf{t}) = (\mathbf{U}\mathbf{W}^\top \mathbf{V}^\top, -\mathbf{u}_3)$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Essential matrixの分解

decompE.py

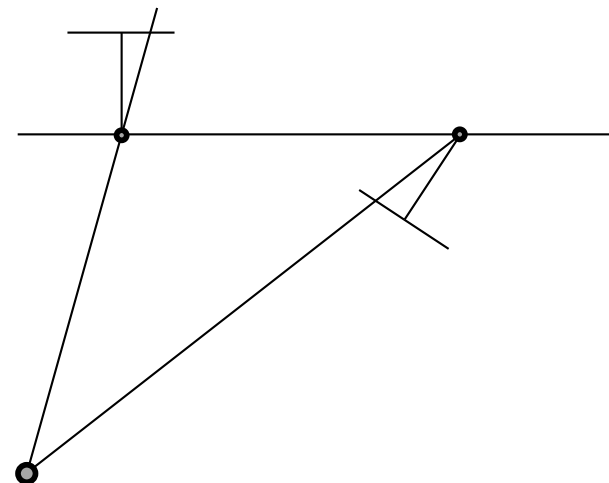
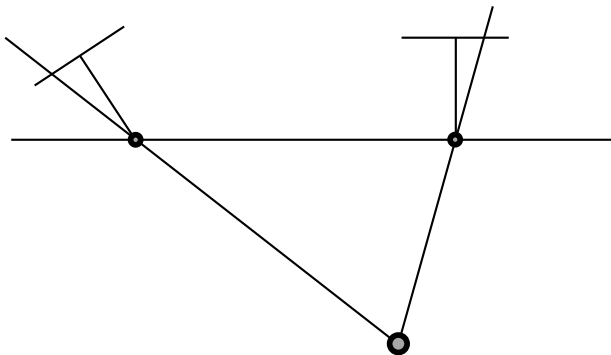
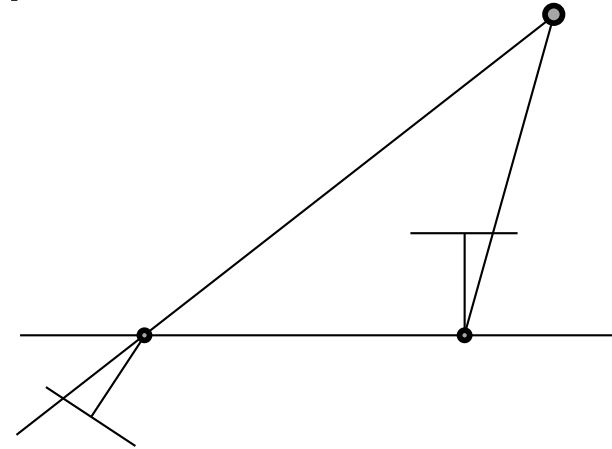
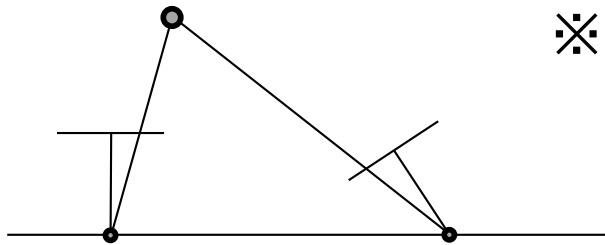
```
import numpy, cv2

def decompose_E(E):
    U, s, Vt = numpy.linalg.svd(E)
    print 'singular value = ', s
    #print U.dot(numpy.diag(s)).dot(Vt), E
    W = numpy.array([[0,-1,0],[1,0,0],[0,0,1]])
    R0=U.dot(W).dot(Vt) # UWVt
    R1=U.dot(W.transpose()).dot(Vt) # UWVt
    t=U[:,2] # u3
    Rt_list=((R0,t),(R0,-t),(R1,t),(R1,-t))
    return Rt_list

if __name__=="__main__":
    F = numpy.loadtxt('fundM.txt')
    K = numpy.loadtxt('K.txt')
    E = (K.transpose().dot(F)).dot(K)
    Rt_list = decompose_E(E)
    print (Rt_list)
```

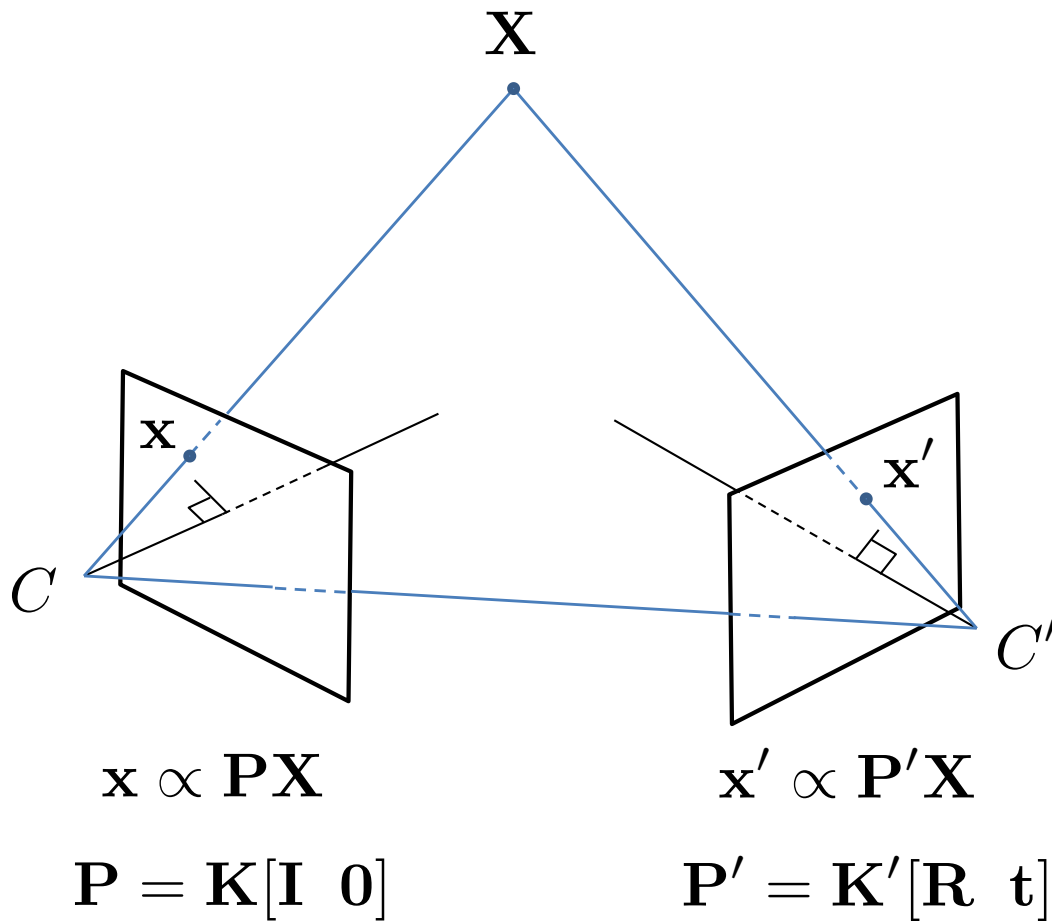
Essential matrixの分解

- 4通りの分解
 - ただし3次元点が両カメラの前に来るのは※だけ



Triangulation

- 2つのカメラ行列 P, P' が与えられたとき, $\mathbf{x}, \mathbf{x}'$ から X を求める

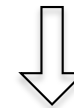


$$x = \frac{p_{00}X + p_{01}Y + p_{02}Z + p_{03}}{p_{20}X + p_{21}Y + p_{22}Z + p_{23}}$$

$$y = \frac{p_{10}X + p_{11}Y + p_{12}Z + p_{13}}{p_{20}X + p_{21}Y + p_{22}Z + p_{23}}$$

$$x' = \frac{p'_{00}X + p'_{01}Y + p'_{02}Z + p'_{03}}{p'_{20}X + p'_{21}Y + p'_{22}Z + p'_{23}}$$

$$y' = \frac{p'_{10}X + p'_{11}Y + p'_{12}Z + p'_{13}}{p'_{20}X + p'_{21}Y + p'_{22}Z + p'_{23}}$$



$$\begin{bmatrix} \mathbf{a}_0^\top \\ \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \mathbf{a}_3^\top \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Triangulation

triangulate.py

```
F = numpy.loadtxt('fundM.txt')
K = numpy.loadtxt('K.txt')
pts0 = numpy.loadtxt('pts0.txt').transpose()
pts1 = numpy.loadtxt('pts1.txt').transpose()

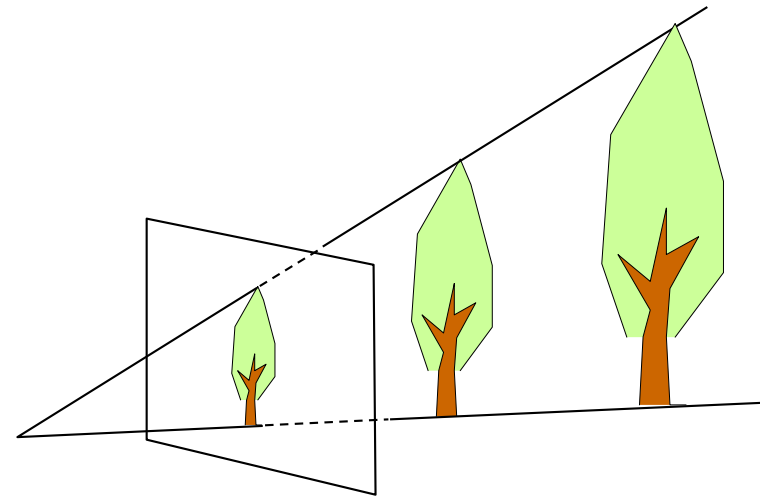
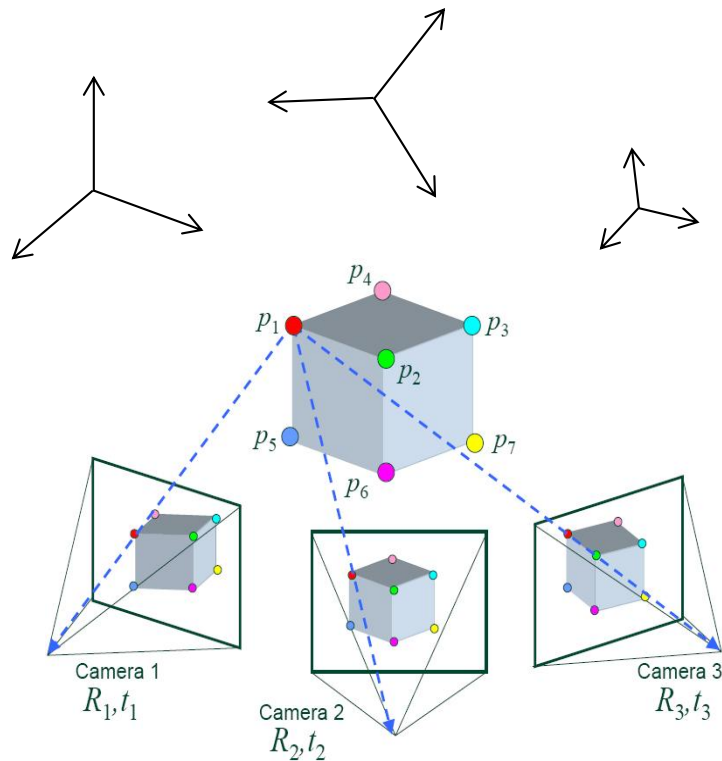
E = (K.transpose().dot(F)).dot(K)
Rt_list = decompE.decompose_E(E)
P0 = numpy.array([[1,0,0,0],[0,1,0,0],[0,0,1,0]])
P0 = K.dot(P0)
for i in range(4):
    P1c = numpy.zeros((3,4))
    P1c[0:3,0:3] = Rt_list[i][0]
    P1c[:,3] = Rt_list[i][1]
    P1 = K.dot(P1c)
    print ('P1=', P1)
    pts3d = cv2.triangulatePoints(P0, P1, pts0, pts1)
    pts3d = pts3d/pts3d[3,:] # normalize
    pts3d_1 = P1c.dot(pts3d) # X'=RX+t
    print (numpy.median(pts3d[2,:]))
    print (numpy.median(pts3d_1[2,:]))
```

エピポーラ幾何まとめ

- カメラ未校正(=内部パラメータ K が未知)でも基礎行列 F は求まる
- 校正済みなら $E=K'^T F K$ のように基本行列を導出できる
- E は分解して R と t を計算可能
 - t の絶対的大きさ(スケール)は決まらない
 - Triangulationも可能になり, カメラの相対姿勢と対応点の3次元座標が推定できたことになる

基本的な不定性

- 画像からの3次元復元では、世界座標系の取り方とスケールは不定
 - 視点が何枚あっても同じ



2視点から多視点へ： Structure-from-Motion (SfM)

- 問題： n 個のシーンの点を m 視点で撮影した画像（の点の座標）を元に、 n 点の3次元座標と m カメラの姿勢を求めたい

INPUT:

m image coordinates
of the n points

$$\mathbf{x}_{ij} = [x_{ij}, y_{ij}]$$

$$(i = 1, \dots, m)$$

$$(j = 1, \dots, n)$$

Image formation model

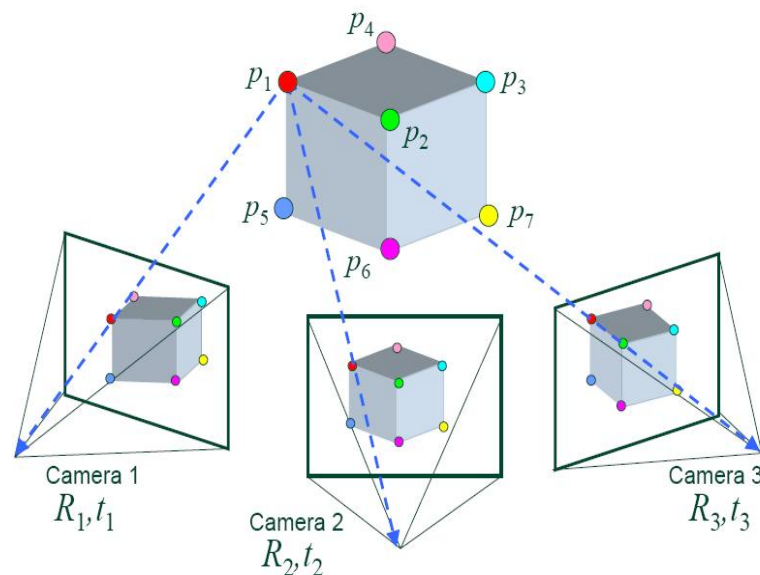
$$\mathbf{x}_{ij} \propto \mathbf{P}_i \mathbf{X}_j$$

OUTPUT:

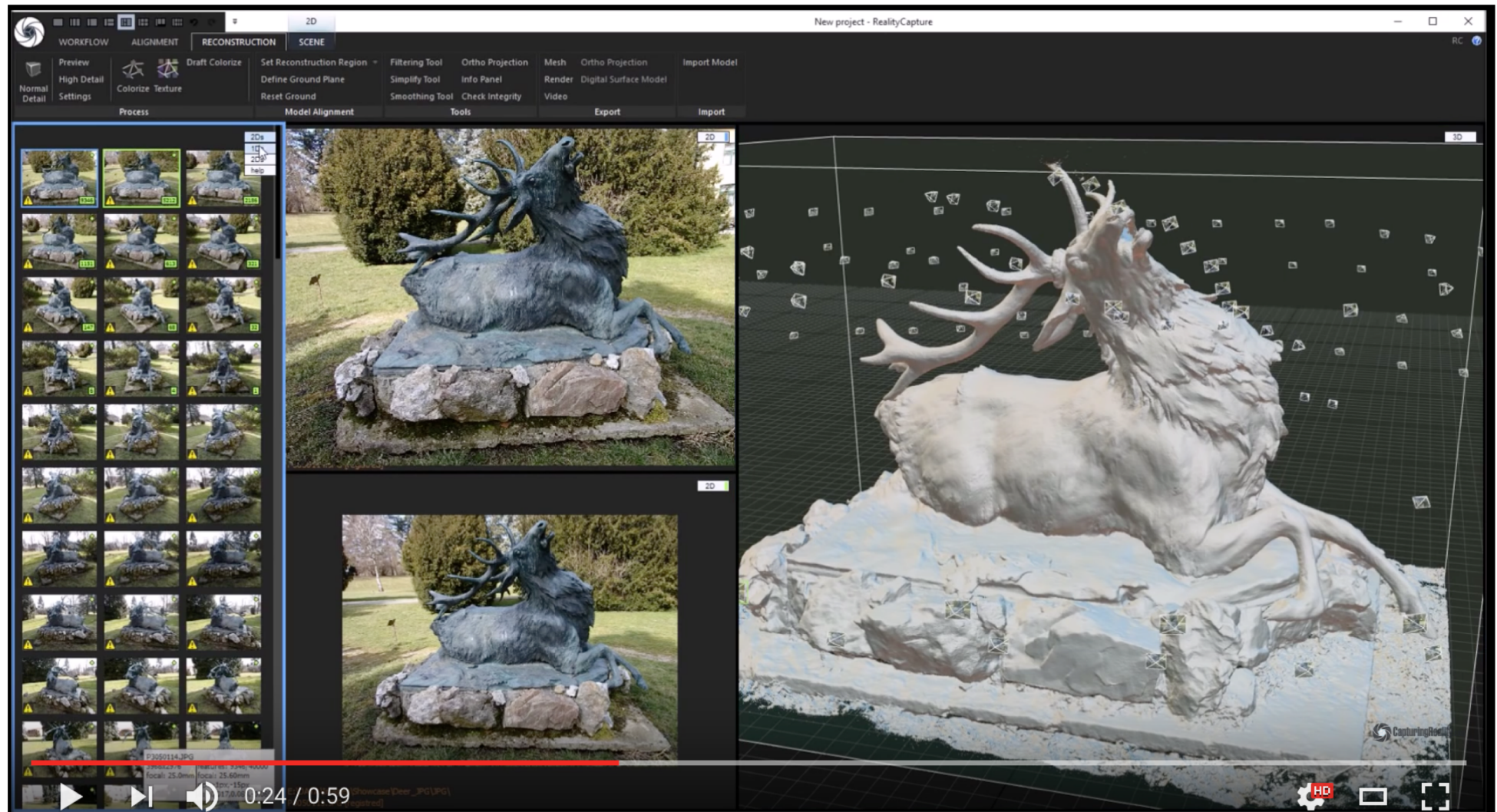
n point coordinates
 m camera info.

$$\mathbf{X}_j \quad (j = 1, \dots, n)$$

$$\mathbf{P}_i \quad (i = 1, \dots, m)$$



SfM (+MVS : Multi-View Stereo)

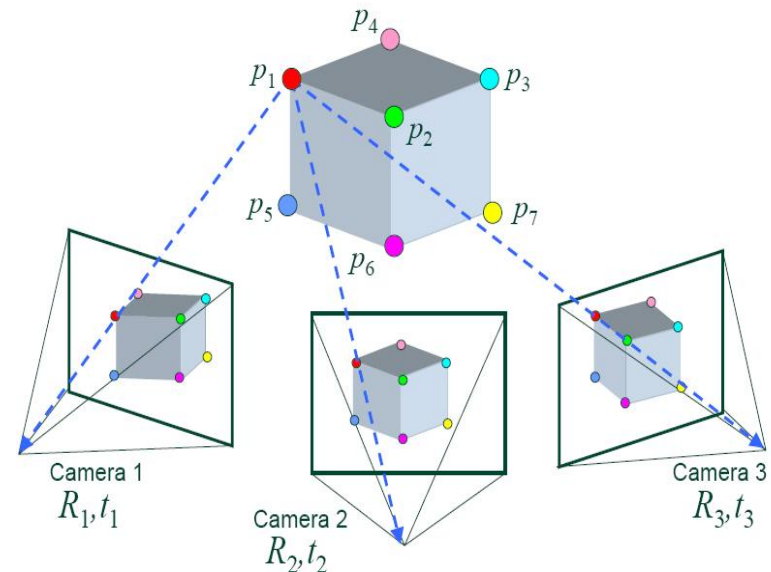


RealityCapture

www.capturingreality.com

(カメラが校正済みの場合の) SfMの計算の例

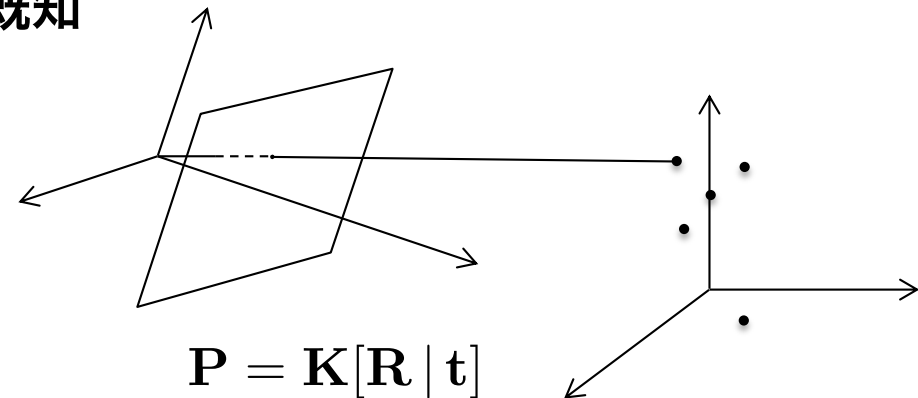
- a. 2視点間で基本行列を求める
 - 2枚の画像間で特徴点を抽出し, 対応を求める
 - スケール(視点間距離)は任意に決める(例えば長さ=1)
- b. 対応点をtriangulationし, 3次元座標を計算
- c. 3視点目のカメラ位置をPnPで求める
- d. 1-3視点あるいは2-3視点のペア間の対応点についてtriangulationを行い, 3次元座標を計算
- e. 4視点目以降もc,dを反復
- f. N視点でバンドル調整を実行



PnP: Perspective-N-Point (4章スライド)

- 3次元座標が既知の点群(≥ 3)の像からカメラの姿勢を計算
 - カメラの内部パラメータ (K) 既知

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \propto \mathbf{P} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}, \quad (i = 1, \dots, n)$$



```
while 1:
    stat, image = cap.read(0)

    retval, centers = cv2.findCirclesGrid(image, (patw, path))

    if retval == True:
        ret, rvec, tvec = cv2.solvePnP(objp, centers, K, dist)
        Rmat, jac = cv2.Rodrigues(rvec)
        cam_model.draw_cube(image, K, ¥
                               numpy.array([[0.0,0.0,0.0]], float).T,¥
                               3.0, Rmat, tvec)
```

pnp.py

行列分解を通じたSfMの理解

Known Unknown Unknown

↓ ↓ ↙

$$\mathbf{x}_{ij} \propto \mathbf{P}_i \mathbf{X}_j$$
$$(i = 1, \dots, m, \quad j = 1, \dots, n)$$

↓

$$\begin{array}{c} \xleftarrow{n} \quad \xrightarrow{n} \end{array} \begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} & \cdots & \mathbf{x}_{1n} \\ \mathbf{x}_{21} & \mathbf{x}_{22} & \cdots & \mathbf{x}_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{x}_{m1} & \mathbf{x}_{m2} & \cdots & \mathbf{x}_{mn} \end{bmatrix} \sim \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \vdots \\ \mathbf{P}_m \end{bmatrix} \begin{array}{c} \xleftarrow{4} \quad \xrightarrow{n} \end{array} \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 & \cdots & \mathbf{X}_n \end{bmatrix} \begin{array}{c} \updownarrow 4 \end{array}$$

A large number of data

Unknowns

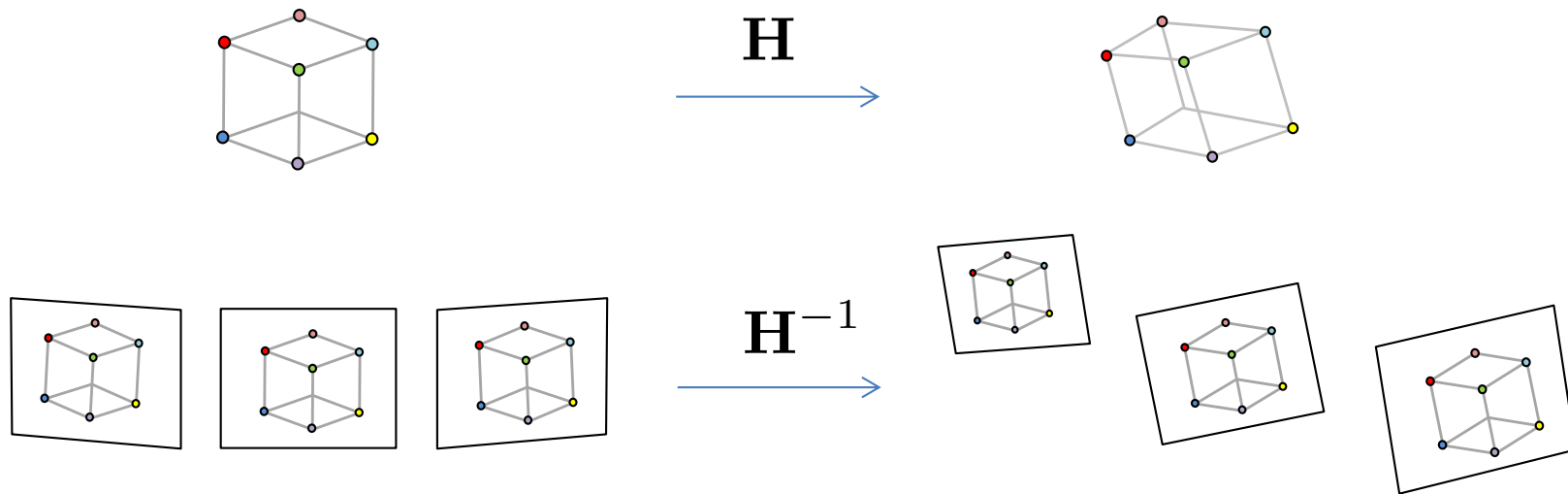
射影的不定性：Projective ambiguity

- 復元の不定性

- 画像だけでは一意には決まらない

$$\mathbf{x}_{ij} \propto \mathbf{P}_i \mathbf{X}_j = \mathbf{P}_i \mathbf{H}^{-1} \mathbf{H} \mathbf{X}_j = (\mathbf{P}_i \mathbf{H}^{-1}) (\mathbf{H} \mathbf{X}_j) = \mathbf{P}'_i \mathbf{X}'_j$$

- 任意の $\mathbf{H}$ (3D projective trans.) 分の自由度 (15 DOF)



射影的不定性の解決

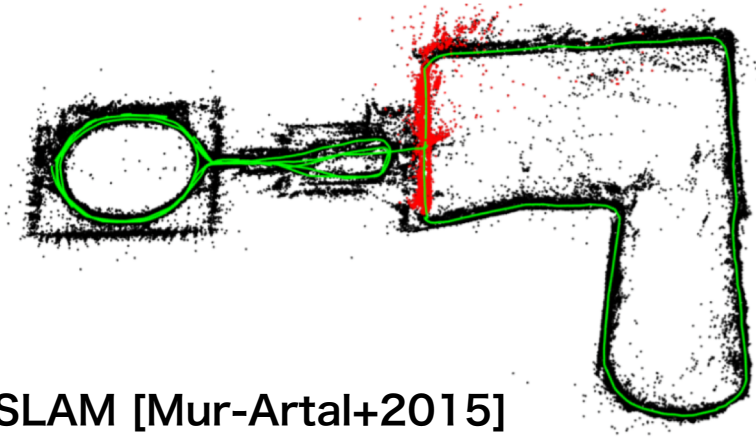
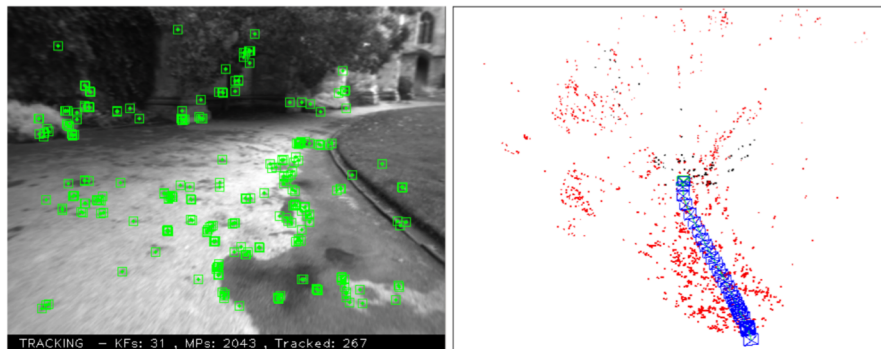
- カメラが校正済み＝射影的不定性は存在しない
 - カメラを校正する＝内部パラメータを知る
 - ただし「基本的な不定性」は残る
- 2視点幾何との関係
 - カメラが校正済み＝基本行列(Essential matrix)
 - カメラが未校正＝基礎行列(Fundamental matrix)
- 自己校正 (autocalibration)
 - 内部パラメータが部分的に既知(残りが未知)の場合, 適当な数の視点とそれらの画像間の対応点が求まれば, 未知な内部パラメータを決められる

$$\underbrace{\mathbf{x} \propto \mathbf{P}\mathbf{X}}_{\mathbf{K} [\mathbf{R} \mid \mathbf{t}]} \quad \mathbf{K} = \begin{bmatrix} f & s & x_0 \\ 0 & \alpha f & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Visual SLAM

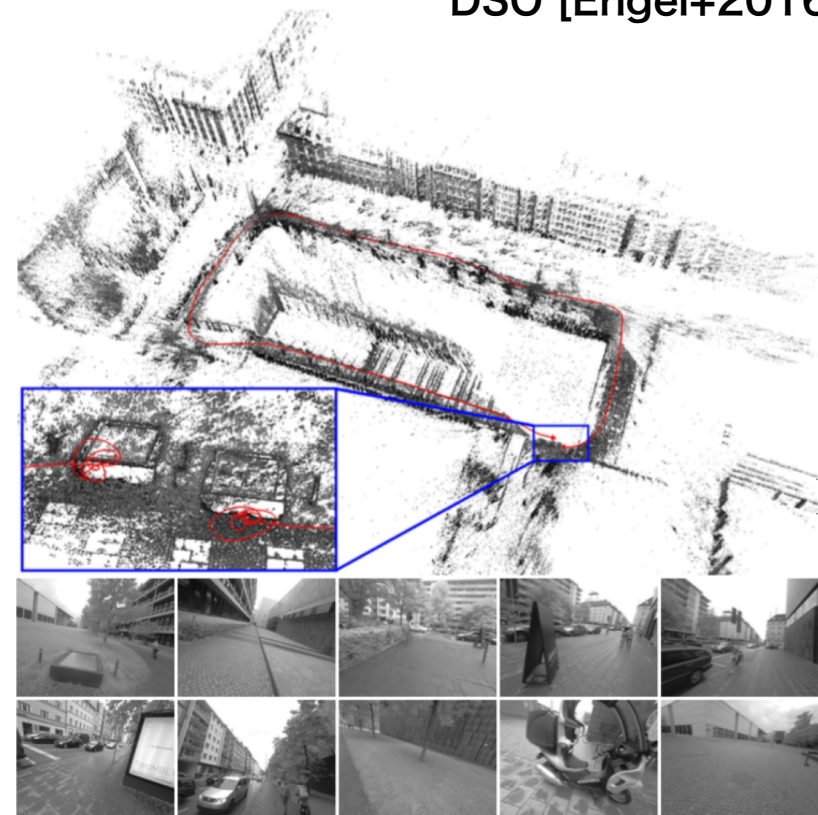
(Simultaneous Localization and Mapping)

- Visual SLAM = カメラの運動にともなって得られる画像を使って、**実時間にSfMの計算を行うこと**
 - カメラの運動(各視点の姿勢) の推定のみに関心がある場合、Visual Odometry (VO)と呼ばれることも



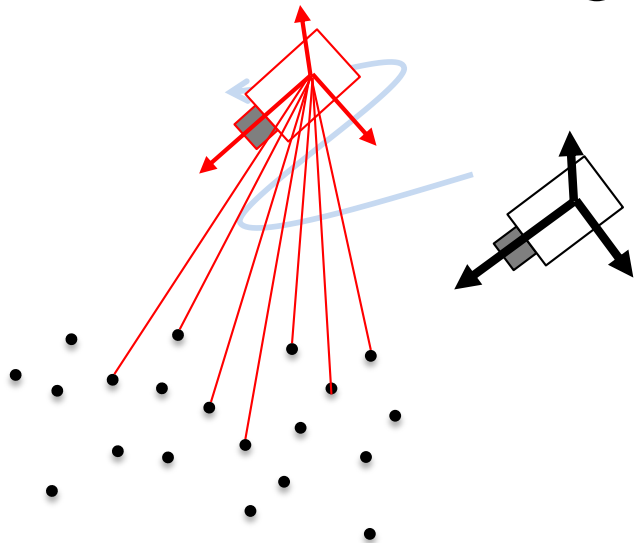
ORB-SLAM [Mur-Artal+2015]

DSO [Engel+2016]

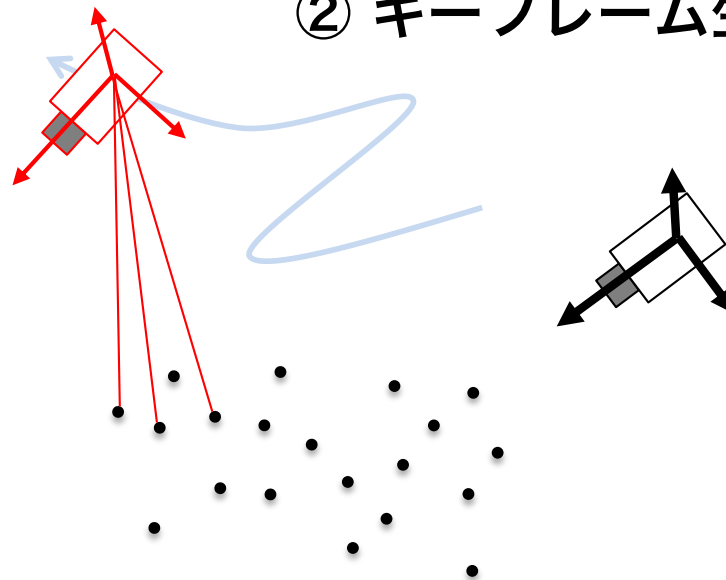


Visual SLAMの1 サイクル

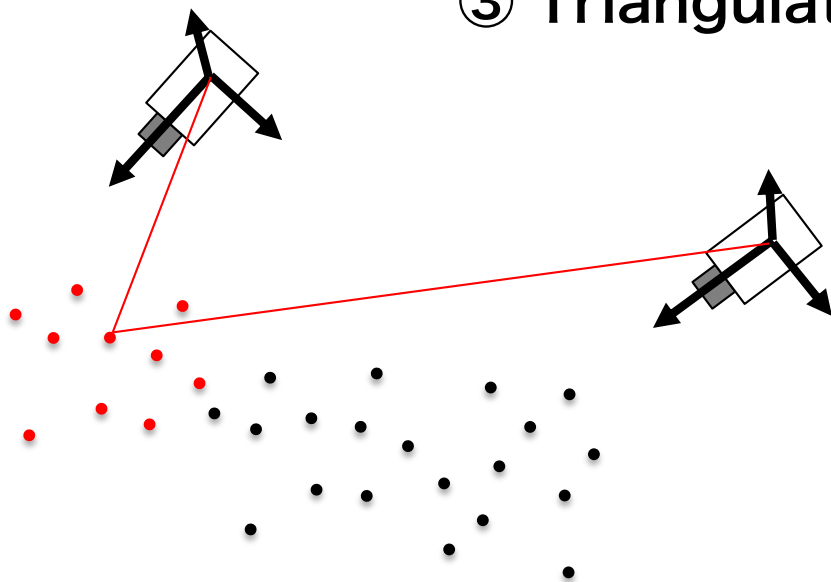
① PNP



② キーフレーム生成



③ Triangulation



④ バンドル調整
(全パラメータ同時最適化)

