映像解析中間レポート

システム情報科学府　情報知能工学専攻 M1
2IE20338G 松田 征也

第 1 章 8point アルゴリズム

1.実験に使用した画像を図 1.1~1.4 に示す.

図 1.1:奥行きの変化の大きい写真(a)

図 1.2:奥行きの変化の大きい写真(b)

図 1.3 : 奥行きの変化が少ない写真(a)

図 1.4 : 奥行きの変化が少ない写真(b)

2.F 行列の算出

2.1 奥行き変化の大きい場合の F 行列

F= [[-7.92879050e-07,-4.74764982e-06 ,-4.81180918e-03],
[5.32627152e-06 ,-2.26028255e-06, 5.30108913e-04],
[3.12206459e-03 , 2.39598985e-05 , 9.99983409e-01]]

2.2 奥行き変化の小さい場合の F 行列

F=[[4.96020448e-06,4.13348359e-05,-1.38199341e-02],
[-4.66094487e-05,2.60887530e-05,3.15572033e-02],
[9.73710799e-03,-3.78475109e-02,9.98642029e-01]]

3.F 行列の固有値とランクの確認

3.1 奥行変化の大きい場合

F の固有値= [9.99968398e-01　1.31259695e-05 -1.16858244e-06]

F の rank= 3

3.2 奥行の変化の小さい場合

F の固有値=[9.97309472e-01,-1.39498794e-07,1.36374515e-03]

F の rank= 3

4.F 行列の算出に用いたエポピーラ線の描写

4.1 奥行き変化の大きな写真

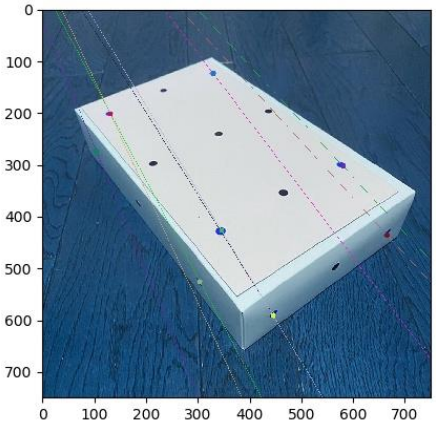

図 1.5：奥行きの変化が大きい写真(a)

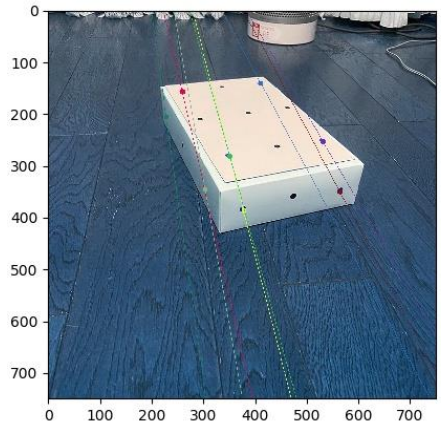

図 1.6：奥行きの変化が大きい写真(b)

4.2 奥行き変化の小さい写真

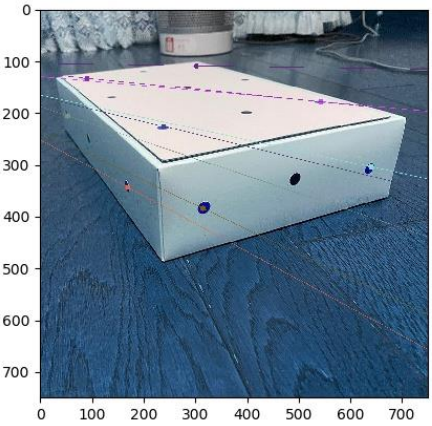

図 1.7：奥行きの変化が少ない写真(a)

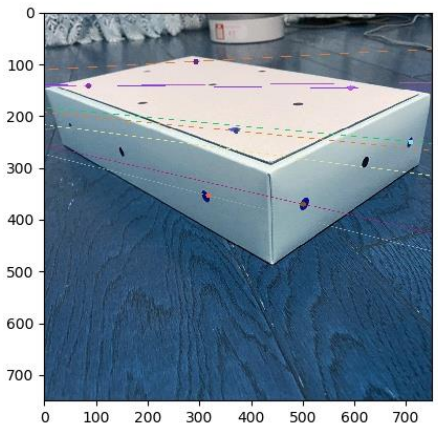

図 1.8：奥行きの変化が少ない写真(b)

5. F 行列の計算に用いた点以外の点に対応するエピポーラ線を画像上に描画する.

5.1 奥行き変化の小さい写真

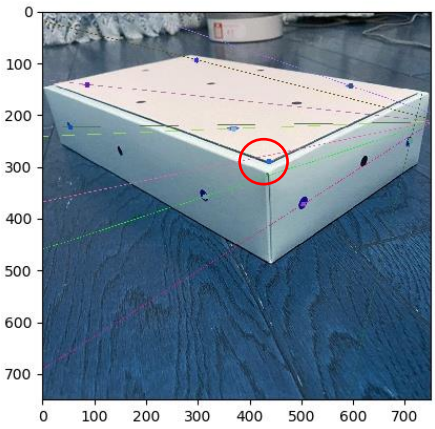

図 1.9：8 点の内側にある点(a)

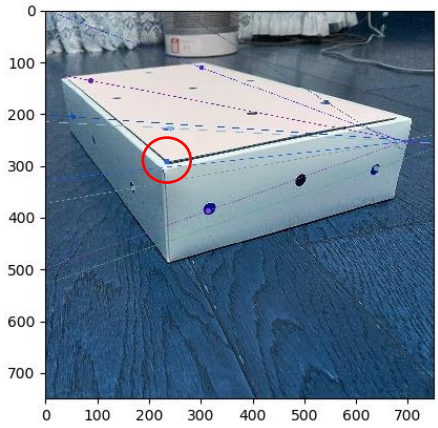

図 1.10：8 点の内側にある点(b)

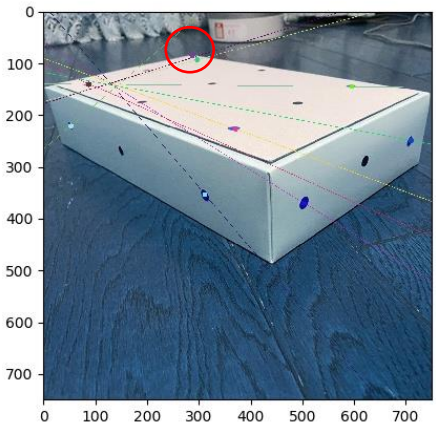

図 1.11：8 点の外側にある点(a)

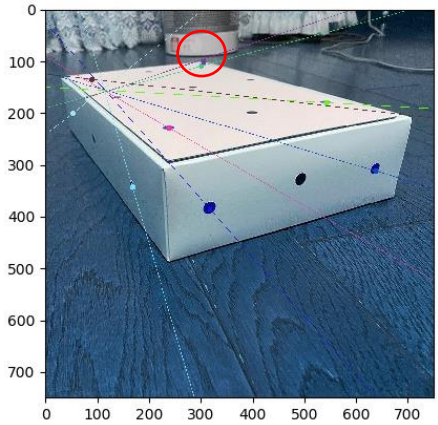

図 1.12：8 点の外側にある点(b)

5.2 奥行き変化の大きい写真

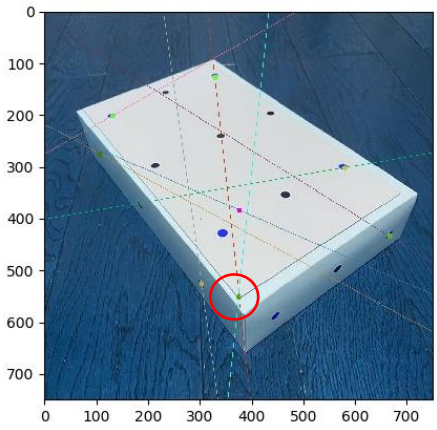

図 1.13：8 点の内側にある点(a)

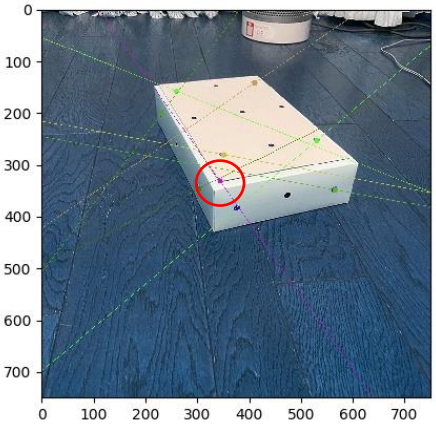

図 1.14：8 点の内側にある点(a)

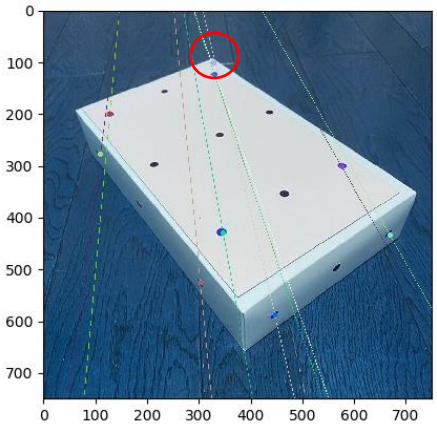

図 1.15：8 点の外側にある点(a)

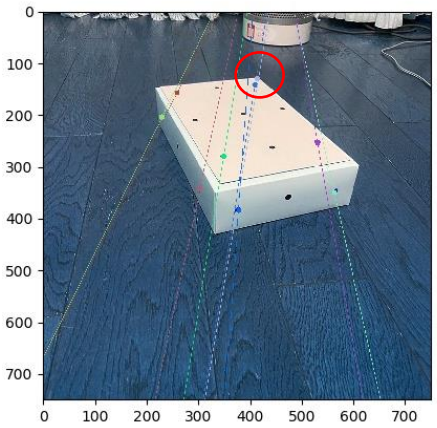

図 1.16：8 点の外側にある点(a)

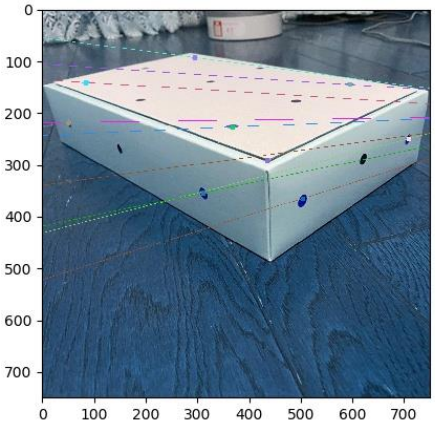

図 1.17：8 点以外の複数点(a)

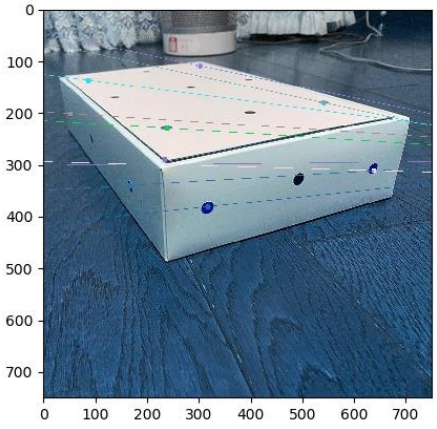

図 1.18：8 点以外の複数点(b)

第２章 15 点に対する考察

1. F 行列の算出

1.1 奥行き変化の大きい写真

F= [[-1.70133504e-06　4.02620998e-06　3.71016410e-05]
[2.79502757e-06　3.49285625e-06 -9.51483582e-03]
[4.04545282e-04　1.50016249e-03　9.99953525e-01]]

1.2 奥行きの変化の小さい写真

F= [[-6.42635896e-06, -3.19795009e-05, 8.48235720e-03]
[7.14364847e-05 ,-1.19070177e-04 ,3.48823293e-02]
[-1.67613581e-02, -2.07014173e-02, 9.99000381e-01]]

2. F 行列のランクと固有値

2.1 奥行の変化の大きい画像

F の固有値= [9.99939265e-01 -2.98746758e-06　1.90387105e-05]

F の rank= 3

2.2 奥行の変化の小さい画像

F の固有値= [9.98134569e-01 -1.63769904e-05　7.56691835e-04]

F の rank= 3

3. F 行列の計算に用いた点に対応するエピポーラ線を画像上に描画する.

3.1 奥行き変化の大きい写真

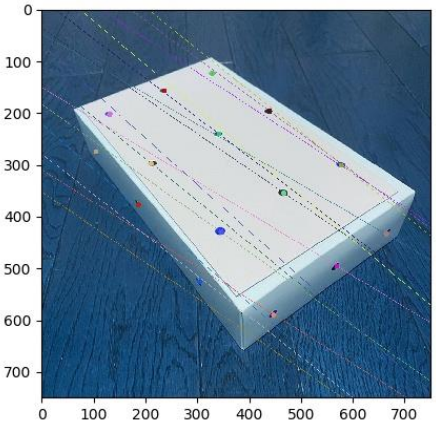

図 1.17：奥行きの変化が大きい写真(a)

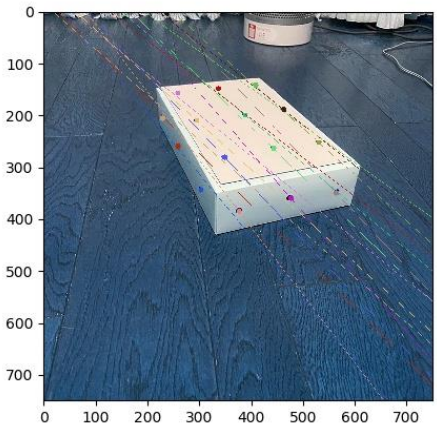

図 1.18：奥行きの変化が大きい写真(b)

3.2 奥行き変化の小さい写真

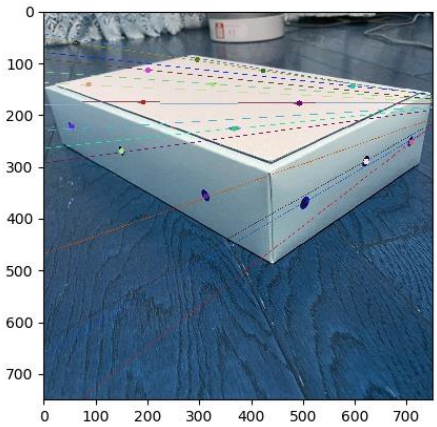

図 1.19：奥行きの変化が小さい写真(a)

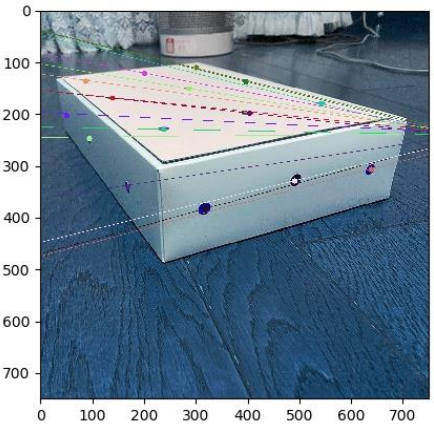

図 1.20：奥行きの変化が小さい写真(b)

4.　F 行列の計算に用いた点以外の点に対応するエピポーラ線を画像上に描画する.

4.1 奥行き変化の大きい写真

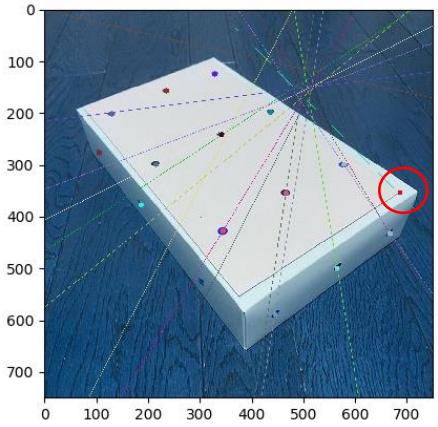

図 1.21：15 点の外側にある点(a)

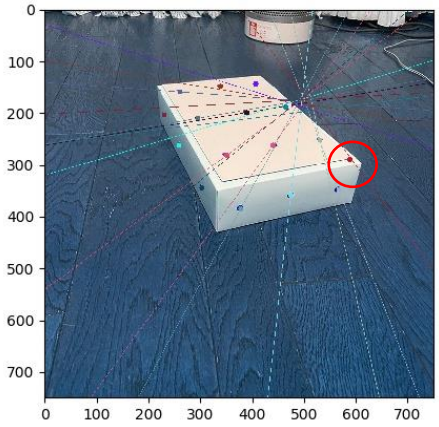

図 1.22：15 点の外側にある点(b)

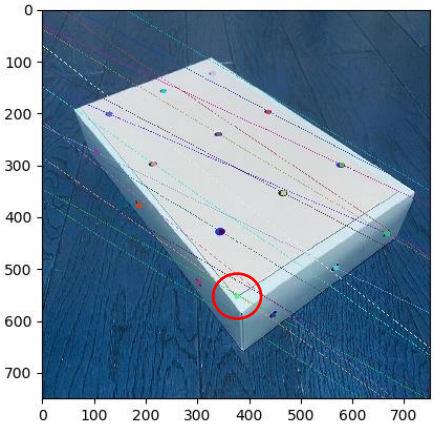

図 1.22：15 点の内側にある点(a)

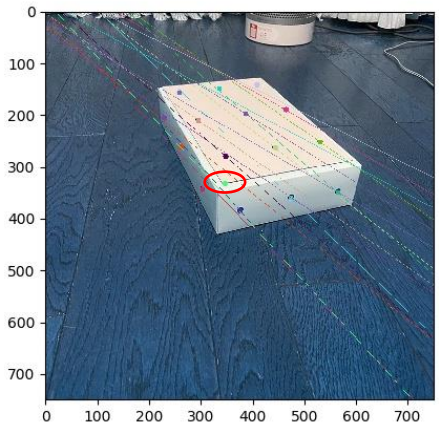

図 1.23：15 点の内側にある点(b)

4.2 奥行き変化の小さい写真

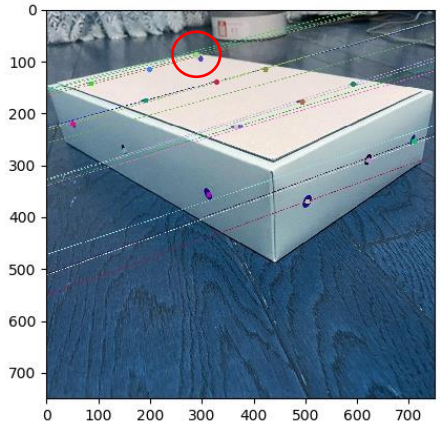

図 1.24：15 点の外側にある点(a)

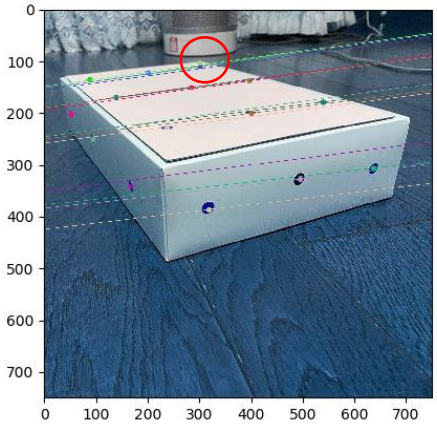

図 1.25：15 点の外側にある点(b)

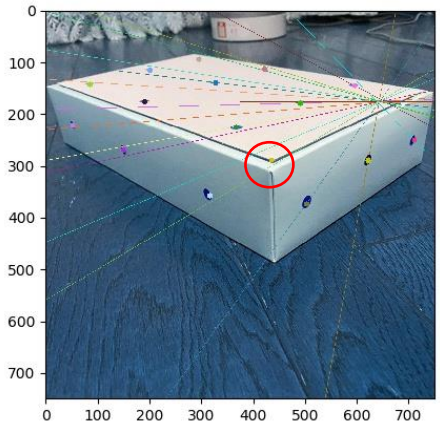

図 1.26：15 点の内側にある点(a)

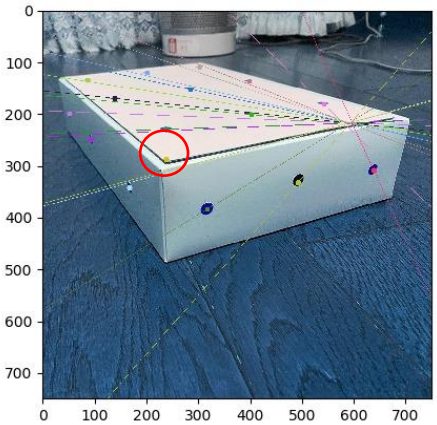

図 1.27：15 点の内側にある点(b)

第３章 Kruppa 方程式

4.1 カメラの焦点距離を求める.

F= [[2.37037353e-06　4.44548747e-07 -4.63749836e-04]
　[-4.02733961e-07　5.20371580e-06 -7.26590564e-03]
　[-7.48856934e-04　1.78894808e-03　9.99971615e-01]]

(f1,f2)= (2311.45874094571, 2188.95014909139)

4.2 画像の中心を出す

元の画像の中心=(375.0,375.0)

(c1_u, c1_v) = (374.999999999997, 374.999999999998)

(c2_u, c2_v) = (374.999999999552 0.e 17*I, 374.499999999951 + 0.e 15*I)

4.3 考察

計算した f1,f2 ともほぼ同じ値をもつことが確認された.同じカメラを用いて撮影したためである．誤差が生じたのはカメラ内部のパラメータを求める際に、未知の変数が 2 個である制限があるためだと考えられる．画像の中心はほぼ同じような結果になった.

第４章 ソースコード

実際に作成したプログラムを以下に示す.

point_get.py

```
import sys
from typing import List, Tuple
import cv2
import os.path
class mouseParam:
    def __init__(self, input_img_name):
        #マウス入力用のパラメータ
        self.mouseEvent = {"x":None, "y":None, "event":None, "flags":None}
        #マウス入力の設定
        cv2.setMouseCallback(input_img_name, self.__CallBackFunc, None)
    #コールバック関数
    def __CallBackFunc(self, eventType, x, y, flags, userdata):
        self.mouseEvent["x"] = x
        self.mouseEvent["y"] = y
        self.mouseEvent["event"] = eventType
        self.mouseEvent["flags"] = flags
    #マウス入力用のパラメータを返すための関数
    def getData(self):
        return self.mouseEvent
    #マウスイベントを返す関数
    def getEvent(self):
        return self.mouseEvent["event"]
    #マウスフラグを返す関数
    def getFlags(self):
        return self.mouseEvent["flags"]
    #x の座標を返す関数
    def getX(self):
        return self.mouseEvent["x"]
    #y の座標を返す関数
    def getY(self):
        return self.mouseEvent["y"]
```

```
    #x と y の座標を返す関数
    def getPos(self):
        return (self.mouseEvent["x"], self.mouseEvent["y"])
def get_point():
    file1 = input("画像 1 のファイル名")
    file2 = input("画像 2 のファイル名")
    file=[file1,file2]
    coordinate_point = []
    img=[]
    for f in file:
        #入力画像
        path=os.path.join(r'C:¥Users¥yaase¥Desktop¥eizou', f)
        read = cv2.imread(path)
        img.append(read)
        #表示する Window 名
        window_name = "input window"
        #画像の表示
        cv2.imshow(window_name, read)
        #コールバックの設定
        mouseData = mouseParam(window_name)
        while 1:
            cv2.waitKey(20)
            #左クリックがあったら表示
            if mouseData.getEvent() == cv2.EVENT_LBUTTONDOWN:
                coordinate_point.append(mouseData.getPos())
                coordinate_point=list(dict.fromkeys(coordinate_point))
            #右クリックがあったら終了
            elif mouseData.getEvent() == cv2.EVENT_RBUTTONDOWN:
                break
        cv2.destroyAllWindows()
    print(coordinate_point)
    uvmat=[coordinate_point[idx]+coordinate_point[idx+int(len(coordinate_point)/2)]
            for idx in range(int(len(coordinate_point)/2))]
    print(uvmat)
    print("Finished")
    return img,uvmat
```

calc_Fmatrix.py

```
import numpy as np
import numpy.linalg as LA
def calc_Fmatrix(uvmat):
    mat = np.zeros((8, 9))
    for i in range(8):
        mat[i, 0] = uvmat[i, 0] * uvmat[i, 2]
        mat[i, 1] = uvmat[i, 1] * uvmat[i, 2]
        mat[i, 2] = uvmat[i, 2]
        mat[i, 3] = uvmat[i, 0] * uvmat[i, 3]
        mat[i, 4] = uvmat[i, 1] * uvmat[i, 3]
        mat[i, 5] = uvmat[i, 3]
        mat[i, 6] = uvmat[i, 0]
        mat[i, 7] = uvmat[i, 1]
        mat[i, 8] = 1.0
    # mat を特異値分解
    U, D, V = LA.svd(mat)
    #  最小の固有値を探す
    print("D=", D)
    print("V=", V)
    print(V[8])
    # V から固有値最小の固有ベクトルを取り出して F にする
    F = V[8].reshape((3, 3))
    print("F=", F)
    # F の固有値は[1,0,0]になる
    value, vec = LA.eig(F)
    rank=np.linalg.matrix_rank(F)
    print("F の固有値=", value)
    print("F の rank=", rank)
    for i in range(8):
      x1 = np.array([uvmat[i, 0], uvmat[i, 1], 1.0])
      x2 = np.array([uvmat[i, 2], uvmat[i, 3], 1.0])
      x1 = np.dot(N, x1.T)
      x2 = np.dot(N, x2.T)
      # x2^T * F * x1 = 0  のチェック
      a = np.dot(x2.T, F)
```

```
        print(np.dot(a, x1))
    return F
def calc_epipole(F):
    # F^T を特異値分解
    U, D, V = LA.svd(F)
    # V から固有値最小の固有ベクトルを取り出す
    fvec = V[2]
    return fvec / fvec[2]
```

draw_img.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def draw_lines(img1, img2, lines, pts1, pts2):
    '''img1 - img2 上の点に対応するエピポーラ線を描画する画像
        lines - 対応するエピポーラ線 '''
    r, c,_ = img1.shape
    #img1 = cv2.cvtColor(img1, cv2.COLOR_GRAY2BGR)
    #img2 = cv2.cvtColor(img2, cv2.COLOR_GRAY2BGR)
    for r, pt1, pt2 in zip(lines, pts1, pts2):
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0, y0 = map(int, [0, -r[2] / r[1]])
        x1, y1 = map(int, [c, -(r[2] + r[0] * c) / r[1]])
        img1 = cv2.line(img1, (x0, y0), (x1, y1), color, 1)
        img1 = cv2.circle(img1, tuple(pt1), 5, color, -1)
        img2 = cv2.circle(img2, tuple(pt2), 5, color, -1)
    return img1, img2

def draw_lines_show(img1,img2,uvmat,F):

    # 右画像(二番目の画像)中の点に対応するエピポーラ線の計算
    # 計算したエピポーラ線を左画像に描画
    pts1 = np.array([[i[0], i[1]] for i in uvmat])

    pts2 = np.array([[i[2], i[3]] for i in uvmat])
    lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1, 1, 2), 2, F)
```

```
    lines1 = lines1.reshape(-1, 3)
    img5, img6 = draw_lines(img1, img2, lines1, pts1, pts2)

    # 左画像(一番目の画像)中の点に対応するエピポーラ線の計算
    # 計算したエピポーラ線を右画像に描画
    lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1, 1, 2), 1, F)
    lines2 = lines2.reshape(-1, 3)
    img3, img4 = draw_lines(img2, img1, lines2, pts2, pts1)

    # 結果の表示
    #plt.subplot(121), plt.imshow(img5)
    #plt.subplot(122), plt.imshow(img3)
    #plt.savefig("epipolar.jpg")
    plt.imshow(img5)
    plt.savefig("epipolar_img1.jpg")
    plt.imshow(img3)
    plt.savefig("epipolar_img2.jpg")
```

calc_camera_matrix.py

```
import sympy as sy
import numpy as np
def calc_inside_param(F, e, p11, p12, p21, p22, calc_phase="f"):
    # a1, a2 を求める
    sy.var('a1, a2')
    if calc_phase == "f":
        A1 = sy.Matrix([[a1, 0, p11],
                        [0, a1, p12],
                        [0, 0, 1]])
        A2 = sy.Matrix([[a2, 0, p21],
                        [0, a2, p22],
                        [0, 0, 1]])
    elif calc_phase == "c1":
        A1 = sy.Matrix([[p11, 0, a1],
                        [0, p11, a2],
                        [0, 0, 1]])
        A2 = sy.Matrix([[p12, 0, p21],
```

```
                                [0, p12, p22],
                                [0, 0, 1]])
    elif calc_phase == "c2":
        A1 = sy.Matrix([[p11, 0, p21],
                                [0, p11, p22],
                                [0, 0, 1]])
        A2 = sy.Matrix([[p12, 0, a1],
                                [0, p12, a2],
                                [0, 0, 1]])
    sy.var('t')
    vec_t = sy.Matrix([1, t, 0])
    # e, F を sympy に変換
    e = sy.Matrix([e[0], e[1], e[2]])
    F = sy.Matrix([[F[0, 0], F[0, 1], F[0, 2]],
                           [F[1, 0], F[1, 1], F[1, 2]],
                           [F[2, 0], F[2, 1], F[2, 2]]])
    # eq1 = (e × t)^T * A1 * A1^T * (e1 × t) = 0
    tmp1 = A1.transpose() * e.cross(vec_t)
    eq1 = sy.expand((tmp1.transpose() * tmp1)[0])
    # eq1 から t^0, t^1, t^2 の係数を取り出す
    k10 = eq1.coeff(t, 0)
    k11 = eq1.coeff(t, 1)
    k12 = eq1.coeff(t, 2)
    # eq2 = (F^T × t)^T * A2 * A2^T * (F^T × t) = 0
    tmp2 = A2.transpose() * F.transpose() * vec_t
    eq2 = sy.expand((tmp2.transpose() * tmp2)[0])
    # eq2 から t^0, t^1, t^2 の係数を取り出す
    k20 = eq2.coeff(t, 0)
    k21 = eq2.coeff(t, 1)
    k22 = eq2.coeff(t, 2)
    # expr1 = k10*k21 - k11*k20
    expr1 = sy.expand(k10 * k21 - k11 * k20)
    # expr2 = k11*k22 - k21*k12
    expr2 = sy.expand(k11 * k22 - k21 * k12)
    print("expr1 = {}".format(expr1))
    print("expr2 = {}".format(expr2))
```

```
    # expr1 = expr2 = 0 を解く(解は a1, a2)
    ans = sy.solve([expr1, expr2], [a1, a2])
    print("(a1, a2)=¥n{}".format(ans))
    return ans
```

main.py

```
from pointget import mouseParam,get_point
from calc_Fmatrix import calc_Fmatrix, calc_epipole
from draw_image import draw_lines,draw_lines_show
from calc_camera_matrix import calc_inside_param
import numpy as np
import numpy.linalg as LA
import sympy as sy
import os
from PIL import Image
import cv2
if __name__ == '__main__':
    # 画像から F を求め、f, c を算出 #
    width = 750
    height = 750
    img,uvmat=get_point()
    img1,img2=img[0],img[1]
    #img1[:,:,0],img1[:,:,1]=img1[:,:,1],img1[:,:,0]
    #img2[:,:,0],img2[:,:,1]=img2[:,:,1],img2[:,:,0]
    uvmat=np.array(uvmat)
    mat=np.zeros((8,9))
    # 正解の中心座標を出しておく
    ans_c_v1, ans_c_u1, _ = np.array(img1.shape) / 2
    ans_c_v2, ans_c_u2, _ = np.array(img2.shape) / 2
    print(ans_c_u1, ans_c_v1, ans_c_u2, ans_c_v2)
    # F を算出
    F = calc_Fmatrix(uvmat)
    #エピポーラ線を描画
    draw_lines_show(img1, img2, uvmat, F)
    # エピポールを算出
    e1 = calc_epipole(F.T)
```

```
    print("e1: {}".format(e1))
    a = input()
    # 内部パラメータを算出
    ans = calc_inside_param(F, e1, ans_c_u1, ans_c_v1, ans_c_u2, ans_c_v2,
calc_phase="f")
    #calc_inside_param(F, e1, f1, f2, ans_c_u2, ans_c_v2, calc_phase="c1")
    #calc_inside_param(F, e1, f1, f2, ans_c_u1, ans_c_v1, calc_phase="c2")
```