**PA2 Kornkamol Anasart 20211182**

(This assignment was written in Python 3.6 All requirements in README.md file)

Disclaimer: I tried to not use open function, but vlfeat for SIFT extraction and feature matching is not working on python, so I use cv2.ORB feature extraction and feature matching from OPENCV. And open3D to write PLY file.

Let’s me introduce my code, my code separated into 2 python files including fundamental\_m.py and triangulation.py. Let’s look at fundamental\_m.py

Overview of finding fundamental matrix:

Import image 🡪 feature extraction 🡪 feature matching 🡪 RANSAC [fivepoint\_calibration]

Graphical user interface, text, application

Description automatically generated

Fig1: Overview code for finding fundamental matrix

Detail:

1. Import images in greyscale format into variable imga [sfm01.jpg] and imgb [sfm02.jpg] which we called image A and image B. Image A is based image for viewing point.

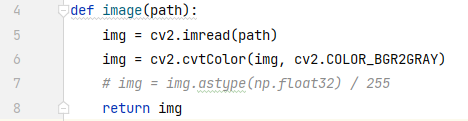


Fig2: image function input is path of an image and output is greyscale image

1. Feature extraction part as I said I cannot use vlfeat in python3 so I switched to OpenCV ORB function. Fig3, shown my code used cv2.ORB\_create to create feature extraction which compute into descriptor and keypoints from both image A and image B. ‘num\_f ‘ is number of feature that we want

A screenshot of a computer

Description automatically generated with medium confidence

Fig3: Feature extraction function

1. Once we got keypoints and descriptor from feature extraction function. We use cv2.BFMatcher for Brute-force matcher constructor. But not all matched point is good, so we need to set threshold for distance of matched points to find good matched following OpenCV document tutorial. And extract point coordinates (x, y) on image A and image B. The result of our possible mathcing points is 296 points.

Text, application

Description automatically generated

Fig4: matching fucntion

A picture containing text, bedclothes

Description automatically generated

Fig5: Matching point using cv.drawKeypointsA picture containing indoor

Description automatically generated

Fig6: Plot matching point coordinates (x, y) extracted from keypoints

1. Find fundamental matrix using fivepoint calibration. I implemented fivepoint calibration from MATLAB version into Python version which I had already compared result from MATLAB version and python version. So, if you investigate five-point calibration code, the function wants point coordinates in size [5, 3]. So, this is mean it needs homogeneous coordinate [x, y, 1]. So, we need to make it into homogenous coordinate first.

The concept is we randomly choose 5 matched coordinates from image A and B. And put into fivepoint calibration function and run it multiple time in RANSAC to find best fundamental matrix which have maximum inlier. So, we need to set parameter*, ‘most inliers’* for setup criteria of number of inlier points which is a greater number of inliers, you will get better fundamental matrix, *‘threshold’* is number of error or epipolar distance between epipolar line and random point coordinates. *‘iteration, itr’* is how many times that you want to run RANSAC. *‘pts\_a’* and *‘pts\_b’* is matched point coordinates on image A and B.

Fivepoint calibration function give more than one fundamental matrix in one time so we add another for loop to extract fundamental matrix and change into size [3x3].

We also set empty variable a\_inlier and b\_inlier to store inlier point coordinates. Another for loop for extract one by one coordinates to calculate error or distance between coordinates and epipolar line [err = Q1tFQ2]. And we add condition if error is less than our pre define threshold that point is going to be stored in a\_inlier and b\_inlier.

Text

Description automatically generated with medium confidence

Fig7: RANSAC [fivepoint calibration]

Note from my experiment: How to do experiment, increase most\_inlier gradually such as 50 to 100, 200. You will discover that when you set large of ‘most\_inlier’, you need to run RANSAC multiple time to find fundamental matrix that meet your criteria.

A screenshot of a computer

Description automatically generated with medium confidence

Fig8: My best fundamental matrix with 241 inlier points from 296 possible matched points.

After we obtain best fundamental matrix. We use fundamental matrix to find 3D point coordinates.

1. Compute essential matrix: E = K1t FK2 from : intrinsic matrix K, which K1 = K2 because it’s same camera.

A screenshot of a computer

Description automatically generated with medium confidence

Fig9: compute essential matrix

1. Decompose essential matrix into K[R|t] which is R = UWVt and t = u3 = last column of U. So we use np.linalg.svd to obtain U and Vt. (R is rotation matrix and t is translation matrix)

Text

Description automatically generated with low confidence

Fig:10 SVD computation

1. There is 4 possible way of camera’s position which give different rotation matrix. I assigned into case1 P' = [UWVt | +u3], case2 P' = [UWVt | -u3], case3 P' = [UWtVt | +u3], case4 P' = [UWtVt | -u3]

P, P’ is camera matrix of image A and B which P = [I | 0], W = [[0, -1, 0], [1, 0, 0], [0, 0, 0]]. So now we have 4 camera matrix Pb1(P’1), Pb2 (P’2), Pb3(P’3), Pb4 (P’4).

Graphical user interface, text, application

Description automatically generated

Fig11: 4 possible camera’s positions

1. Triangulation, finding 3D point (X) from (x, y). We need to calculate matrix A. from fig 12. And fin matrix B to solve AX = b. Dot product between [I | (x,y)] and P[:, 2] = R

Text, table

Description automatically generatedA picture containing calendar

Description automatically generated

Fig12 Matrix A equation

Graphical user interface, text, application

Description automatically generated

Fig13: Triangulation

1. Choose camera position. (I choose it manually)

A picture containing clock

Description automatically generated

Fig 14: 4 case possible camera positions

Case1: P = [I |0], P’ = Pb1, X = X1, depth of point (z coordinate) should be all positive.

Fig14: Camera’s position

A screenshot of a computer

Description automatically generated with medium confidence

Fig15: case 1

Case 2: P = [I|0], P’ = Pb2, X = X2, depth of point should be in range positive and negative

Text

Description automatically generated

Fig16: case2

Case 3: P=[I|0], P’ = Pb3, X=X3, depth of points at camera B should be negative

Text

Description automatically generated

Fig 17: case 3

Case 4: P=[I|0], P’ = Pb4, X=X4, depth of points at camera B should be positive but camera A should be in range of negative and positive

Text, chat or text message

Description automatically generated

Fig 18: case 4

In conclusion, the camera position is case 3. How we calculate it? We calculate following this equation Fig 19.

Text, letter

Description automatically generated

Fig 19: Calculate depth of point

Text

Description automatically generated

Fig 20: Calculate depth of point

A picture containing diagram

Description automatically generated

Fig 21: plot 3D points

**My image dataset: (Camera position: Case1)**

A picture containing toilet

Description automatically generated

Fig 22: My image dataset for two view

Diagram

Description automatically generated

Fig 23: Matching Keypoints

A picture containing chart

Description automatically generated

Fig 24: Plot 3D point from my images