Home Work 4

Name : Sai Teja Chava
Course Name : Advanced Computer Vision(CSCI 677)
USC ID : 5551847788
Instructor Name : Prof. Ram Nevatia

Steps in Code:

1. Import all the required packages.
2. Read the 2 images from the paths sent as arguments while the program is run.
3. Convert the images to grayscale as SIFT features need gray scale images for calculation of descriptors and the specified key points.
4. Initialize the intrinsic matrix with the given values.
5. Initiate the SIFT detector
6. Find the key points and descriptors with SIFT for both images.
7. Find the orientation of all the key points.
8. Display the images with key points laid on top of them.
9. Use flann to perform feature matching.
10. Store all the good matches as per the Lowe's test ratio and sort these matches.
11. Compute the essential matrix using findEssentialMat function in OpenCV and print it.
12. Recover relative camera rotation and translation from an estimated essential matrix using recoverpose function in OpenCV.
13. Calculate the projection matrices for both cameras and print the camera matrices.
14. Undistort points using undistortPoints function in OpenCV.
15. Do the triangulation using trangulatePoints function in OpenCV.
16. Output the 3D point cloud using functions in matplot lib library

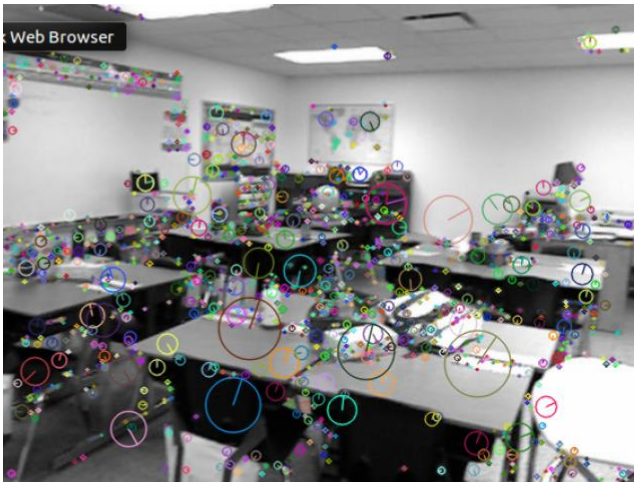
Discussion:

- To find structure from motion, the correspondence between images needs to be found.
- For that, used SIFT and filtered out bad matches.
- RANSAC is used to filter out the outlier correspondences.
- Essential matrix is generated and used to get the rotation and translation between the two cameras.
- Result is validated by finding the rank of the essential matrix and verifying it, which should be 2.
- Since the intrinsic parameters for both the cameras are the same, we need to generate the extrinsic matrix of the second camera with respect to the first camera.
- We can generate the camera matrix with these matrices.
- With these parameters, we can triangulate points for visualization.
- We can generate a point cloud with a scatter plot, by appropriate rotation of this point cloud, we can visualize.

Results:

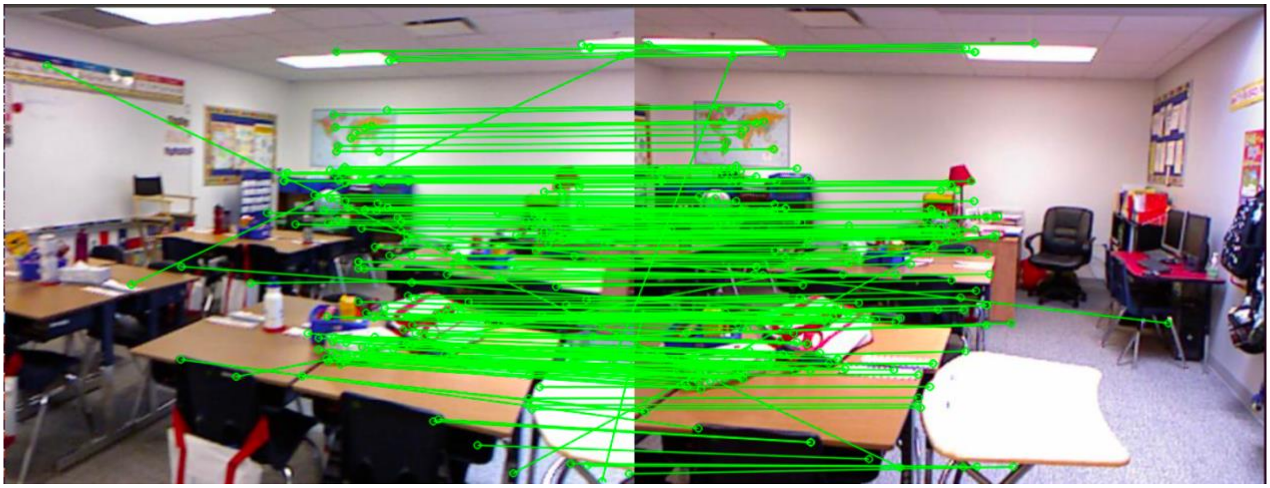
a1.png and a2.png:

a1.png with SIFT key points

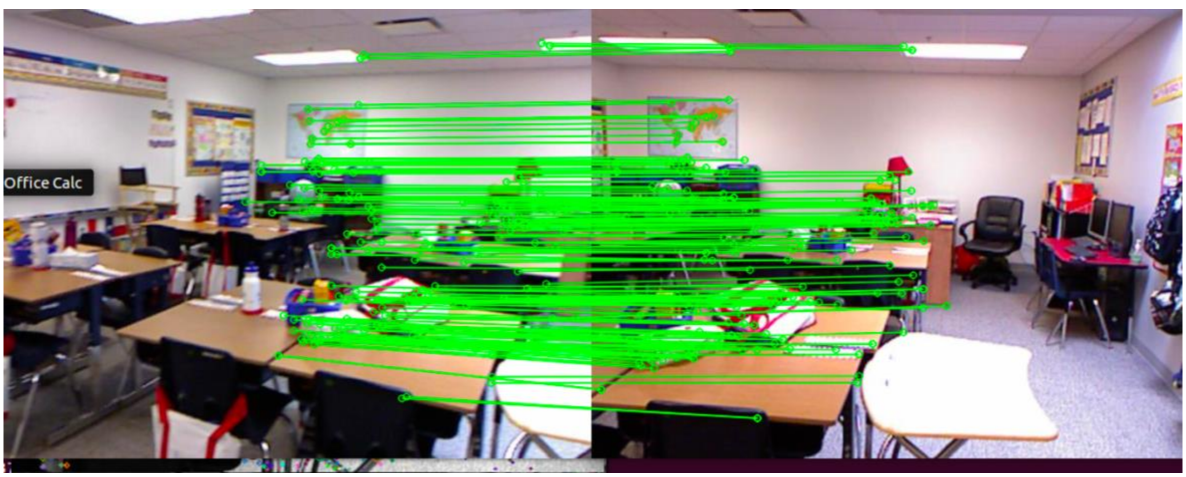


a2.png with SIFT Key points

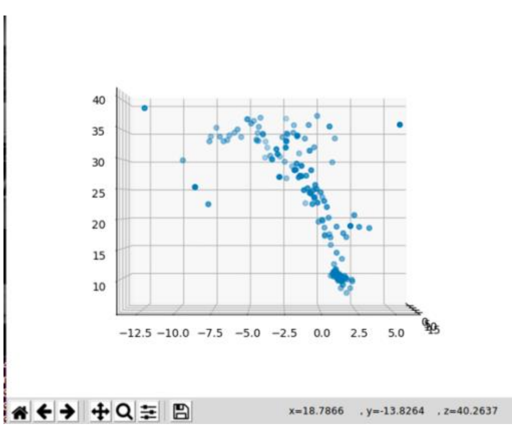
Matches between a1.png and a2.png



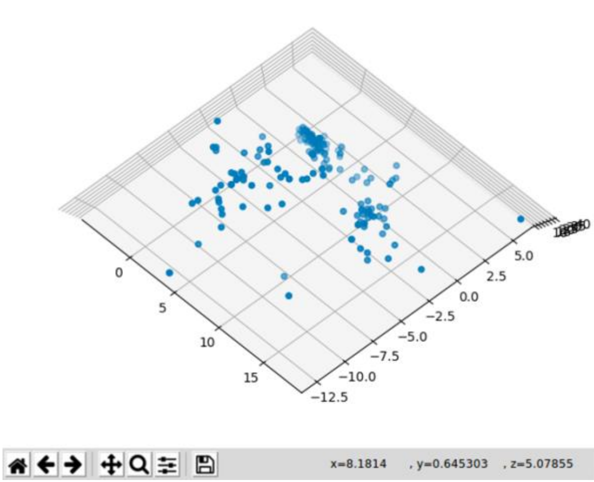
Inlier matches are:



Points are(Front View) :



Top View:

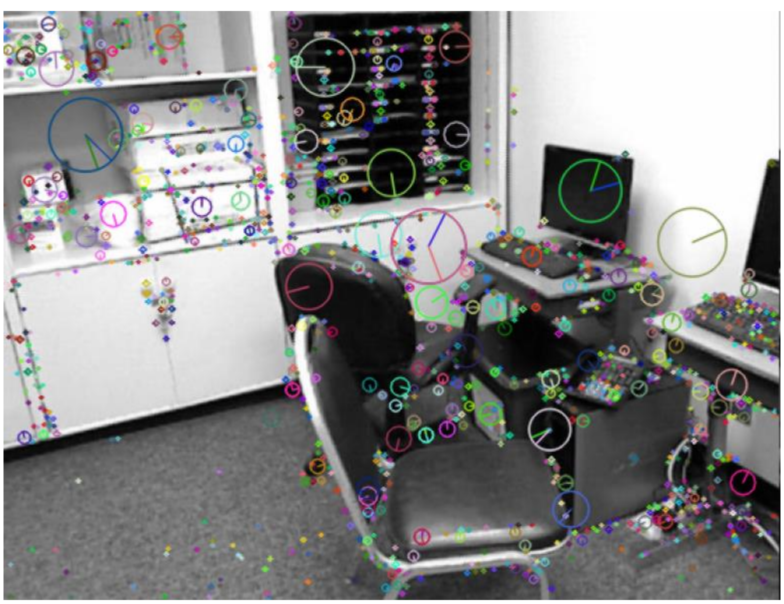


The features found in image1 : 1400
The features found in image2 are :1804
No of matches before RANSAC :202
No of matches after RANSAC :178
Essential matrix is :
[[0.04791642 -0.66015955 0.03924212]
[0.527688 0.02954776 -0.4669734]
[-0.05857102 0.24304144 0.02149146]]
Essential Matrix rank is : 2
Rotation Matrix is :
[[0.93193754 0.04626376 -0.35965552]
[-0.04468172 0.99892035 0.01271561]
[0.35985549 0.00421988 0.93299851]]
Translation Matrix is :
[[-0.34742896]
[-0.07222394]
[-0.93492076]]
Camrera Matrix for the first camera is :
[[518.86 0. 285.58 0.]
[0. 519.47 213.74 0.]
[0. 0. 1. 0.]]
Camrera Matrix for the second camera is :
[[5.86312640e+02 2.52095278e+01 7.98348516e+01 -4.47261660e+02]
[5.37046974e+01 5.19811108e+02 2.06024479e+02 -2.37348133e+02]
[3.59855488e-01 4.21987508e-03 9.32998510e-01 -9.34920756e-01]]
Avg euclidean distance or error w.r.t camera 1 : 4.259213568549116e-05

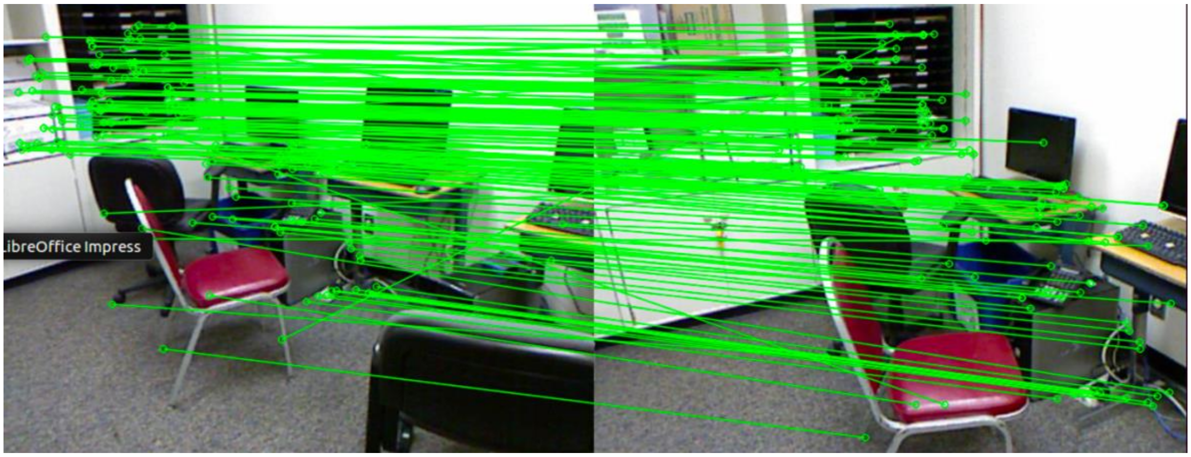
b1.png and b2.png:

b1.png with SIFT Key points

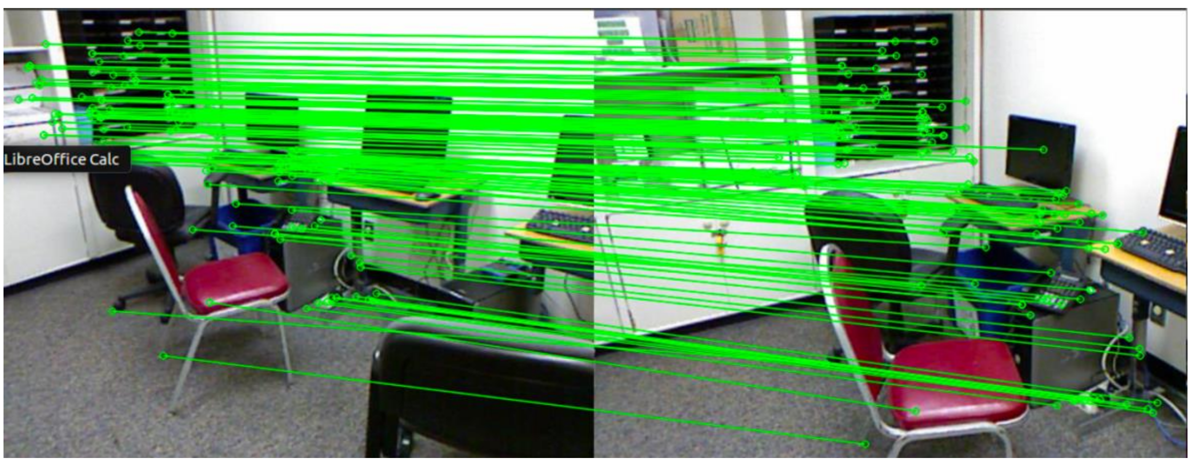
b2.png with SIFT Key points



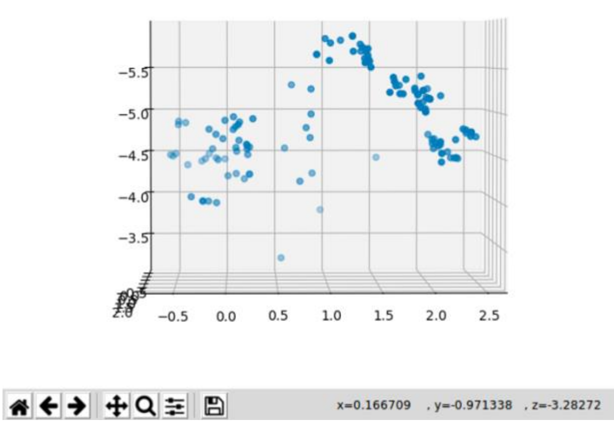
Matches between b1.png and b2.png



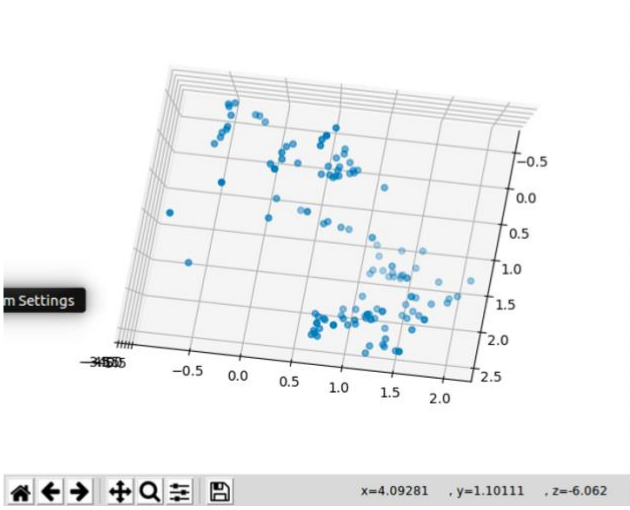
Inlier matches are:



Points are:



Top View:



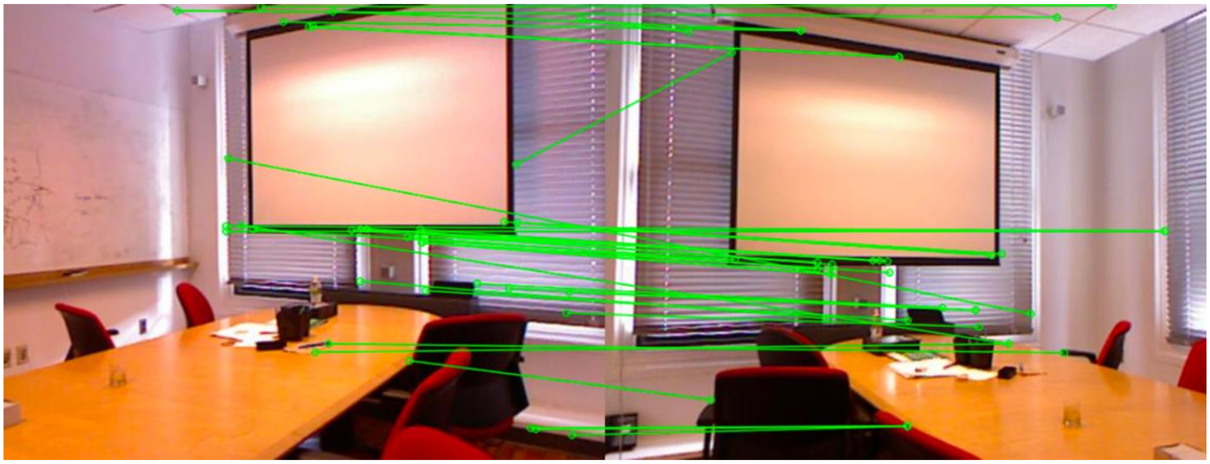
The features found in image1 : 1156
The features found in image2 are :1519
No of matches before RANSAC :167
No of matches after RANSAC :131
Essential matrix is :
[[0.03190652 -0.59099912 -0.28284293]
 [0.63757402 -0.06483951 -0.06487106]
 [0.30219647 0.2333719   0.0948878]]
Essential Matrix rank is :2
Rotation Matrix is :
[[0.94348168 -0.10024847 0.31589961]
 [0.10221538 0.99470811 0.01038189]
 [-0.31526868 0.02249468 0.94873582]]
Translation Matrix is :
[[0.37336816]
 [-0.4125214]
 [0.83091655]]
Camrera Matrix for the first camera is :
[[518.86   0.   285.58   0.]
 [0.   519.47 213.74   0.]
 [0.   0.   1.   0.]]
Camrera Matrix for the second camera is :
[[3.99500476e+02 -4.55908918e+01 4.34847647e+02 4.31018952e+02]
 [-1.42877056e+01 5.21529037e+02 2.08175872e+02 -3.66923854e+01]
 [-3.15268676e-01 2.24946779e-02 9.48735818e-01 8.30916551e-01]]
Avg euclidean distance or error w.r.t camera 1 : 4.963136631548642e-05

c1.png and c2.png:

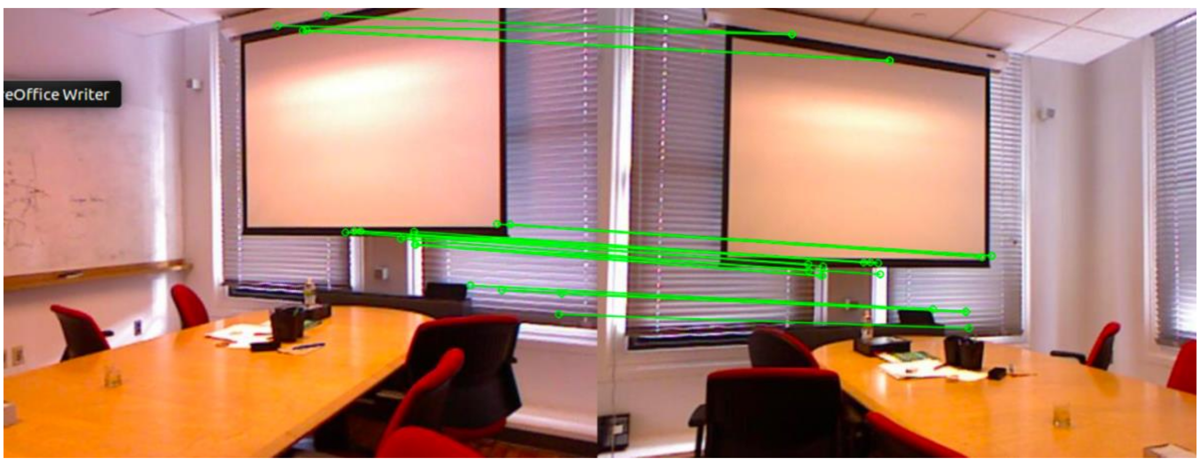
c1.png with SIFT Key points

c2.png with SIFT Key points

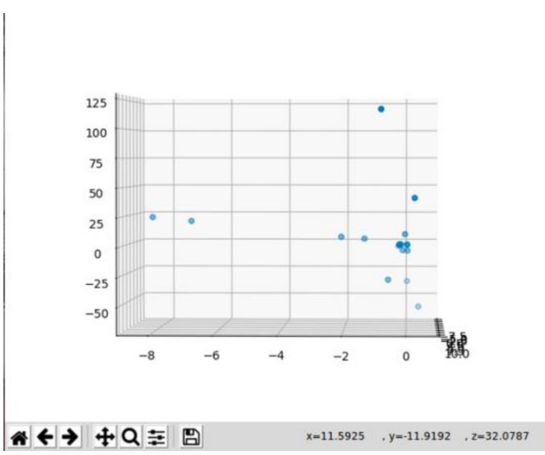
Matches between c1.png and c2.png



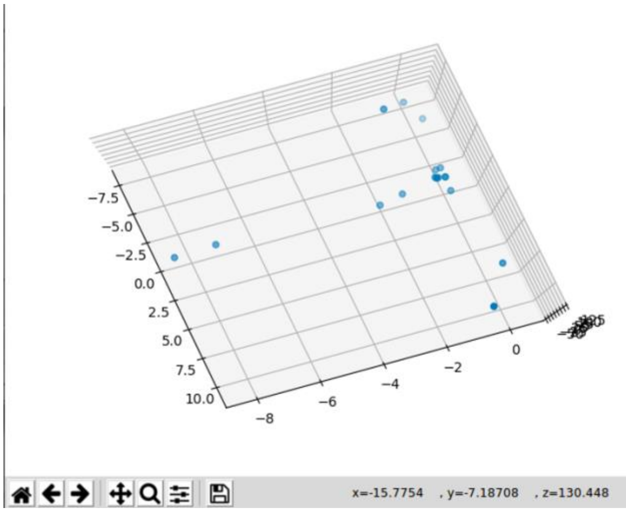
Inlier matches are:



Points are:



Top View:



The features found in image1 : 729
The features found in image2 are :1278
No of matches before RANSAC :36
No of matches after RANSAC :19
Essential matrix is :
[[-0.00537345 -0.29458701 -0.00866702]
 [-0.31099211 0.04068331 -0.63369749]
 [0.01963764 0.64152752 0.03503678]]
Essential Matrix rank is :2
Rotation Matrix is :
[[0.99949304 -0.01744784 -0.02663146]
 [-0.01888806 -0.99831746 -0.05482241]
 [-0.02563012 0.05529764 -0.99814091]]
Translation Matrix is :
[[0.9089714]
 [0.01060849]
 [0.41672348]]
Camrera Matrix for the first camera is :
[[518.86   0.   285.58   0.]
 [0.   519.47 213.74   0.]
 [0.   0.   1.   0.]]
Camrera Matrix for the second camera is :
[[5.11277510e+02 6.73891169e+00 -2.98867079e+02 5.90636790e+02]
 [-1.52899626e+01 -5.06776652e+02 -2.41821236e+02 9.45812697e+01]
 [-2.56301178e-02 5.52976360e-02 -9.98140906e-01 4.16723476e-01]]
Avg euclidean distance or error w.r.t camera 1 : 0.00013545933564704398

Conclusion:

- Did back projection and computed avg Euclidean distance. The error was very minute which means the process worked well i.e 3D points are constructed well.
- This works well when disparity between cameras is acceptable, that is if the images have sufficient number of matches, we can easily reconstruct 3D from it.
- 3D reconstruction is not possible if we have insufficient number of matches between images.
- This fails, when there is huge amount of barrel or pin cushion distortion, which hinders the matching, resulting in failing of 3D reconstruction.
- Can be improved by using multiple cameras by improving the probability of reconstruction.

Code:

```
import cv2
import numpy as np
import argparse
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
from numpy.linalg import matrix_rank

ap = argparse.ArgumentParser()
ap.add_argument("-i","--image1",required = True, type = str, help = "Path to image 1")
ap.add_argument("-j","--image2",required = True, type = str, help = "Path to image 2")

args = vars(ap.parse_args())

image_1 = args["image1"]
image_2 = args["image2"]

MIN_MATCH_COUNT = 10

#TODO: Load Different Image Pairs
img1=cv2.imread(image_1)
img2=cv2.imread(image_2)

#Gray images
gray1 = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

#TODO: Replace K with given Intrinsic Matrix
K = np.array([[518.86, 0.0, 285.58],
        [0.0, 519.47, 213.74],
        [0.0,   0.0,   1.0]])

###############################
#1----SIFT feature matching---#
###############################

#detect sift features for both images
```

```
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

print('The features found in image1 : ' + str(des1.shape[0]))
print('The features found in image2 are :' + str(des2.shape[0]))

x = image_1.split(".")[0]
y = image_2.split(".")[0]

#Drawing keypoints on the images
img4 =
cv2.drawKeypoints(gray1,kp1,gray1,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOIN
TS)
img5 =
cv2.drawKeypoints(gray2,kp2,gray2,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOIN
TS)

#Displaying the key point descriptors
cv2.imshow('Image1 with SIFT keypoints',img4)
cv2.imwrite(x+"SIFT keypoints.png",img4)
cv2.waitKey(0)
cv2.imshow('Image2 with SIFT keypoints',img5)
cv2.imwrite(y+"SIFT keypoints.png",img5)
cv2.waitKey(0)

#use flann to perform feature matching
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)

flann = cv2.FlannBasedMatcher(index_params, search_params)

matches = flann.knnMatch(des1,des2,k=2)

# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
print("No of matches before RANSAC :"+str(len(good)))
if len(good)>MIN_MATCH_COUNT:
    p1 = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    p2 = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)

draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   flags = 2)

img_siftmatch = cv2.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)
cv2.imshow('Matches between ' + image_1 + ' and ' + image_2, img_siftmatch)
```

```
cv2.waitKey(0)
#s = x+"Macthes between "+ x +" and "+ y +".png"
#cv2.imwrite(s,img_siftmatch)
#cv2.waitKey(0)
#cv2.imwrite('../results/sift_match.png',img_siftmatch)

#########################
#2----essential matrix--#
#########################
E, mask = cv2.findEssentialMat(p1, p2, K, cv2.RANSAC, 0.999, 1.0);

matchesMask = mask.ravel().tolist()

draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                singlePointColor = None,
                matchesMask = matchesMask, # draw only inliers
                flags = 2)

print("No of matches after RANSAC :"+str(matchesMask.count(1)))
img_inliermatch = cv2.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)
cv2.imshow("Inlier matches are",img_inliermatch)
cv2.waitKey(0)
#s = x+"inlier_match between "+ x +" and "+ y +".png"
#cv2.imwrite(s,img_inliermatch)
print("Essential matrix is :")
print(E)

# Verify the rank is 2
print("Essential Matrix rank is :"+str(matrix_rank(E)))

####################
#3----recoverpose--#
####################

points, R, t, mask = cv2.recoverPose(E, p1, p2)
print("Rotation Matrix is :")
print(R)
print("Translation Matrix is :")
print(t)
# p1_tmp = np.expand_dims(np.squeeze(p1), 0)
p1_tmp = np.ones([3, p1.shape[0]])
p1_tmp[:2,:] = np.squeeze(p1).T
p2_tmp = np.ones([3, p2.shape[0]])
p2_tmp[:2,:] = np.squeeze(p2).T
#print((np.dot(R, p2_tmp) + t) - p1_tmp)

#######################
#4----triangulation---#
#######################

#calculate projection matrix for both camera
```

```
M_r = np.hstack((R, t))
M_l = np.hstack((np.eye(3, 3), np.zeros((3, 1))))

P_l = np.dot(K,  M_l)
P_r = np.dot(K,  M_r)

#Camera Matrixes
print("Camrera Matrix for the first camera is :")
print(P_l)
print("Camrera Matrix for the second camera is :")
print(P_r)

# undistort points
p1 = p1[np.asarray(matchesMask)==1,:,:]
p2 = p2[np.asarray(matchesMask)==1,:,:]
p1_un = cv2.undistortPoints(p1,K,None)
p2_un = cv2.undistortPoints(p2,K,None)
p1_un = np.squeeze(p1_un)
p2_un = np.squeeze(p2_un)

#triangulate points this requires points in normalized coordinate
point_4d_hom = cv2.triangulatePoints(M_l, M_r, p1_un.T, p2_un.T)
point_3d = point_4d_hom / np.tile(point_4d_hom[-1, :], (4, 1))
point_try = point_3d
point_3d = point_3d[:3, :].T

points_2d_perspective = np.dot(M_l,point_try)
point_2d = points_2d_perspective/np.tile(points_2d_perspective[-1:],(3,1))
point_2d = point_2d[:2,:].T

diff1 = np.linalg.norm(point_2d-p1_un)
diff2 = np.linalg.norm(point_2d-p2_un)
print("Avg euclidean distance or error w.r.t camera 1 :",diff1/point_2d.shape[0])
#print("Avg euclidean distance or error w.r.t camera 2 :",diff2/point_2d.shape[0])
##############################
#5----output 3D pointcloud--#
##############################
#TODO: Display 3D points
fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(point_3d[:,0],point_3d[:,1],point_3d[:,2])
plt.show()


cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(1)
```