Shashank N S Homework 3 CSCI677 | Fall-2020

***Objective:** In this assignment, we are going to implement a simplified version of **Structure from Motion** using functions from opencv.*

1. **Introduction:** Structure from Motion (SFM) helps us to recover 3D shape from one or more 2D images. There are many methods to recover 3D from multiple 2D images like Stereo, Silhouettes, Motion, Texture, Shading, Contours etc. But Structure from Motion makes following assumptions, a) The Camera Model Should be Orthographic, b) Camera Calibration is not needed. In short, we can say Structure from Motion is a combined problem of camera calibration and stereo where **"given only the patching points in 2D images, we need to find both camera parameters and scene properties".**

2. **Structure from Motion Pipeline:**
   a. Using SIFT, salient points in all the three images are found.
   b. Using **FLANN (Fast Library for Approximate Nearest Neighbors)** matches the SIFT keypoints between the pair of images. FLANN finds the nearest matching descriptors by finding the euclidean distance between the descriptor vectors in both the images and gets the nearest match in the corresponding image.
   c. Set the threshold to eliminate bad matches (0.7, 0.8, 0.9)
   d. Draw the matches and visualize the SIFT point.
   e. Using RANSAC find the essential matrix which provides inlier and outlier points.
   f. Estimate the pose i.e, find the Rotation and Translation matrix .
   g. Compute 3D points using triangulation.

3. **Summary of Program:**

2.1 Code Structure: The data required for the assignment is available in `./HW3_data` where the 3 images from different viewpoints and the intrinsic matrix of the camera is provided. `./sfm.py` will have the code which implements Structure from Motion pipeline. `./output` folder contains the saved images of SIFT matches, Epipolar lines and the point cloud in `.obj` format.

2.2 Source Listing.

1. **Detect and Match features**

```
#Compute SIFT descriptors
detector = cv2.xfeatures2d.SIFT_create()
kp1, descriptors1 = detector.detectAndCompute(img1, None)
kp2, descriptors2 = detector.detectAndCompute(img2, None)
```

```
#FLANN Based 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)
knn_matches = flann.knnMatch(descriptors1,descriptors2,k=2)

#Extract Good Matches using Ratio Test
ratio_thresh = 0.8
matches_good = []

for m,n in knn_matches:
    if m.distance < ratio_thresh * n.distance:
      matches_good.append(m)
p1  =  np.float32([  kp1[m.queryIdx].pt  for  m  in  matches_good
]).reshape(-1,1,2)
p2  =  np.float32([  kp2[m.trainIdx].pt  for  m  in  matches_good
]).reshape(-1,1,2)

return p1, p2, matches_good, kp1, kp2
```

2. **Finding Essential Matrix**

```
#Find Essential Matrix function returns the E matrix and the
mask which contains inlier and outlier points
E,   mask   =   cv2.findEssentialMat(p1,   p2,   self.intrinsic,
cv2.RANSAC, 0.999, 1.0);
print("Essential Matrix: ", E)
return E, mask
```

3. **Finding Pose (Rotation and Translation)**

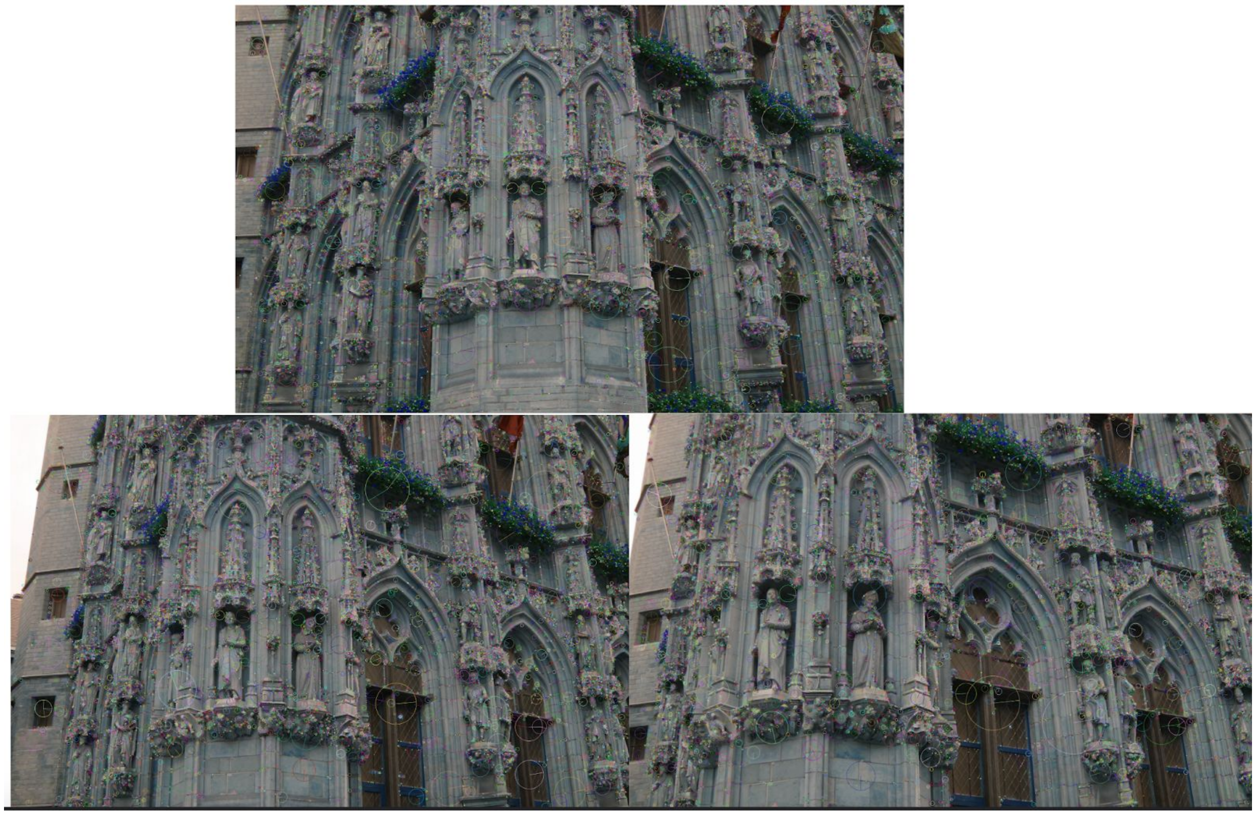
```
#Finding pose involves getting the rotation and translation
matrix using the matching points and Essential matrix
points, R, trans, mask = cv2.recoverPose(E, p1, p2)
print("Rotation", R)
print("Translation", trans)
return R, trans
```

4. **Triangulation:** *Given Camera Matrix and Matching points find the intersection points in 3D*

```
# Extrinsic Matrix for both the images
M_r = np.hstack((R, trans))
M_l = np.hstack((np.eye(3, 3), np.zeros((3, 1))))

#Find the Final Camera Matrix
P_l = np.dot(self.intrinsic,  M_l)
P_r = np.dot(self.intrinsic,  M_r)
print("Left Projection:", P_l)
print("Right Projection:",P_r)


  ind_range = (np.asarray(mask) == 1).reshape(-1,)
  p1_new = p1[ind_range,:,:]
  p2_new = p2[ind_range,:,:]
  p1_un = cv2.undistortPoints(p1_new,self.intrinsic,None)
  p2_un = cv2.undistortPoints(p2_new,self.intrinsic,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(P_l, P_r, p1_un.T, p2_un.T)
point_3d = point_4d_hom / np.tile(point_4d_hom[-1, :], (4, 1))
point_3d = point_3d[:3, :].T

print("Final Points in 3D: ", point_3d.shape)
return point_3d
```

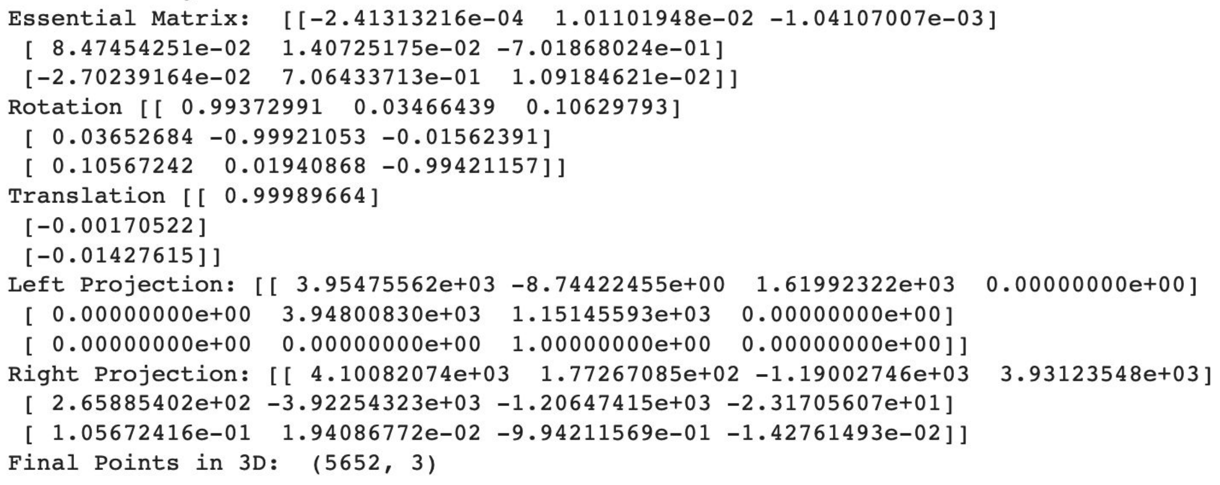
3. Results:

3.1 SIFT Keypoints in all three views.



3.2 Results for all combinations with different threshold values.

- **0-1 (0.7 Threshold)**



```
Essential Matrix:  [[-2.41313216e-04  1.01101948e-02 -1.04107007e-03]
 [ 8.47454251e-02  1.40725175e-02 -7.01868024e-01]
 [-2.70239164e-02  7.06433713e-01  1.09184621e-02]]
Rotation [[ 0.99372991  0.03466439  0.10629793]
 [ 0.03652684 -0.99921053 -0.01562391]
 [ 0.10567242  0.01940868 -0.99421157]]
Translation [[ 0.99989664]
 [-0.00170522]
 [-0.01427615]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00]
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 4.10082074e+03  1.77267085e+02 -1.19002746e+03  3.93123548e+03]
 [ 2.65885402e+02 -3.92254323e+03 -1.20647415e+03 -2.31705607e+01]
 [ 1.05672416e-01  1.94086772e-02 -9.94211569e-01 -1.42761493e-02]]
Final Points in 3D:  (5652, 3)
```

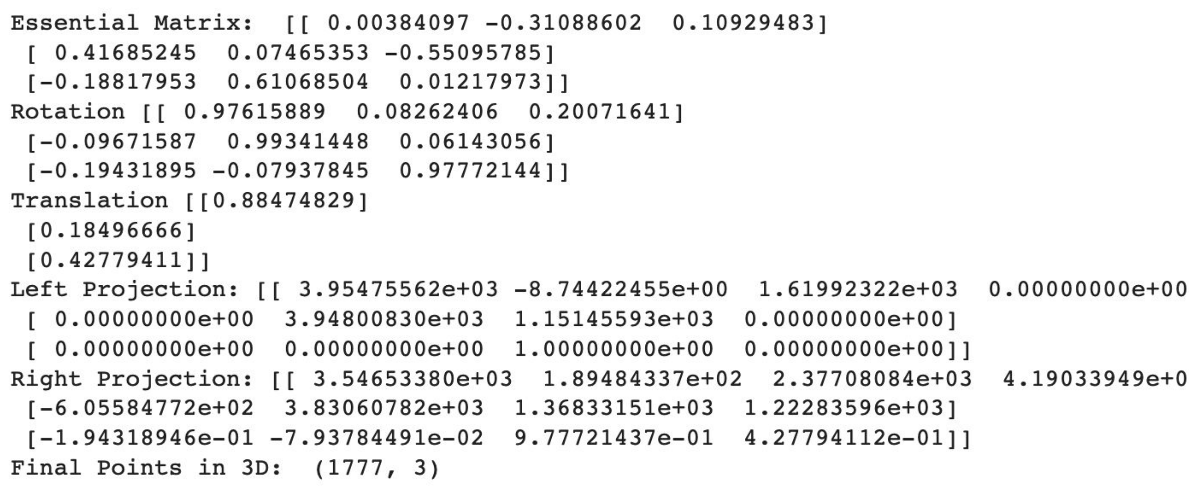
- **0-1 (0.8 Threshold)**

```
Essential Matrix:  [[-5.37288646e-04  1.41302708e-02  4.41142623e-05]
 [ 8.39430154e-02  1.41658861e-02 -7.01963580e-01]
 [-2.79062749e-02  7.06329994e-01  1.09204175e-02]]
Rotation [[ 0.99432887  0.03749761  0.09951897]
 [ 0.03922644 -0.99911056 -0.0154716 ]
 [ 0.0988503   0.01928763 -0.99491538]]
Translation [[ 9.99800025e-01]
 [-2.48248787e-04]
 [-1.99962179e-02]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00]
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 4.09211458e+03  1.88274825e+02 -1.21797804e+03  3.92157460e+03]
 [ 2.68688060e+02 -3.92228793e+03 -1.20668322e+03 -2.40048520e+01]
 [ 9.88503017e-02  1.92876348e-02 -9.94915376e-01 -1.99962179e-02]]
Final Points in 3D:  (7283, 3)
```

- **0-1 (0.9 Threshold)**

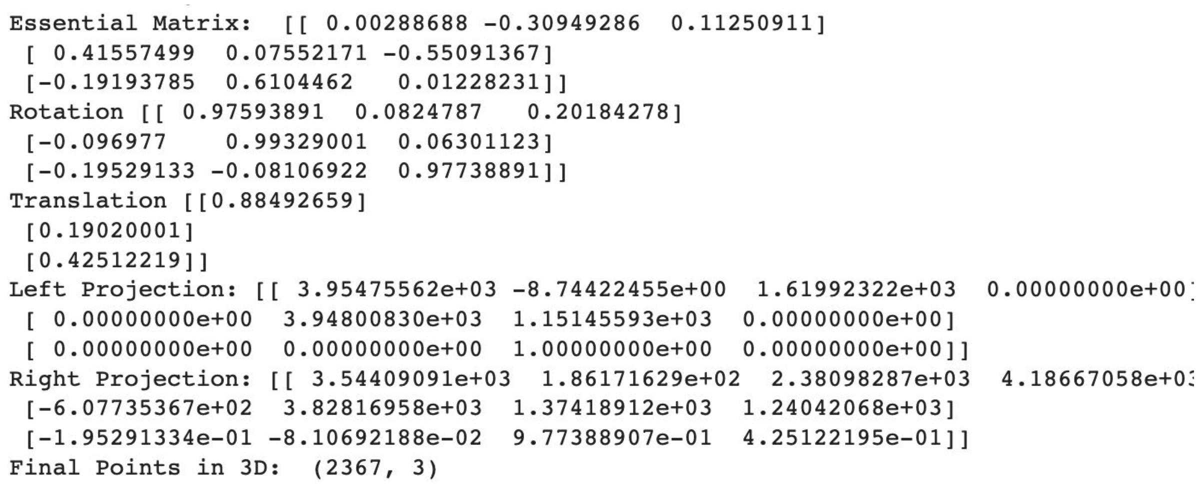
```
Essential Matrix:  [[ 0.00114117 -0.0173887  -0.00336613]
 [-0.08459615 -0.01473677  0.70186652]
 [ 0.03108011 -0.70612158 -0.01100244]]
Rotation [[ 0.99429297  0.0466826   0.09592821]
 [ 0.04835111 -0.99871562 -0.01514181]
 [ 0.09509815  0.01969363 -0.99527308]]
Translation [[ 0.99968495]
 [ 0.00440711]
 [-0.02470986]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00]
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 4.08581461e+03  2.25253439e+02 -1.23276092e+03  3.91344306e+03
 [ 3.00391911e+02 -3.92026122e+03 -1.20579308e+03 -1.10530127e+01]
 [ 9.50981469e-02  1.96936298e-02 -9.95273080e-01 -2.47098615e-02]]
Final Points in 3D:  (7925, 3)
```

- **0-2 (0.7 Threshold)**



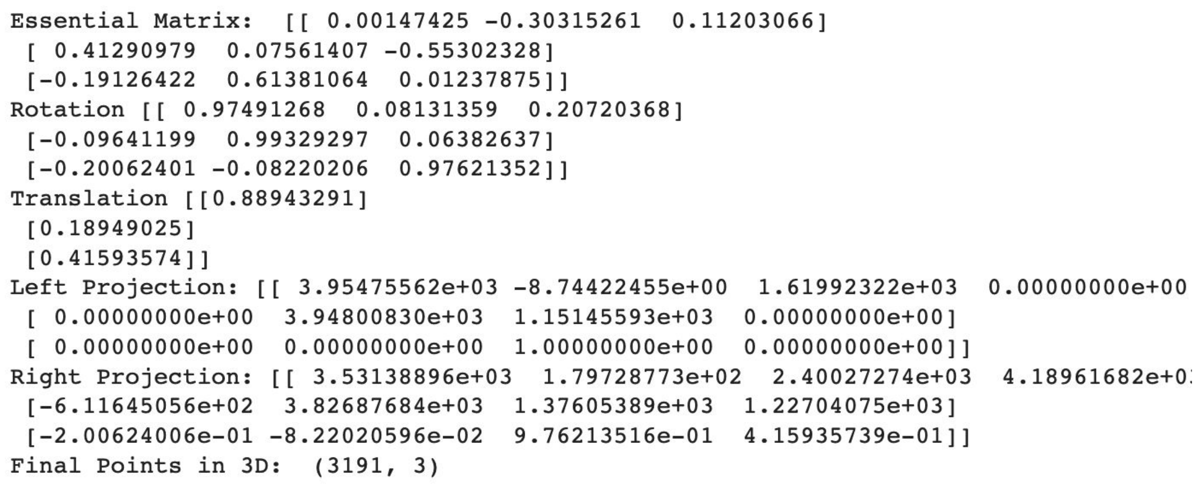
```
Essential Matrix:  [[ 0.00384097 -0.31088602  0.10929483]
 [ 0.41685245  0.07465353 -0.55095785]
 [-0.18817953  0.61068504  0.01217973]]
Rotation [[ 0.97615889  0.08262406  0.20071641]
 [-0.09671587  0.99341448  0.06143056]
 [-0.19431895 -0.07937845  0.97772144]]
Translation [[0.88474829]
 [0.18496666]
 [0.42779411]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 3.54653380e+03  1.89484337e+02  2.37708084e+03  4.19033949e+0
 [-6.05584772e+02  3.83060782e+03  1.36833151e+03  1.22283596e+03]
 [-1.94318946e-01 -7.93784491e-02  9.77721437e-01  4.27794112e-01]]
Final Points in 3D:  (1777, 3)
```

- **0-2 (0.8 Threshold)**



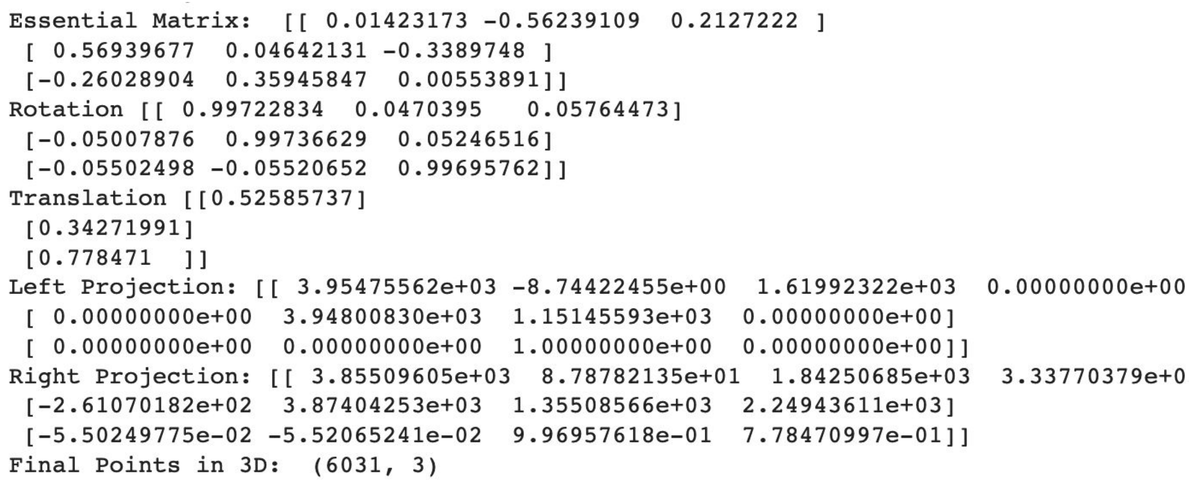
```
Essential Matrix:  [[ 0.00288688 -0.30949286  0.11250911]
 [ 0.41557499  0.07552171 -0.55091367]
 [-0.19193785  0.6104462   0.01228231]]
Rotation [[ 0.97593891  0.0824787   0.20184278]
 [-0.096977    0.99329001  0.06301123]
 [-0.19529133 -0.08106922  0.97738891]]
Translation [[0.88492659]
 [0.19020001]
 [0.42512219]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 3.54409091e+03  1.86171629e+02  2.38098287e+03  4.18667058e+0
 [-6.07735367e+02  3.82816958e+03  1.37418912e+03  1.24042068e+03]
 [-1.95291334e-01 -8.10692188e-02  9.77388907e-01  4.25122195e-01]]
Final Points in 3D:  (2367, 3)
```

- **0-2 (0.9 Threshold)**



```
Essential Matrix:  [[ 0.00147425 -0.30315261  0.11203066]
 [ 0.41290979  0.07561407 -0.55302328]
 [-0.19126422  0.61381064  0.01237875]]
Rotation [[ 0.97491268  0.08131359  0.20720368]
 [-0.09641199  0.99329297  0.06382637]
 [-0.20062401 -0.08220206  0.97621352]]
Translation [[0.88943291]
 [0.18949025]
 [0.41593574]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 3.53138896e+03  1.79728773e+02  2.40027274e+03  4.18961682e+0
 [-6.11645056e+02  3.82687684e+03  1.37605389e+03  1.22704075e+03]
 [-2.00624006e-01 -8.22020596e-02  9.76213516e-01  4.15935739e-01]]
Final Points in 3D:  (3191, 3)
```

- **1-2 (0.7 Threshold)**



```
Essential Matrix:  [[ 0.01423173 -0.56239109  0.2127222 ]
 [ 0.56939677  0.04642131 -0.3389748 ]
 [-0.26028904  0.35945847  0.00553891]]
Rotation [[ 0.99722834  0.0470395   0.05764473]
 [-0.05007876  0.99736629  0.05246516]
 [-0.05502498 -0.05520652  0.99695762]]
Translation [[0.52585737]
 [0.34271991]
 [0.778471  ]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 3.85509605e+03  8.78782135e+01  1.84250685e+03  3.33770379e+0
 [-2.61070182e+02  3.87404253e+03  1.35508566e+03  2.24943611e+03]
 [-5.50249775e-02 -5.52065241e-02  9.96957618e-01  7.78470997e-01]]
Final Points in 3D:  (6031, 3)
```

- **1-2 (0.8 Threshold)**

```
Essential Matrix:  [[ 0.01246772 -0.5531874   0.21649732]
 [ 0.5615045   0.04674803 -0.34906649]
 [-0.2646476   0.37090172  0.00553412]]
Rotation [[ 0.99705663  0.04636763  0.06105835]
 [-0.04966674  0.99732273  0.05367106]
 [-0.05840628 -0.05654565  0.99669017]]
Translation [[0.54213867]
 [0.34836734]
 [0.7646737 ]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00]
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 3.84893592e+03  8.30522116e+01  1.85556307e+03  3.37969241e+03]
 [-2.63336973e+02  3.87232860e+03  1.35953859e+03  2.25584520e+03]
 [-5.84062762e-02 -5.65456535e-02  9.96690170e-01  7.64673698e-01]]
Final Points in 3D:  (6205, 3)
```

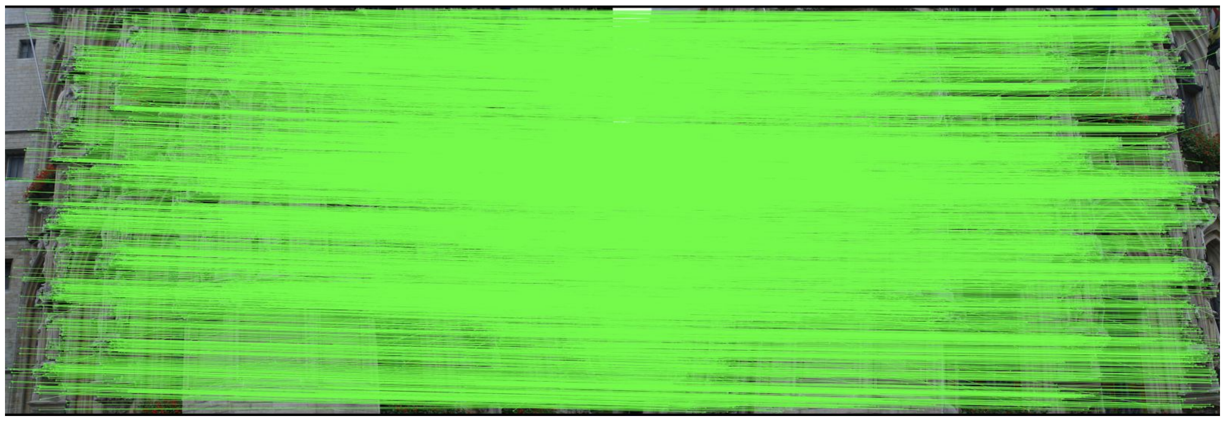
- **1-2 (0.9 Threshold)**

```
Essential Matrix:  [[ 0.01202906 -0.55364245  0.22006419]
 [ 0.56113564  0.04704907 -0.34634284]
 [-0.26897288  0.36808545  0.0057957 ]]
Rotation [[ 0.99707334  0.04595025  0.06110106]
 [-0.0493589   0.9972379   0.05550015]
 [-0.05838204 -0.0583536   0.99658737]]
Translation [[0.53834392]
 [0.35485156]
 [0.76437307]]
Left Projection: [[ 3.95475562e+03 -8.74422455e+00  1.61992322e+03  0.00000000e+00]
 [ 0.00000000e+00  3.94800830e+03  1.15145593e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Right Projection: [[ 3.84903855e+03  7.84735645e+01  1.85554947e+03  3.36414142e+03]
 [-2.62093707e+02  3.86991191e+03  1.36664151e+03  2.28109882e+03]
 [-5.83820446e-02 -5.83536039e-02  9.96587374e-01  7.64373074e-01]]
Final Points in 3D:  (7447, 3)
```

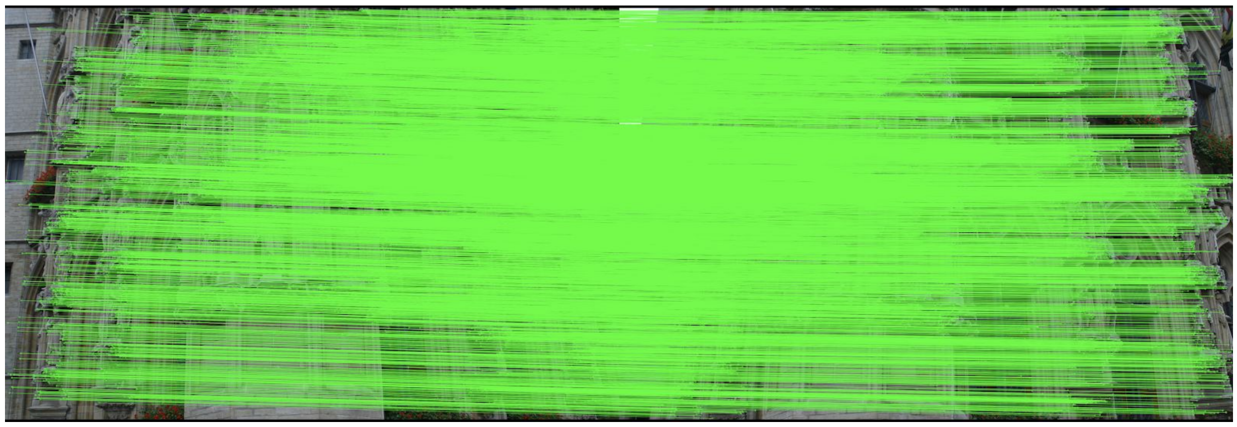
3.3 Visual Results:

- **0-1 : Visualizations for optimal threshold value (0.8)**

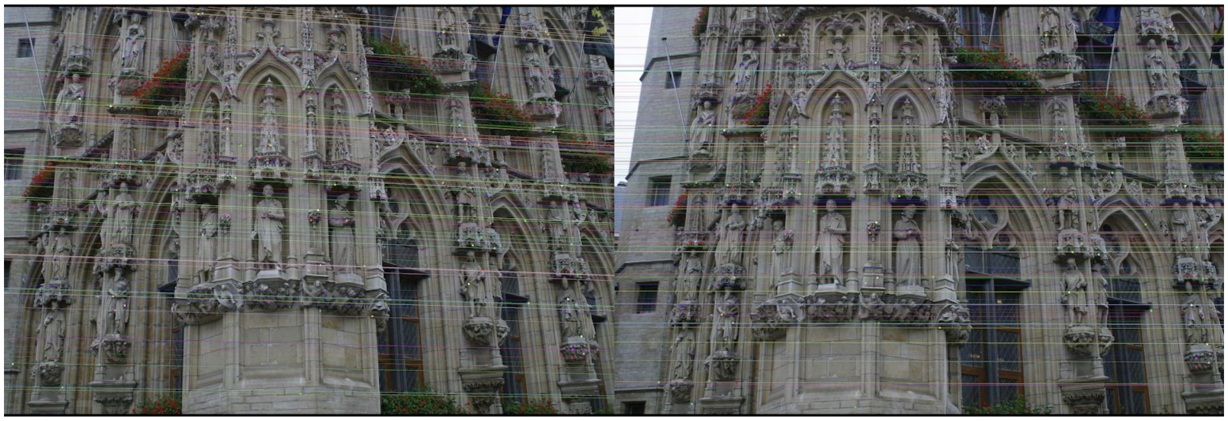
  Matched SIFT Features:



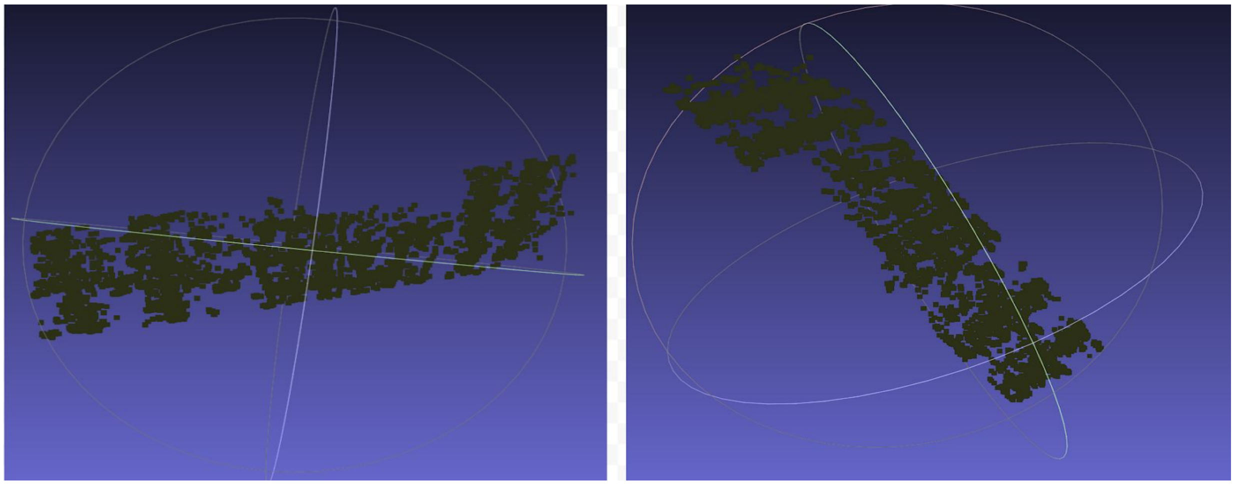
  Pruned Features:

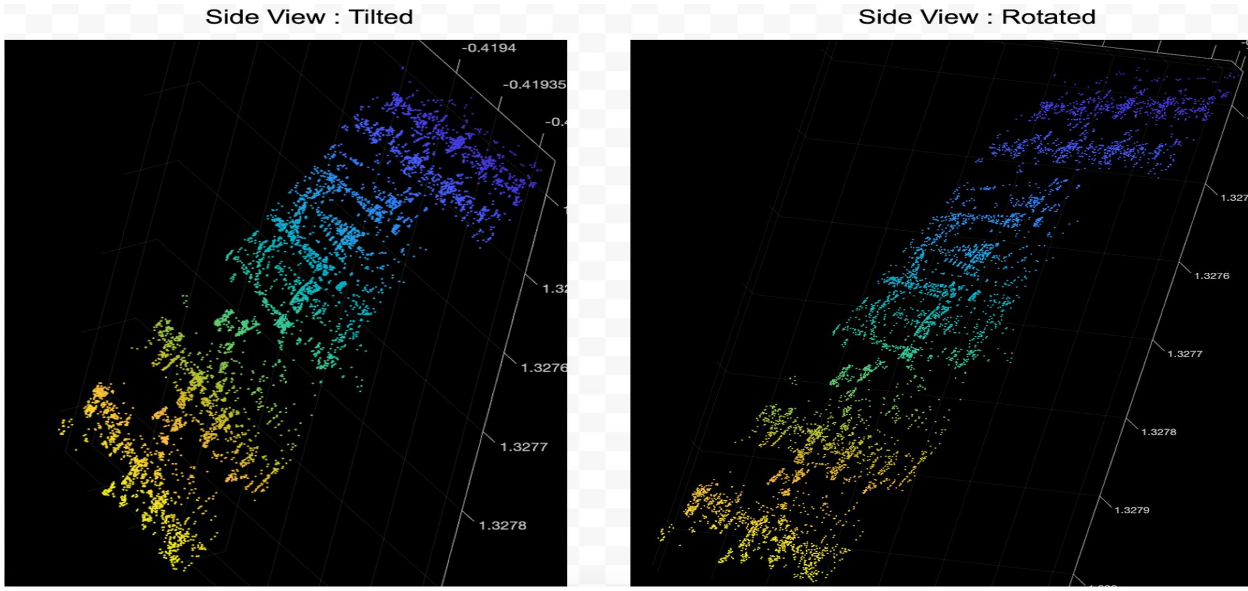


Epipolar lines:



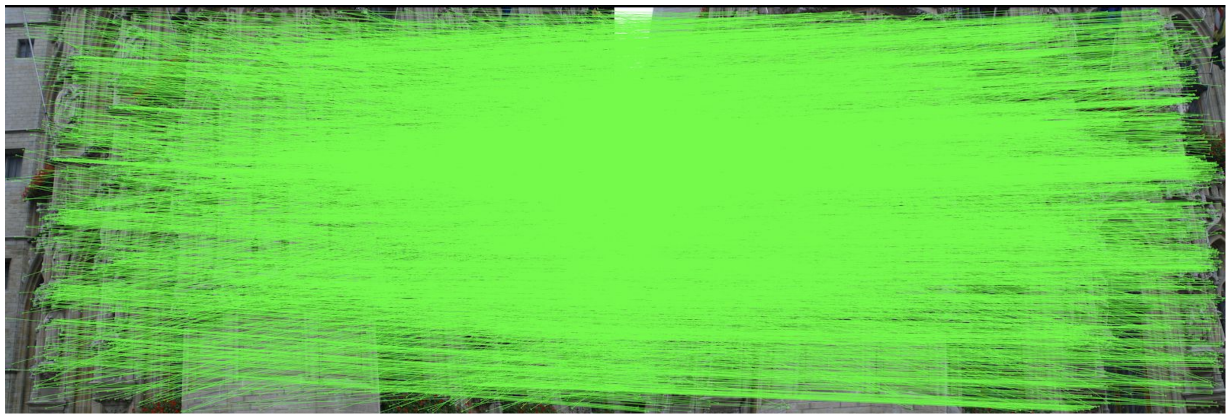
Point Cloud: (Both Point Cloud and Matlab Visualizations are attached)



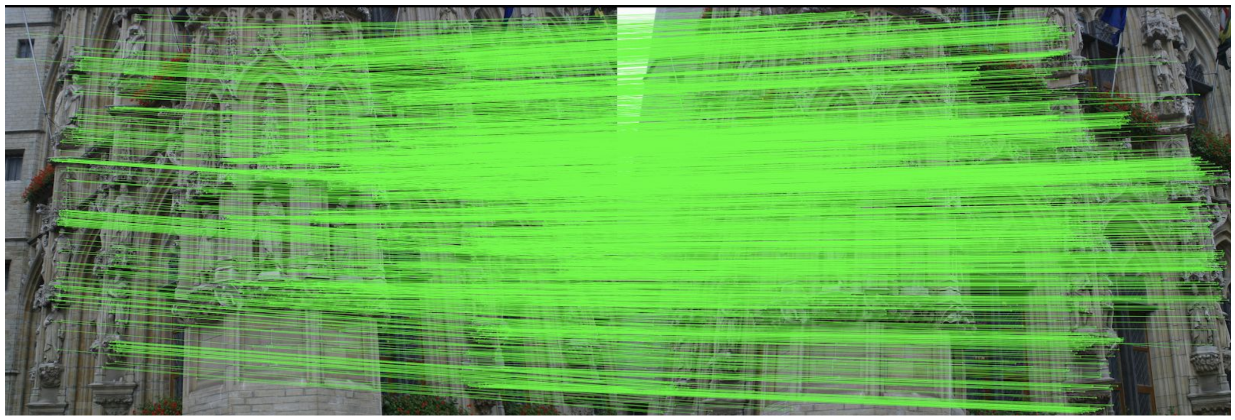


- **0-2 : Visualizations for optimal threshold value (0.9)**

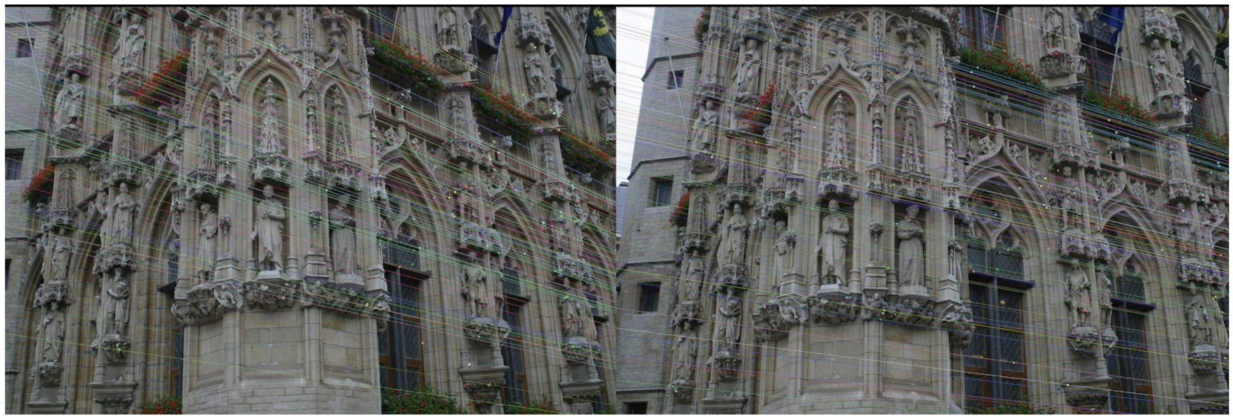
Matched SIFT Features:



Pruned Features:

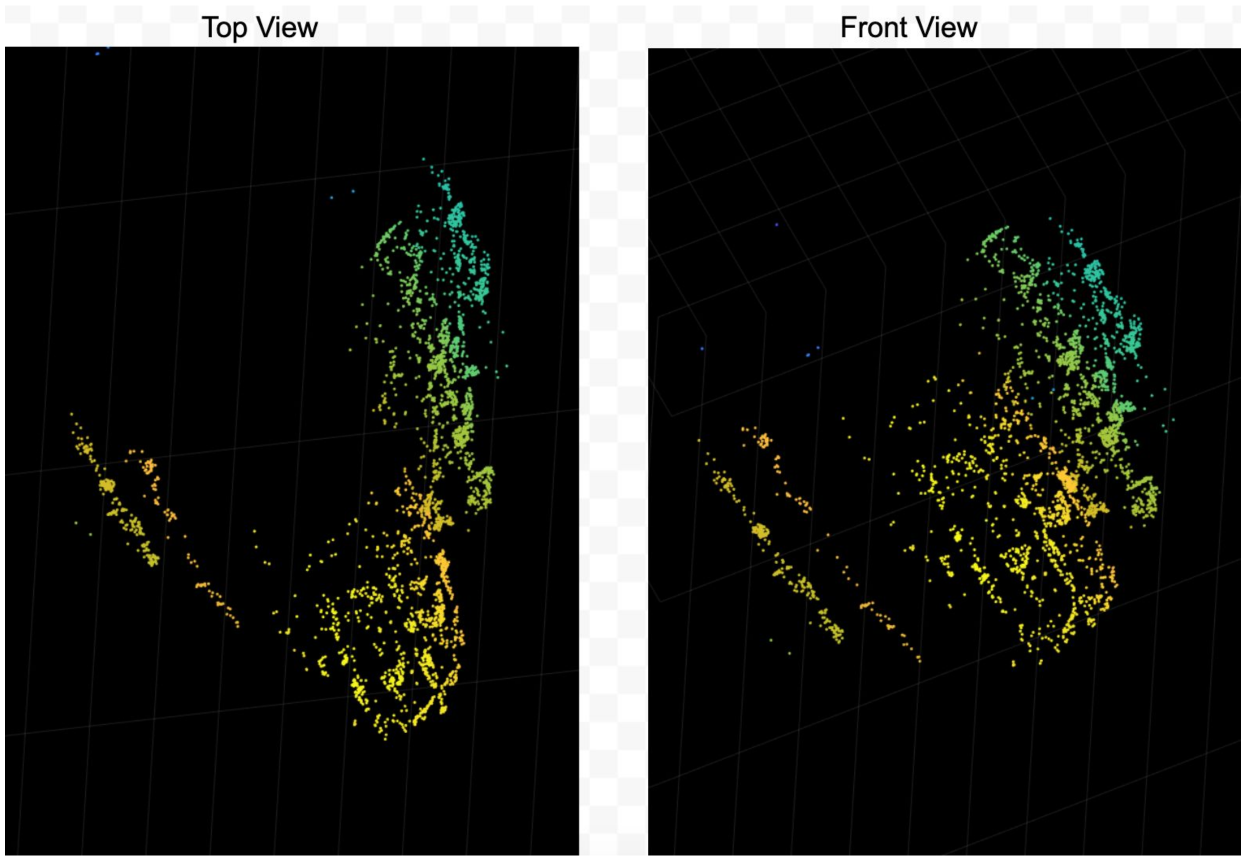


Epipolar lines:



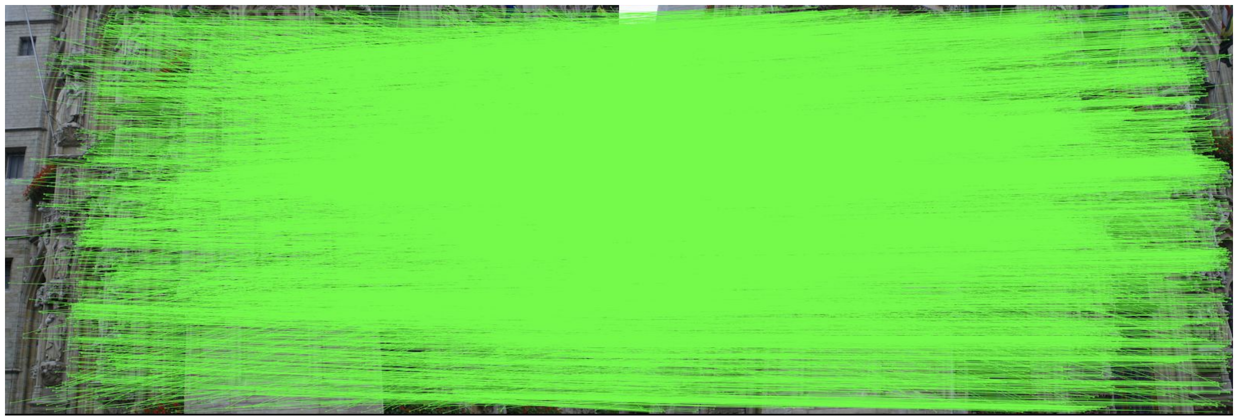
Point Cloud: (Matlab Visualization)

*I got better visualization in **matlab** compared to meshlab. I wrote 3D points to a csv file and converted it to **pointCloud** object in matlab to visualize it. The reconstruction of the 3D scene (building) can be clearly observed in this.*

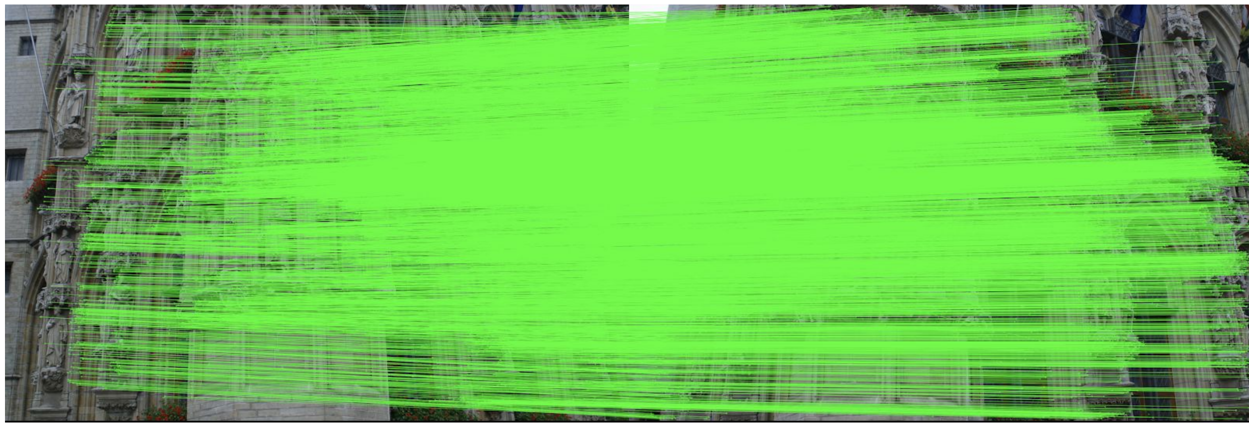


- **1-2 : Visualizations for optimal threshold value (0.8)**

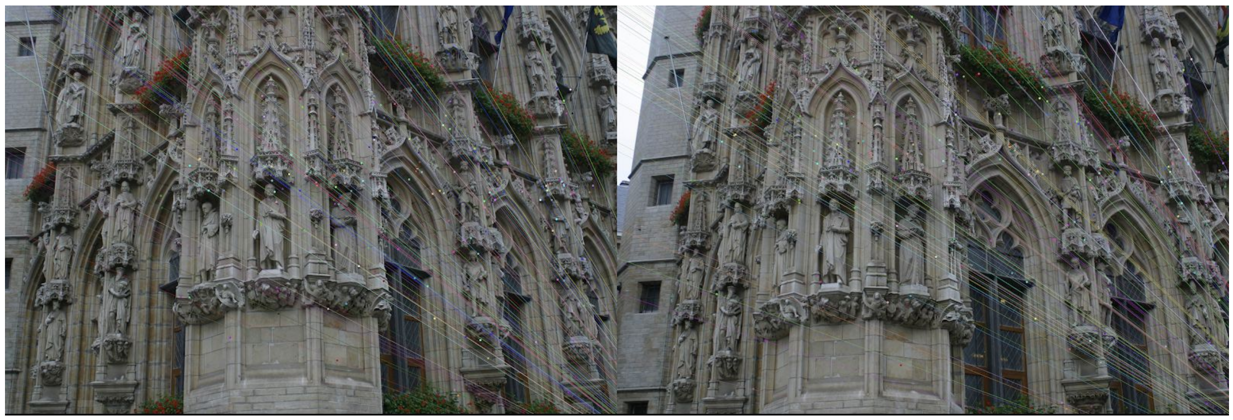
  Matched SIFT Features:



  Pruned Features:

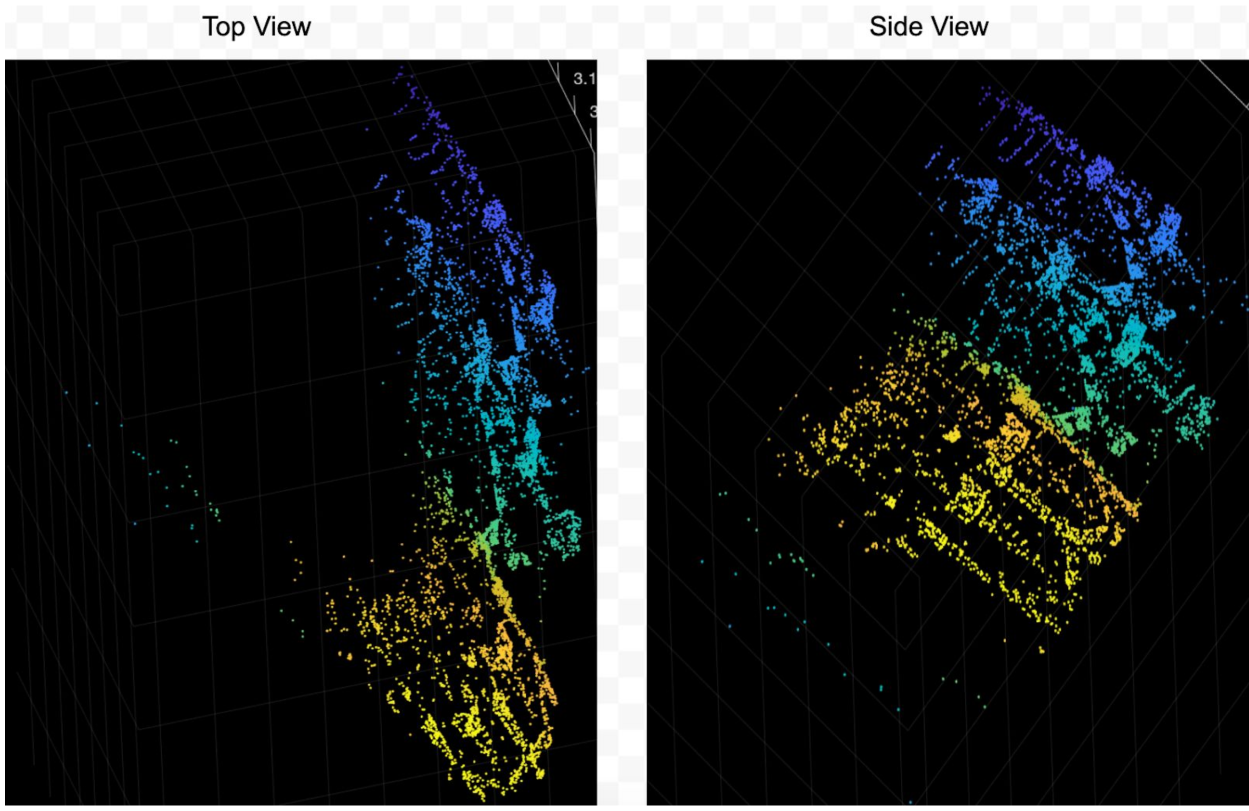


Epipolar lines:



Point Cloud: (Matlab Visualization)

*I got better visualization in **matlab** compared to meshlab. I wrote 3D points to a csv file and converted it to **pointCloud** object in matlab to visualize it. The reconstruction of the 3D scene (building) can be clearly observed in this.*



4. Conclusion/Qualitative Analysis: The 3D scene has been reconstructed using the given images. Although the number of images are very less for the entire scene to be reconstructed clearly, the obtained point clouds are accurately representing the structure. The method worked very well using the image combination 0-1 and 1-2 because the views are not distorted and we got more SIFT keypoints. However, for 0-2 combinations the number of SIFT points was less (around 2500) because the field of view (FoV) was completely different and we got sparse point clouds.

As we increase the threshold the number of points to compute the essential matrix differs. So, an optimal threshold is chosen by visualizing the point clouds in the meshlab.

The performance of SFM can be improved by,

1. Employing **Bundle Adjustment** after Triangulation will simultaneously refine the 3D coordinates obtained from all pairs of images.
2. Using **more images** of the structure we can get more points to reconstruct the entire scene completely with dense point clouds.
3. **Mean Normalization** of input images. Mean normalization scales all the images in between 0-1 values. Using scaled values the essential matrix computation will be accurate. We can then un-normalize the essential matrix to find pose.