CSCI677 HW-3

Nisha Tiwari| USC ID- 7888495181 | email- nishatiw@usc.edu

Table of Contents

Description of Programs

For writing code for this task, I have used the provided sfm_student.py as the starting point. Following is the description of the code added to this file.

1. **Detect Feature Points and Match them:** This step extracts SIFT features from the pair of images, matches the points using Brute Force Matcher and retrieves the good matches based on the ratio test.

```python
def detect_and_match_feature(self, img1, img2):
    """
    img1, img2: are input images
    The following outputs are needed:
    kp1, kp2: keypoints (here sift keypoints) of the two images
    matches_good: matches which pass the ratio test
    p1, p2: only the 2d points in the respective images
    pass ratio test. These points should correspond to each other.

    Steps:
    1. Compute sift descriptors.
    2. Match sift across two images.
    3. Use ratio test to get good matches.
    4. Store points retrieved from the good matches.
    """
    sift = cv2.xfeatures2d.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    #Either use FLANN matcher or BF Matcher
    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)
    bf = cv2.BFMatcher()
    # Match descriptors.
    matches = bf.knnMatch(des1, des2, k=2)

    # store all the good matches as per the ratio test.
    matches_good = []
    matches_not_good=[]
    for m, n in matches:
        if m.distance < self.t * n.distance:
            matches_good.append(m)
        else:
            matches_not_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, matches_not_good
```

2. **Compute Essential Matrix**: Given the points from the previous step, this method estimates the essential matrix. The parameters are fixed here as per the guidelines. This method also returns a mask that has 1 for all the inliers and 0 otherwise.

```python
def compute_essential(self, p1, p2):
    """
    p1, p2: only the 2d points in the respective images
    pass ratio test. These points should correspond to each other.
    Outputs:
    Essential Matrix (E), and corresponding (mask)
    used in its computation. The mask contains the inlier matches
    to compute E
    """
    E, mask = cv2.findEssentialMat( p1, p2, self.intrinsic, method= cv2.RANSAC, prob=0.999, threshold=1.0)

    return E, mask
```

3. **Recover Rotation and Translation from Essential Matrix:** Once the essential matrix is computed, we can use the following method to decompose it and get rotation and translation.

```python
def compute_pose(self, p1, p2, E):
    """
    p1, p2: only the 2d points in the respective images
    pass ratio test. These points should correspond to each other.
    E: Essential matrix
    Outputs:
    R, trans: Rotation, Translation vectors

    Hint: recoverPose
    """
    retval, R, t, mask = cv2.recoverPose(E, p1, p2, self.intrinsic)
    return R, t
```

4. **Apply Triangulation to the Points:** Here, in this method, we start with removing outliers using the mask. We create the projection matrix for both the cameras. For the first camera, it is [Id 0] and for the second camera, [R trans]. Before applying the triangulation, we call undistortpoints method to normalize the points

```python
def triangulate(self, p1, p2, R, trans, mask):
    """
    p1,p2: Points in the two images which correspond to each other
    R, trans: Rotation and translation matrix.
    mask: is obtained during computation of Essential matrix

    Outputs:
    point_3d: should be of shape (NumPoints, 3). The last dimension
    refers to (x,y,z) co-ordinates

    Hint: triangulatePoints
    """
    #Remove outliers from the given points using mask
    p1 = p1[mask.ravel() == 1]
    p2 = p2[mask.ravel() == 1]

    #Combine Rotation and Translation matrix
    rotation=np.append(np.array(R), trans, axis=1)

    #Rotation and Translation matrix for first camera will be an [Id 0] matrix
    rotationFirst=np.pad(np.identity(3), [[0,0], (0, 1)], mode='constant')

    #Normalise points
    p1_n=cv2.undistortPoints(p1, self.intrinsic, None)
    p2_n = cv2.undistortPoints(p2, self.intrinsic, None)

    #Apply triangulation
    pts4D = cv2.triangulatePoints(rotationFirst, rotation, p1_n, p2_n).T

    # convert from homogeneous coordinates to 3D
    point_3d = pts4D[:, :3] / np.repeat(pts4D[:, 3], 3).reshape(-1, 3)
    return point_3d
```

Intermediate Results

For showing intermediate results, I have chosen the **Images 0 and 1 and t=0.7**

Numerical Computations

1. All values in the tables are rounded upto 3 decimals.
2. Table 1 shows the essential Matrix and recovered rotation and translation from it.

3. Table 2 shows the first 3 points from both the images (Outliers have been removed) and their corresponding calculated 3D and 4D points

Table 1

Essential Matrix	**Rotation**	**Translation**
[[0.005 0.176 -0.072] [-0.129 -0.014 -0.692] [0.067 0.681 -0.009]]	[[0.998, 0.003, 0.070] [-0.002, 1.000, -0.020] [-0.071, 0.020, 0.997]]	[[-0.963], [0.097], [0.251]]

Table 2

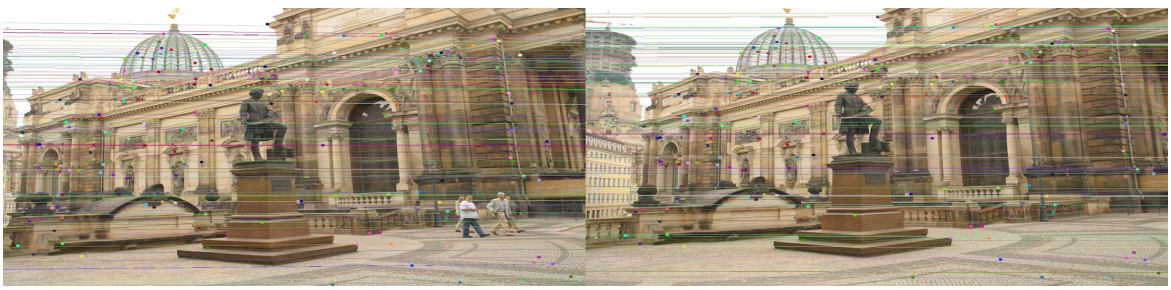
Points from image1 (normalised)	**Points from image2 (normalised)**	**Corresponding 4D points**	**Corresponding 3D points**
[[[-0.550 0.039]] [[-0.550 0.042]] [[-0.550 -0.150]]	[[[-0.469 0.020]] [[-0.470 0.023]] [[-0.470 -0.163]]	[[0.482 -0.034 -0.876 -0.009] [0.482 -0.037 -0.876 -0.009] [0.478 0.130 -0.869 -0.008]	[[-54.473 3.895 99.002] [-51.039 3.910 92.766] [-63.242 -17.217 115.008]

Visual Representation

Image 1 shows the epipolar lines in both the images and Table 3 shows SIFT, Inlier and outlier matches. The 3D reconstruction of the original points and after changing the camera 2 viewpoints are shown in Image 2. The 3D reconstructions from the chosen different point of view is not significantly different from the ones already presented.

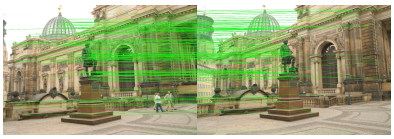
Epipolar Lines

Image 1

Epipolar Lines


SIFT, Inlier and Outlier Matches

Table 3

SIFT Matches (all matches)	Inlier Matches (Good matches based on ratio test)	Outlier Matches (Matches which did not pass ratio test)
		

3D Reconstruction

Image 2

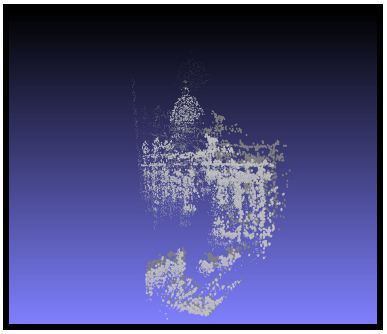
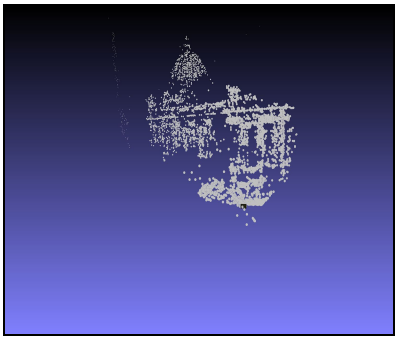
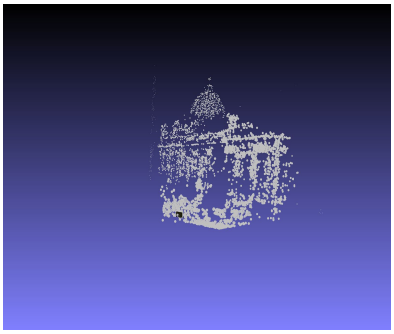
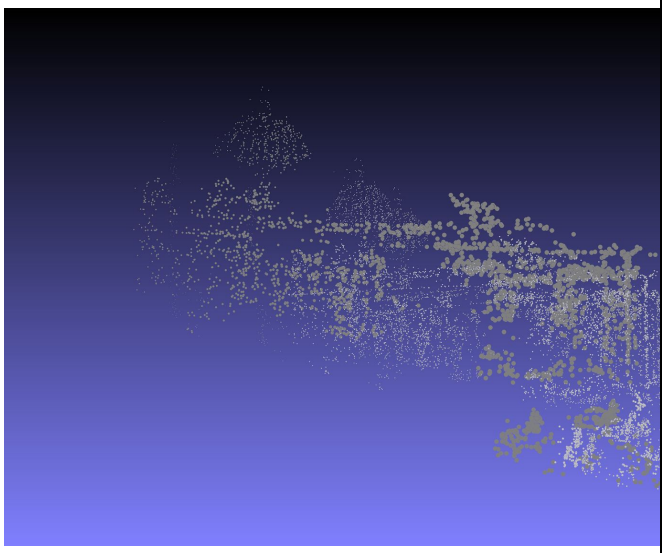
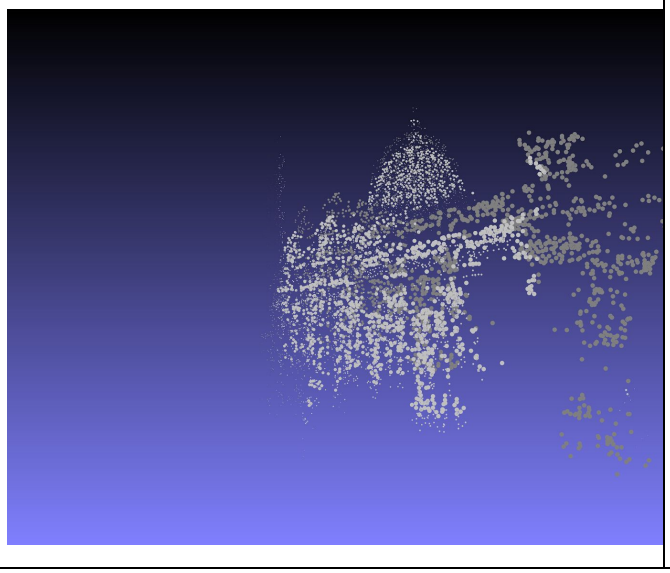
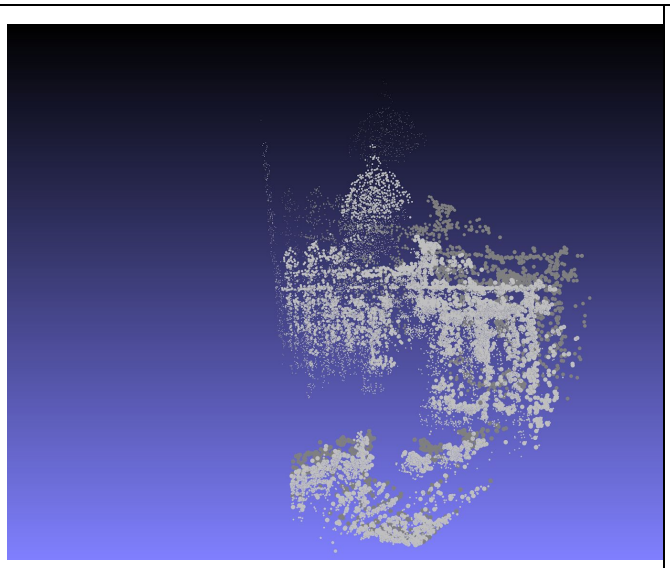
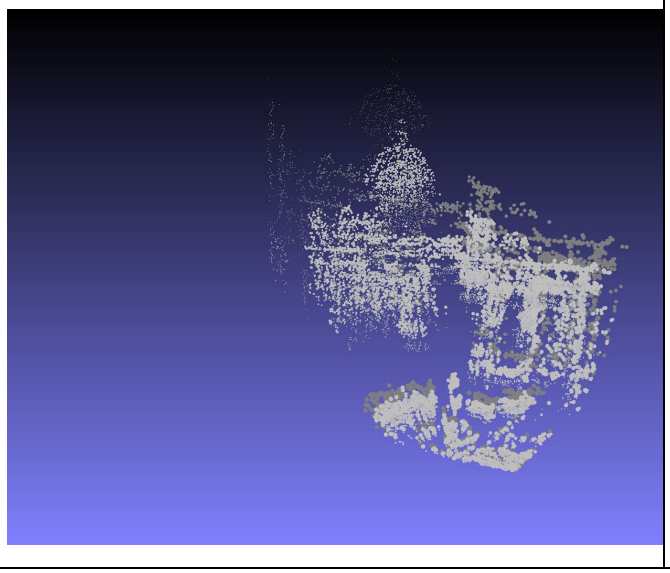
Original view from both cameras	**Camera 2 view changed (points2=points2/2)**	**Camera 2 view changed (points2.y=points2.y-1000)**
		

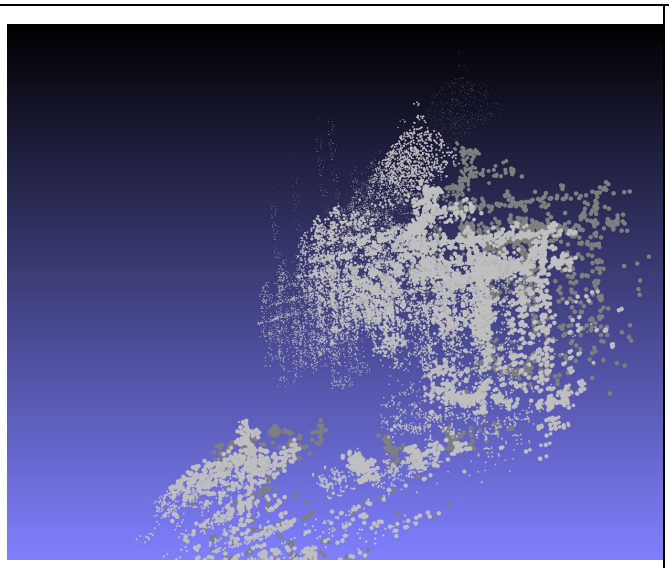
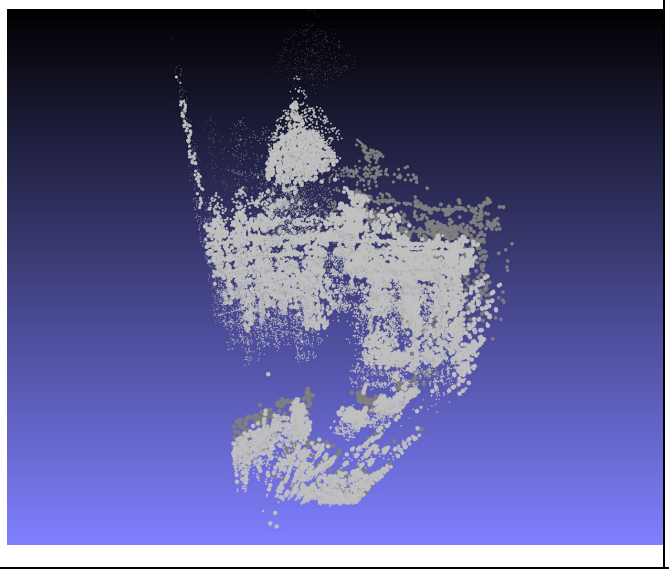
Image Pairwise Result

Table 4, 5 and 6 shows the pairwise results for image 0 & 1, image 1 & 2 and image 0 & 2 respectively

Table 4

t (ratio test)	Image 0 and 1	#Feature Points	#Feature Points after applying mask
0.5		3696	3476
0.6		4228	4133

0.7		4670	4469
0.75		4862	4394

0.8		5071	4557
0.9		5910	4946

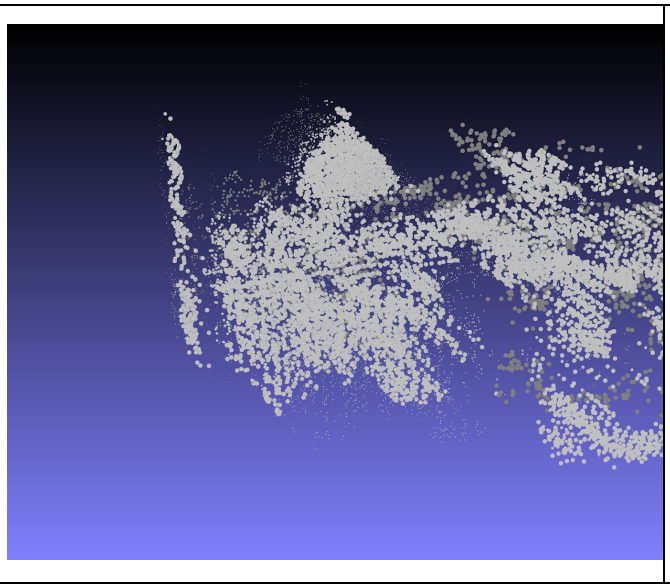
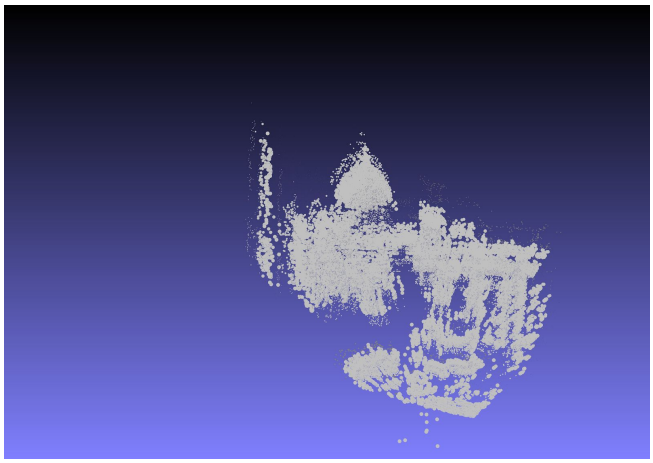
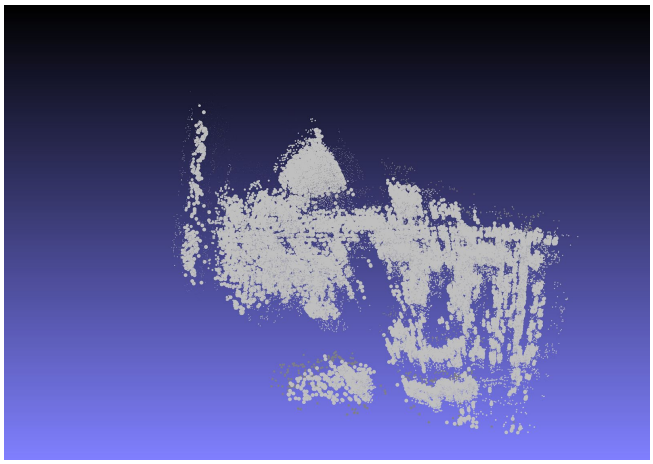
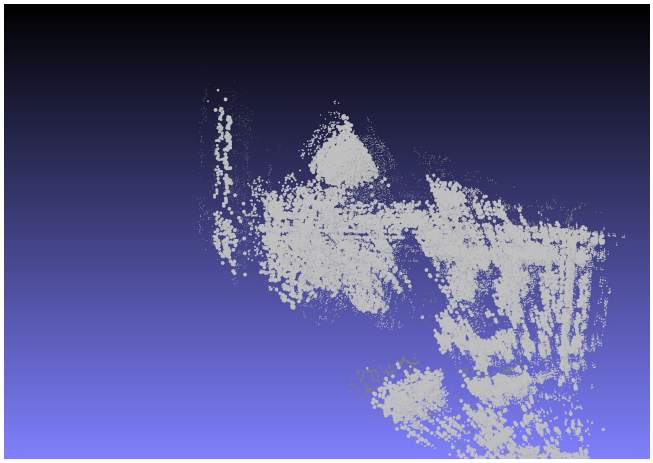
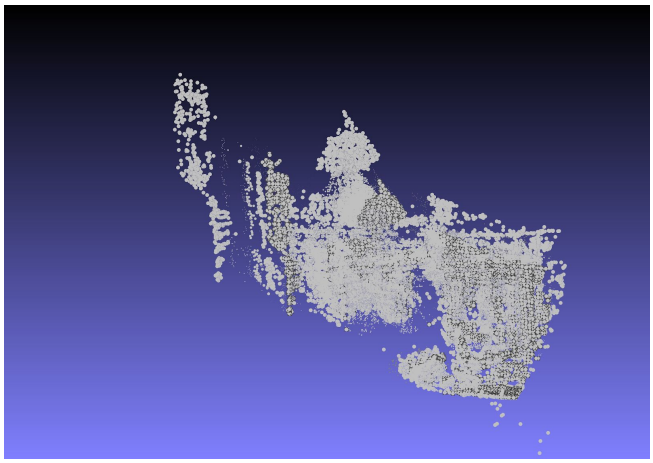
0.99		9102	5033

Table 5

t	Image 1 and 2	#Feature Points	#Feature Points after mask
0.5		4454	4407

0.6		5038	4364
0.7		5514	5318
0.75		5735	5479

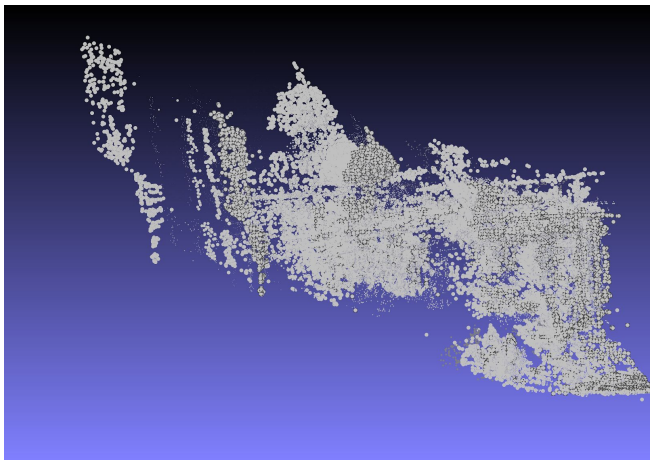
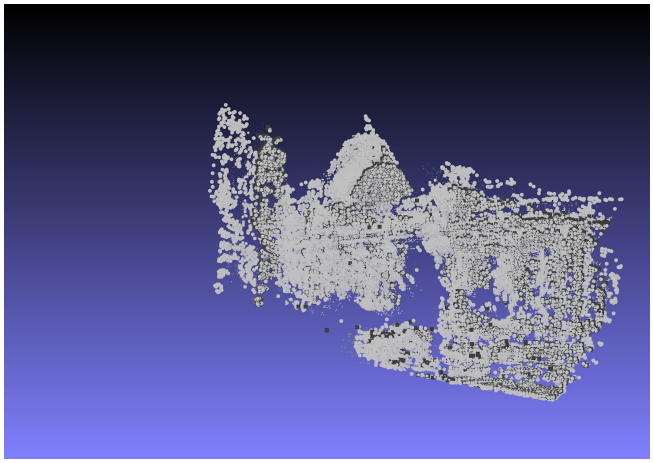
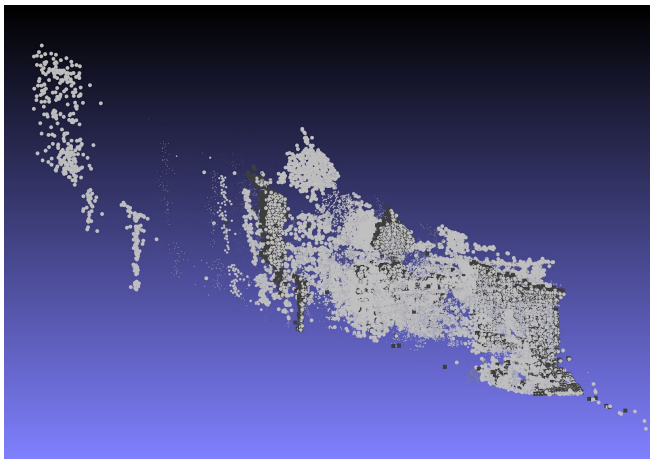
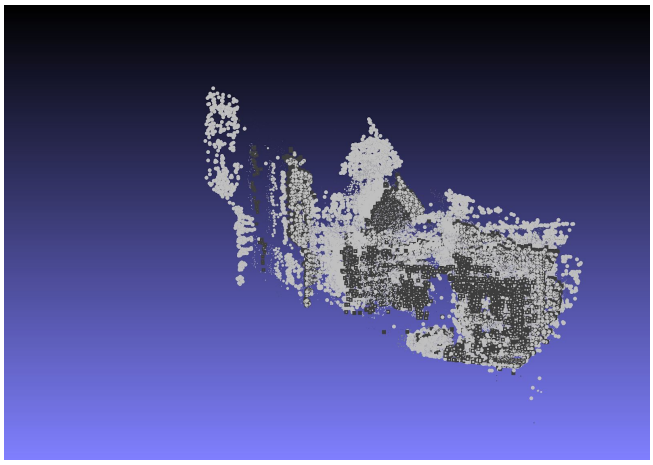
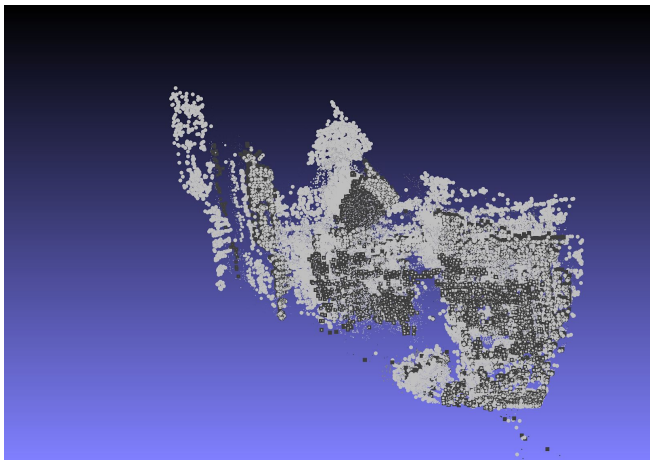
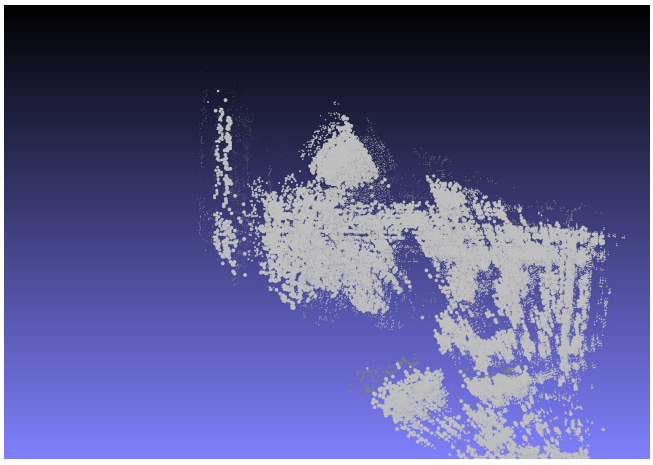
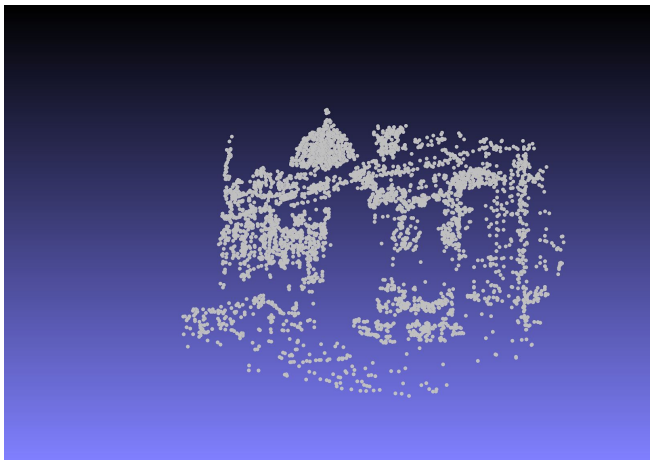
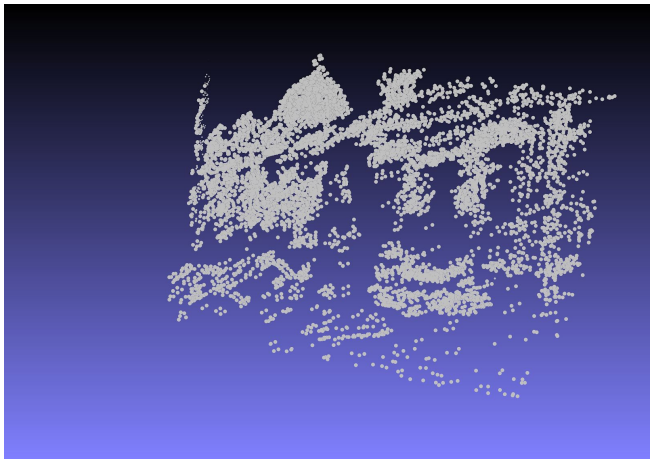
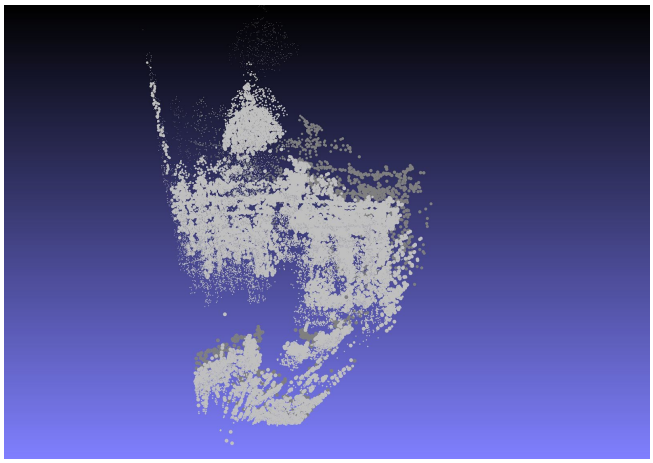
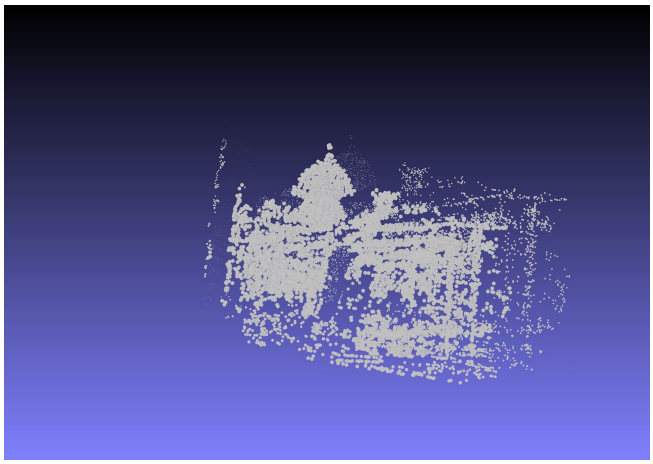
0.8		5973	5594
0.9		6786	5807
0.99		9600	6013

Table 6

t	Image 0 and 2	#Feature Points	#Feature Points after mask
0.5		3131	3105
0.6		3578	3497
0.7		3989	3792

0.75		4265	3910
0.8		4525	4065
0.9		5543	4365

0.99		9036	4642

Analysis

1. This method of 3D reconstruction by triangulation gives fairly good results and if the ratio test value is chosen wisely, the 3D construction comes out well.
2. Based on the observations, Flann Matcher has fewer points in 3D reconstruction and Brute Force works better in terms of getting better visualization.
3. In our current work, we have hardcoded the value of K=2 for KNN matches. It would be interesting to see the results for higher values of K.
4. The results might improve if we have more than 3 images.
5. We can use Bundle adjustment to further refine the results.

Optimal Value for Ratio Test

1. As we increase the ratio test (t), the number of matches also increases.
2. More number of matches give more outliers.
3. If t is low (<0.6), the number of matches are not enough and hence 3D visualization misses out important information. On the other hand, if t is high (>.85), the number of matches and outliers are large in numbers.
4. Appropriate value as per my observations is somewhere between 0.7 and 0.85.