Structure from Motion

By- Uday Raghuvanshi

ABSTRACT

This project aims at implementing the factorization method for structure from motion. The dataset for the project is a set of sequence images on the CMU hotel. The project involves detecting features in the first frame and tracking them along all the remaining frames to generate a matrix of tracked points. The points are tracked using the KLT tracker. The matrix is decomposed into a Motion and Structure Matrix using SVD. Finally, the aim is to find the Structure matrix which stores the 3D x,y and z coordinates of tracked points. The points obtained are then output in a PLY file so that they can be read and displayed using Meshlab.

For the project, the programming language used is python and modules including cv2, numpy, matplot and os are used.

DESCRIPTION OF ALGORITHM

In order to implement Structure from Motion, a SFM class is formed that has several methods to perform the steps involved in obtaining the Structure Matrix. The instance of SFM class takes 'imgs_arr' as the array of grayscale images, 'show_tracked_points' and 'show_scatter_plot' as a boolean value. If 'show_tracked_points' is set to True, the tracked points in each image will be displayed on the images. If 'show_scatter_plot' is set to True, the code will output a 3D plot, showing the x,y and z coordinates of tracked points.

The **SFM** class is initialized as follows:

```
if __name__ == "__main__":
    path_to_images = r'C:\Users\udayr\PycharmProjects\CVfiles\term_project\hotel\hotel'
    imgs_arr = []
    seq_files = [f for f in os.listdir(path_to_images) if f.startswith('hotel.seq')]
    seq_numbers=[int(f.split('.')[1].split('seq')[-1]) for f in seq_files]
    seq_files=[f for _, f in sorted(zip(seq_numbers, seq_files))]
    for img_name in seq_files:
        img = cv2.imread(path_to_images + '\\' + img_name, 0)
        imgs_arr.append(np.asarray(img).astype(float))

    corner_images=SFM(imgs_arr,show_tracked_points=False,show_scatter_plot=True).point_coords()
```

Figure 1: Initializing SFM class

The description of each method of the algorithm is as follows:

1. **tracking:** This method starts by taking the first frame of the sequence and finding corners on it using 'cv2.goodfeaturesToTrack'. This inbuilt function takes in the 'prev_frame' which is the frame on which it finds corner points, 'max_pts' which is the maximum number of corners it finds in the frame, 'quality' which defines the strength of corner points that are to be retained. It is a value between 0 and 1, higher quality would mean strong but a smaller number of points. It also takes 'min_dist' which is the minimum distance between the corner points.
   After finding the points, it loops over all the images in the sequence and tracks the corner points using 'cv2.calcOpticalFlowPyrLK'. It is an inbuilt function that implements the Lucas Kanade tracking. It takes in 'prev_gray' (previous frame), 'frame_gray' (current frame) and 'prev_corners' (previous corners) and outputs 'next_corners' (location of corners in current frame) and 'status' (a 0 or 1 value that states whether a point in previous frame was found in current frame).
   While iterating through the images, all the corners obtained in each frame are stored in a 'feat' list and their corresponding status in a 'ind' list. Additionally, the good pointsare filtered out by taking only the points with status=1 and use those points for the next frame. Finally, only those points that are tracked in all the frames are kept and stored in 'final_feat'. The W matrix is formed using these and later W_hat is formed after subtracting the mean of x and y coordinates in every frame. The tracking method returns W_hat matrix.
   Additionally, if the 'show_tracked_points' parameter is set to True, a sequence of images displaying the tracked corners starts running.

```python
def tracking(self):
    prev_gray=self.array_of_images[0].astype(np.uint8)
    max_pts=6000
    quality=0.01
    min_dist=0.01
    prev_corners = cv2.goodFeaturesToTrack(prev_gray,maxCorners=max_pts,qualityLevel=quality,minDistance=min_dist)
    termcrit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)
    feat=[]
    ind=[]
    for i in range(1,len(self.array_of_images)):
        frame_gray=self.array_of_images[i].astype(np.uint8)
        next_corners,status,_=cv2.calcOpticalFlowPyrLK(prev_gray,frame_gray, prev_corners, None, criteria=termcrit)
        good_pts=next_corners[status==1]
        feat.append(next_corners)
        ind.append(status)
        if self.show_tracked_points:
            good_pts_int=np.round(good_pts).astype(np.int32)
            output=self.array_of_images[i].astype(np.uint8)
            color_image = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
            # print(status[5])
            for j in range(good_pts_int.shape[0]):
                cv2.circle(color_image,(good_pts_int[j][0],good_pts_int[j][1]),2,(0,0,255))
            cv2.imshow('Klt tracking',color_image)
            cv2.waitKey(200)
        # print(good_pts.shape)
        prev_gray=frame_gray.copy()
        prev_corners = good_pts.reshape(-1,1,2)
    final_feat=[]
    final_good_pts=[]
    for i in range(len(ind)):
        p_temp=feat[i]
        for j in range(i,len(ind)):
            st_temp=ind[j]
            final_good_pts=p_temp[st_temp==1]
            p_temp=final_good_pts.reshape(-1,1,2)
        final_feat.append(final_good_pts)
```

Figure 2: tracking (i)

```
        final_feat.append(final_good_pts)
    # Forming W matrix
    rows=len(final_feat)
    cols=len(final_feat[0])
    W=np.zeros((2*rows,cols))
    for i in range(rows):
        for j in range(cols):
            W[i][j]=final_feat[i][j][0]
            W[i+rows][j]=final_feat[i][j][1]

    # Subtracting the mean
    W_hat=W-W.mean(axis=1).reshape(-1,1)

    return W_hat
```

Figure 3: tracking (ii)

2. **point_coords:** This method takes as input the W_hat matrix returned by the tracking method. As a first step, it performs Singular Value Decomposition (SVD) on the W_hat matrix and enforces a rank 3 on it. Then it forms a R_hat matrix of size 2*m x 3 and S_hat matrix of size 3xN, where m and N are number of frames and number of tracked points respectively. R_hat is a stack of rotation matrices.
   It then separates R_i and R_j from R_hat, R_i being all the x coordinates of tracked points in all frames and R_j being the y coordinates.
   It then forms a G matrix of size 2*m x 6 which finally helps obtain the Q matrix using Cholesky decomposition. The Structure matrix, S_true, is then obtained by the dot product of inverse of Q and S_hat matrix. The S_true matrix of size 3xN stores the x,y and z coordinates of all the tracked points. They are extracted and stored in a pointcloud matrix.
   The points are then written to a 'term_project.ply' file in ASCII format to be displayed in the meshlab software.
   Additionally, if the 'show_scatter_plot' parameter is set to True, a 3D scatter plot of all the points is obtained.

```python
    def point_coords(self):
        W_hat=self.tracking()
        u, d, v_transpose = np.linalg.svd(W_hat,full_matrices=True)
        d[3:]=0
        d_f=np.array([[d[0],0,0],[0,d[1],0],[0,0,d[2]]])
        u_f=np.array(u[:,:3])
        v_f=np.array(v_transpose[:3,:])
        d_f=np.sqrt(d_f)
        R_hat=np.dot(u_f,d_f)
        S_hat=np.dot(d_f,v_f)
        # m = number of frames
        m=R_hat.shape[0]//2
        R_i=R_hat[:m,:]
        R_j=R_hat[m:2*m,:]
        G=np.zeros((2*m,6))
        for i in range(m):
            G[2*i,0]=(R_i[i,0]**2)-(R_j[i,0]**2)
            G[2*i+1,0]=R_i[i,0]*R_j[i,0]
            G[2*i,1]=2*((R_i[i,0]*R_i[i,1])-(R_j[i,0]*R_j[i,1]))
            G[2*i+1,1]=R_i[i,1]*R_j[i,0]+R_i[i,0]*R_j[i,1]
            G[2*i,2]=2*((R_i[i,0]*R_i[i,2])-(R_j[i,0]*R_j[i,2]))
            G[2*i+1,2]=R_i[i,2]*R_j[i,0]+R_i[i,0]*R_j[i,2]

            G[2*i,3]=(R_i[i,1]**2)-(R_j[i,1]**2)
            G[2*i+1,3]=R_i[i,1]*R_j[i,1]
            G[2*i,4]=2*((R_i[i,2]*R_i[i,1])-(R_j[i,2]*R_j[i,1]))
            G[2*i+1,4]=R_i[i,2]*R_j[i,1]+R_i[i,1]*R_j[i,2]
            G[2*i,5]=(R_i[i,2]**2)-(R_j[i,2]**2)
            G[2*i+1,5]=R_i[i,2]*R_j[i,2]
        GU,GS,GVh=np.linalg.svd(G)
        Q_sq=np.zeros((3,3))
        GV=GVh[-1,:]
        Q_sq[0, 0] = GV[0]
        Q_sq[0, 1] = GV[1]
        Q_sq[1, 0] = GV[1]
```

Figure 5: point_coords (i)

```python
        Q_sq[2, 0] = GV[2]
        Q_sq[1, 1] = GV[3]
        Q_sq[2, 1] = GV[4]
        Q_sq[1, 2] = GV[4]
        Q_sq[2, 2] = GV[5]
        Q_sq=np.abs(Q_sq)
        Q=np.linalg.cholesky(Q_sq)
        R_true = np.dot(R_hat, Q)
        S_true = np.dot(np.linalg.inv(Q), S_hat)
        X = S_true[0, :]
        Y = S_true[1, :]
        Z = S_true[2, :]
        print(S_true.shape)
        factor = 1
        pointcloud = np.zeros((X.shape[0], 3))
        pointcloud[:, 0] = X
        pointcloud[:, 1] = Y
        pointcloud[:, 2] = Z * factor
        if self.show_scatter_plot:
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            pnt3d = ax.scatter(pointcloud[:, 0],  # x
                               pointcloud[:, 1],  # y
                               pointcloud[:, 2],  # z
                               c='b',
                               marker="o")
            plt.show()
        with open('term_project.ply', 'w') as f:
            # Write the PLY header
            f.write('ply\nformat ascii 1.0\nelement vertex {}\nproperty float x\nproperty float y\nproperty float z\nend_header\n'.format(pointcloud.shape[0]))
            # Write the point coordinates from the matrix
            for i in range(pointcloud.shape[0]):
                f.write('{} {} {}\n'.format(pointcloud[i, 0], pointcloud[i, 1], pointcloud[i, 2]))
```

Figure 4: point_coords (ii)

RESULTS

The results obtained in the meshlab show roughly the 3D structure of the CMU hotel. More the number of tracked points, denser is the mesh obtained. Hence, the parameters of the 'cv2.goodfeaturesToTrack' function were varied to generate different types of meshes. The results are shown below:

1. Max corners in first frame: 500

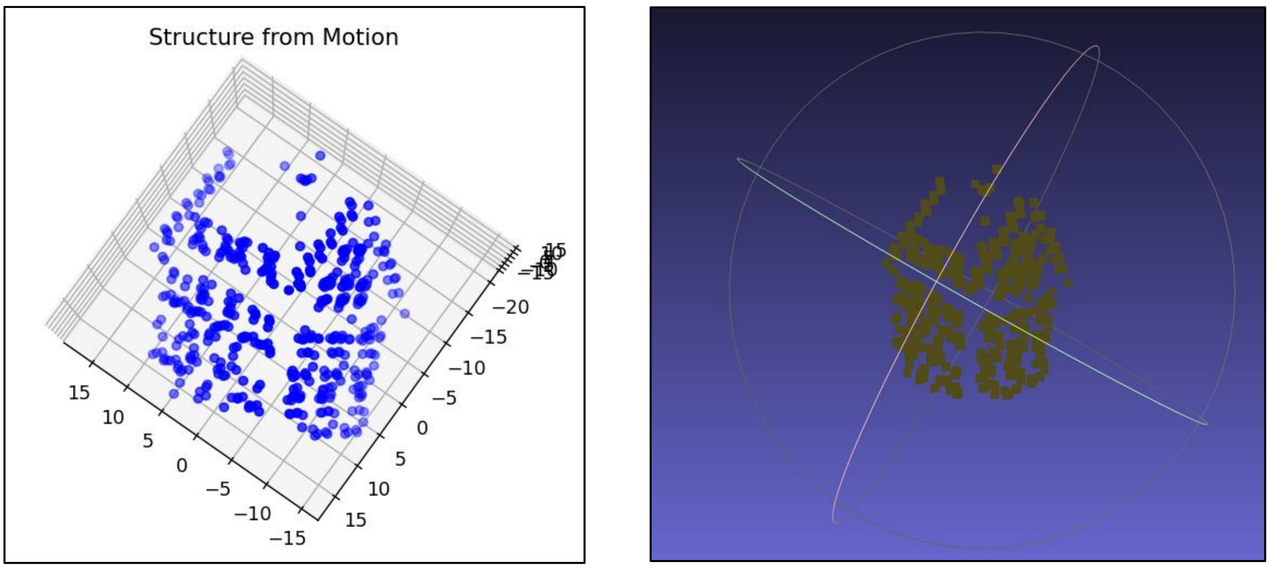

Figure 6: Scatter plot and mesh (max 500 corners)

2. Max corners in first frame: 1000

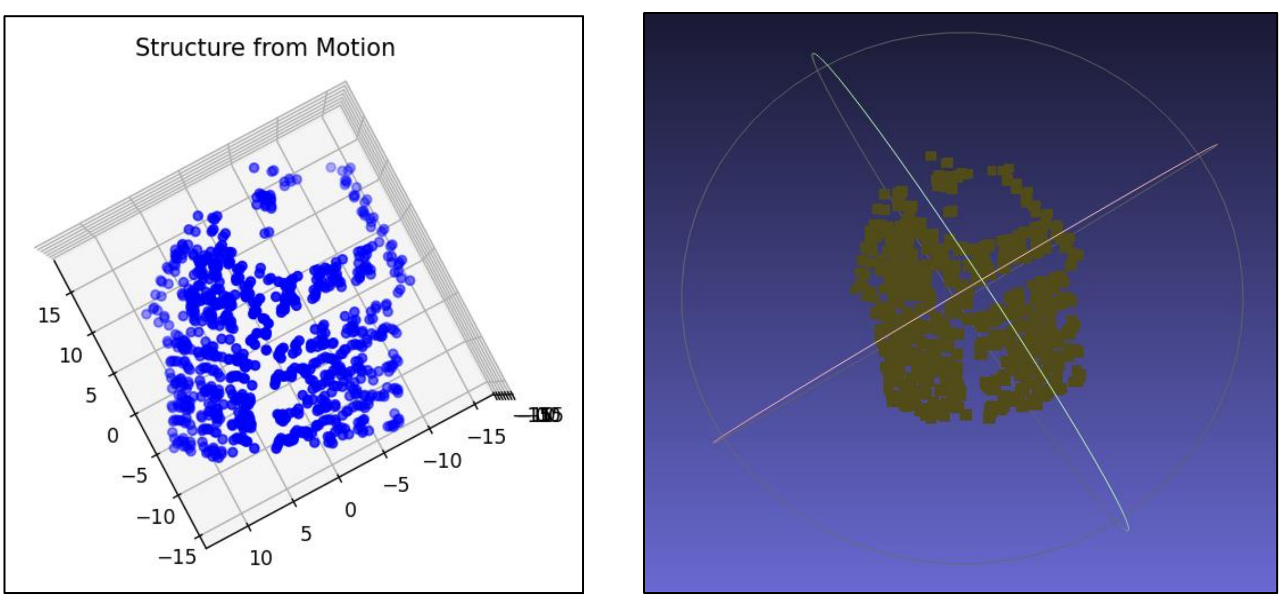

Figure 7: Scatter plot and mesh (max 1000 corners)

3. Max corners in first frame: 3000 and quality: 0.001

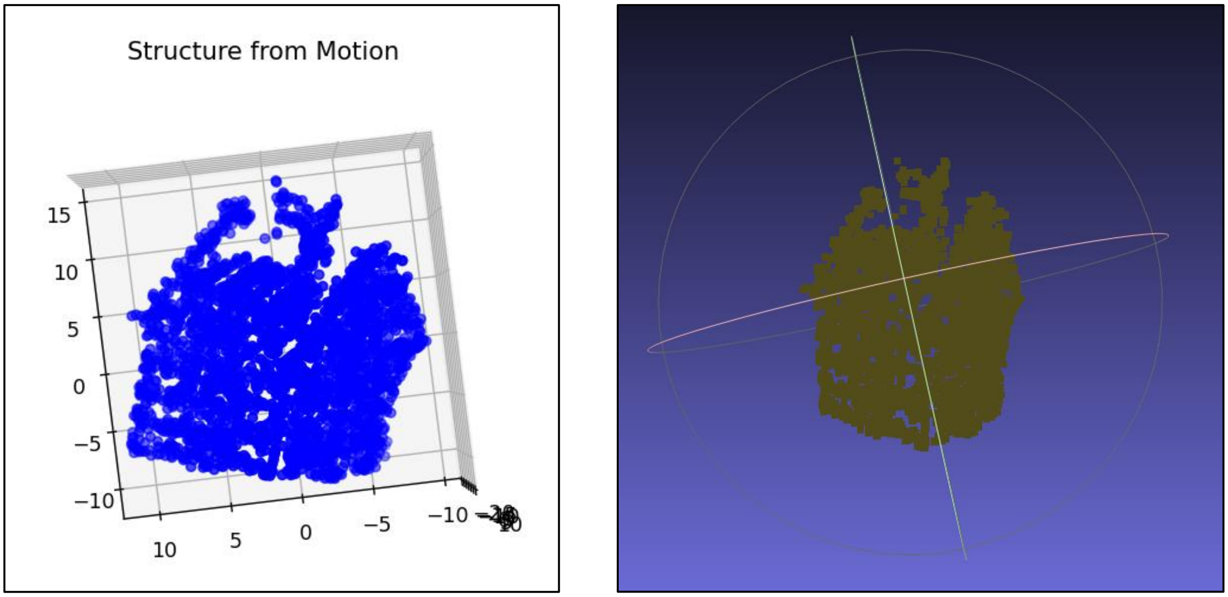

Figure 8: Scatter plot and mesh (max 3000 corners)

4. Max corners in first frame: 6000 and quality 0.0005

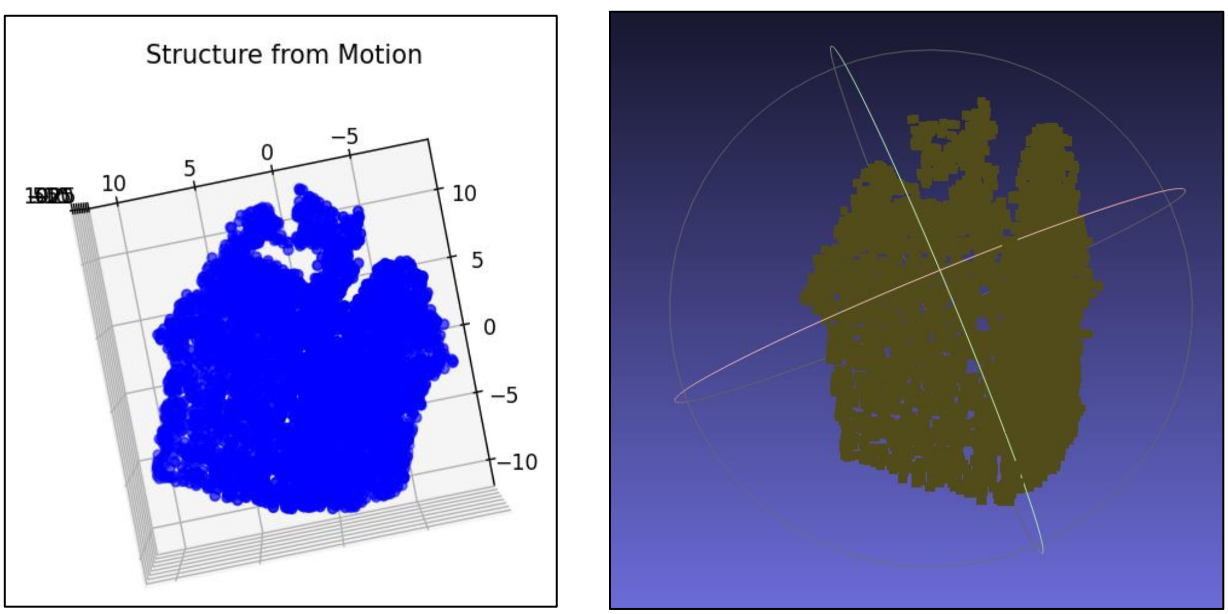

Figure 9:Scatter plot and mesh (max 6000 corners)

CONCLUSION

From the above results, it can be concluded that the code is successful in implementing the structure from motion. The greater the number of points that are tracked across all frames, more dense is the mesh obtained at the end. So, depending upon how much features are present and how dense mesh is required, the parameters can be tuned and the mesh can be obtained using the Factorization method of Structure from Motion.

APPENDIX

```
import numpy as np
import cv2
import sys
import os
import matplotlib.pyplot as plt

np.set_printoptions(threshold=sys.maxsize)

class SFM:
    def __init__(self, array_of_images:
list,show_tracked_points:bool,show_scatter_plot:bool):
        self.array_of_images = array_of_images
        self.show_tracked_points=show_tracked_points
        self.show_scatter_plot=show_scatter_plot

    def tracking(self):
        prev_gray=self.array_of_images[0].astype(np.uint8)
        max_pts=6000
        quality=0.0005
        min_dist=0.01
        prev_corners =
cv2.goodFeaturesToTrack(prev_gray,maxCorners=max_pts,qualityLevel=quality,minDistance=
min_dist)
        termcrit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)
        feat=[]
        ind=[]
        for i in range(1,len(self.array_of_images)):
            frame_gray=self.array_of_images[i].astype(np.uint8)
            next_corners,status,_=cv2.calcOpticalFlowPyrLK(prev_gray,frame_gray,
prev_corners, None, criteria=termcrit)
            good_pts=next_corners[status==1]
            feat.append(next_corners)
            ind.append(status)
            if self.show_tracked_points:
                good_pts_int=np.round(good_pts).astype(np.int32)
                output=self.array_of_images[i].astype(np.uint8)
                color_image = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
                # print(status[5])
                for j in range(good_pts_int.shape[0]):

cv2.circle(color_image,(good_pts_int[j][0],good_pts_int[j][1]),2,(0,0,255))
                cv2.imshow('Klt tracking',color_image)
                cv2.waitKey(200)
            print(good_pts.shape)
            prev_gray=frame_gray.copy()
            prev_corners = good_pts.reshape(-1,1,2)
        final_feat=[]
```

```python
        final_good_pts=[]
        for i in range(len(ind)):
            p_temp=feat[i]
            for j in range(i,len(ind)):
                st_temp=ind[j]
                final_good_pts=p_temp[st_temp==1]
                p_temp=final_good_pts.reshape(-1,1,2)
            final_feat.append(final_good_pts)
        # Forming W matrix
        rows=len(final_feat)
        cols=len(final_feat[0])
        W=np.zeros((2*rows,cols))
        for i in range(rows):
            for j in range(cols):
                W[i][j]=final_feat[i][j][0]
                W[i+rows][j]=final_feat[i][j][1]

        # Subtracting the mean
        W_hat=W-W.mean(axis=1).reshape(-1,1)

        return W_hat

    def point_coords(self):
        W_hat=self.tracking()
        u, d, v_transpose = np.linalg.svd(W_hat,full_matrices=True)
        d[3:]=0
        d_f=np.array([[d[0],0,0],[0,d[1],0],[0,0,d[2]]])
        u_f=np.array(u[:,:3])
        v_f=np.array(v_transpose[:3,:])
        d_f=np.sqrt(d_f)
        R_hat=np.dot(u_f,d_f)
        S_hat=np.dot(d_f,v_f)
        # m = number of frames
        m=R_hat.shape[0]//2
        R_i=R_hat[:m,:]
        R_j=R_hat[m:2*m,:]
        G=np.zeros((2*m,6))
        for i in range(m):
            G[2*i,0]=(R_i[i,0]**2)-(R_j[i,0]**2)
            G[2*i+1,0]=R_i[i,0]*R_j[i,0]
            G[2*i,1]=2*((R_i[i,0]*R_i[i,1])-(R_j[i,0]*R_j[i,1]))
            G[2*i+1,1]=R_i[i,1]*R_j[i,0]+R_i[i,0]*R_j[i,1]
            G[2*i,2]=2*((R_i[i,0]*R_i[i,2])-(R_j[i,0]*R_j[i,2]))
            G[2*i+1,2]=R_i[i,2]*R_j[i,0]+R_i[i,0]*R_j[i,2]

            G[2*i,3]=(R_i[i,1]**2)-(R_j[i,1]**2)
            G[2*i+1,3]=R_i[i,1]*R_j[i,1]
            G[2*i,4]=2*((R_i[i,2]*R_i[i,1])-(R_j[i,2]*R_j[i,1]))
            G[2*i+1,4]=R_i[i,2]*R_j[i,1]+R_i[i,1]*R_j[i,2]
            G[2*i,5]=(R_i[i,2]**2)-(R_j[i,2]**2)
            G[2*i+1,5]=R_i[i,2]*R_j[i,2]
        GU,GS,GVh=np.linalg.svd(G)
        Q_sq=np.zeros((3,3))
        GV=GVh[-1,:]
        Q_sq[0, 0] = GV[0]
        Q_sq[0, 1] = GV[1]
        Q_sq[1, 0] = GV[1]
        Q_sq[0, 2] = GV[2]
        Q_sq[2, 0] = GV[2]
        Q_sq[1, 1] = GV[3]
        Q_sq[2, 1] = GV[4]
        Q_sq[1, 2] = GV[4]
        Q_sq[2, 2] = GV[5]
```

```
        Q_sq=np.abs(Q_sq)
        Q=np.linalg.cholesky(Q_sq)
        R_true = np.dot(R_hat, Q)
        S_true = np.dot(np.linalg.inv(Q), S_hat)
        X = S_true[0, :]
        Y = S_true[1, :]
        Z = S_true[2, :]
        # print(S_true.shape)
        factor = 1
        pointcloud = np.zeros((X.shape[0], 3))
        pointcloud[:, 0] = X
        pointcloud[:, 1] = Y
        pointcloud[:, 2] = Z * factor
        if self.show_scatter_plot:
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            pnt3d = ax.scatter(pointcloud[:, 0],  # x
                               pointcloud[:, 1],  # y
                               pointcloud[:, 2],  # z
                               c='b',
                               marker="o")
            plt.title(f'Structure from Motion')
            plt.show()
        with open('term_project.ply', 'w') as f:
            # Write the PLY header
            f.write('ply\nformat ascii 1.0\nelement vertex {}\nproperty float
x\nproperty float y\nproperty float z\nend_header\n'.format(pointcloud.shape[0]))
            # Write the point coordinates from the matrix
            for i in range(pointcloud.shape[0]):
                f.write('{} {} {}\n'.format(pointcloud[i, 0], pointcloud[i, 1],
pointcloud[i, 2]))

if __name__ == "__main__":
    path_to_images =
r'C:\Users\udayr\PycharmProjects\CVfiles\term_project\hotel\hotel'
    imgs_arr = []
    seq_files = [f for f in os.listdir(path_to_images) if f.startswith('hotel.seq')]
    seq_numbers=[int(f.split('.')[1].split('seq')[-1]) for f in seq_files]
    seq_files=[f for _, f in sorted(zip(seq_numbers, seq_files))]
    for img_name in seq_files:
        img = cv2.imread(path_to_images + '\\' + img_name, 0)
        imgs_arr.append(np.asarray(img).astype(float))


corner_images=SFM(imgs_arr,show_tracked_points=False,show_scatter_plot=True).point_coo
rds()
```