

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

School of Computer Science and Engineering

Nanyang Technological University

AY2024/25

AI6121 Computer Vision

Group Project

Group Members:

Kee Ming Yuan (G2304842E)

Yip Chen Fei (G2304486K)

Timothy Yeo Xiao Jing (G2406412E)

Lehan Gong (G2205030D)

Wong Seik Man (G2303494L)

Content

Content	2
1. Introduction.....	3
1.1 Image Capture Process for 3D Reconstruction	4
1.1.1 Textured Scene Selection	4
1.1.2 Consistent Illumination	4
1.1.3 High Visual Overlap Between Images	5
1.1.4 Different viewpoints with camera movement	5
2. Experiment Methodology.....	6
2.1 Evaluation Methods	6
2.1.1 Visual Analysis by team members	6
3. Structure from Motion Algorithm.....	7
3.1 Feature Matching	7
3.2 Estimating Fundamental Matrix	8
3.3 Estimating Essential Matrix from Fundamental Matrix.....	13
3.3.1 Finding Intrinsic Camera Calibration Matrix	13
3.4 Estimate Camera Pose from Essential Matrix	15
3.5 Check for Cheirality Condition using Triangulation	18
3.6 Perspective-N-Point	20
4. Sparse Model Result and Discussion	25
4.1 Experiment Results of the Hive	25
4.2 Experiment Results of “The Wind and Wings” sculpture.....	27
4.3 Experiment Results of Nanyang University Memorial	29
5. COLMAP Dense Model.....	31
5.1 Photometric Information	32
5.2 Geometric Information	33
5.3 Filtering and Fusion (Final step)	34
5.4 Experiment Results	34
6. Conclusion	36
References	37

1. Introduction

Structure-from-Motion (SfM) is a powerful computer vision technique that enables the reconstruction of three-dimensional (3D) structures from a series of two-dimensional (2D) images. By leveraging the geometric relationships between multiple overlapping views of a scene, SfM incrementally estimates camera poses and reconstructs the spatial coordinates of points in 3D space. SfM can be used in applications such as 3D modelling, augmented reality and urban planning.

In this project, the SfM pipeline was employed to reconstruct 3D models of various structures at Nanyang Technological University (NTU), including the Hive, the "Wind and Wings" sculpture, and the Nanyang University Memorial. The team begin with the systematic capture of multi-view images under carefully controlled conditions to ensure high-quality reconstructions. Key considerations during the image acquisition process included selecting textured scenes, maintaining consistent illumination, ensuring significant visual overlap between images, and capturing the scene from diverse viewpoints.

In addition, ablation study on the SfM algorithm was conducted. The SfM algorithm utilized robust feature matching techniques, such as Scale-Invariant Feature Transform (SIFT), to extract distinctive keypoints and descriptors across images. These features formed the basis for estimating the fundamental matrix through the epipolar geometry concept, essential matrix, and camera poses. The iterative process of triangulating new 3D points and incorporating additional camera views allowed for the incremental construction of sparse 3D models, which were later refined into dense reconstructions using advanced techniques such as Multi-View Stereo (MVS).

The experimental results showcase the effectiveness of SfM in capturing intricate architectural and geometric details of NTU structures. However, the team faced challenges such as sparse coverage in textureless or reflective areas. This highlights the importance of image capturing process to ensure high quality images and good coverage of the captured structure in the image to ensure a comprehensive and detailed 3D reconstruction.

1.1 Image Capture Process for 3D Reconstruction

Structures in NTU are used for 3D reconstruction in this report. To achieve a high-quality 3D reconstruction of a selected scene on the NTU campus, a systematic approach was taken to capture multi-view 2D images. The following guidelines were followed to ensure effective 3D reconstruction:

1.1.1 Textured Scene Selection

The scene was chosen with an emphasis on capturing surfaces that contain sufficient texture, as texture-less areas (like white walls or empty desks) lack distinctive features necessary for feature matching. Capturing the useful information is a crucial step in the Structure from Motion (SfM) reconstruction process.



Fig 1: Pagoda with many unique shapes create good texture for the image.

1.1.2 Consistent Illumination

To prevent issues related to high dynamic range (HDR) scenes, images were captured under controlled lighting conditions, avoiding areas with extreme light contrasts, such as direct sunlight, shadows, or images taken through doors and windows. Reflective surfaces, especially mirrors, were avoided, as they can confuse the SfM algorithm by introducing duplicate or misleading points in the scene. This consistency in lighting and careful exclusion of reflections ensures uniformity in color and texture across images, enhancing the accuracy of feature detection and matching.



Fig 2: A poor image capture where the camera is pointed at the Sun causing irregular lighting throughout the image.

1.1.3 High Visual Overlap Between Images

Images were captured with significant overlap (60-80%) between each view, ensuring that key features appeared across multiple frames. This overlap allows the same object or area to be visible, aiding in accurate triangulation of points for the 3D structure. By capturing each part of the scene from multiple angles, the dataset supports a robust point cloud reconstruction, representing the structure with higher fidelity.



Fig 3: Image pair with 60-80% of image content overlapping the other image content.

1.1.4 Different viewpoints with camera movement

Images were taken from different perspectives by changing both position and angle, rather than only rotating the camera from a fixed location. This involved taking a few steps after each shot to introduce spatial diversity in the capture points, which assists in resolving depth and spatial relationships within the scene. Ensure that images from relatively similar viewpoints were also included to balance coverage and maintain manageable data volume for processing.



Fig 4: Building image taken at different angle.

2. Experiment Methodology

3D reconstruction will be conducted using both the dense and sparse model on the same set of NTU structures.

Dense models are normally preferred for 3D reconstruction as they can output a more detailed and realistic representation of the NTU structures such as its actual shape, colour and relative size of the features of the NTU structure. That said, dense models are normally computationally expensive as they have more points in a 3D cloud.

As sparse models have less points in the 3D cloud as they are faster to compute, they are normally used to give an overall shape of the structure and the shape of its key features. They lack the finer details such as colours compared to dense models.

2.1 Evaluation Methods

The team will be using visual analysis of the dense and sparse model to evaluate its performance.

2.1.1 Visual Analysis by team members

Visual analysis to evaluate performance of the dense and sparse model is an effective approach to assessing the 3D reconstruction of the structure in the image set. The more the 3D point cloud from the dense or sparse model resembles the actual NTU structure, the more the dense or sparse model will be concluded as performing well.

For the dense model, the 3D reconstruction of the NTU structures by the dense model will be rotated at all angles carefully to have a holistic view of the 3D reconstruction for visual evaluation. For the sparse model, the points in the 3D clouds can be compared side by side with the actual image of the NTU structure.

However, while visual assessment is straightforward and can be quite telling, quantitative evaluation is also necessary for several reasons:

1. The human eye is unable to detect small deviation of the 3D point cloud from the actual shape of the NTU structure. Moreover, it is difficult to judge if the scale of all parts of the 3D point cloud reconstruction is the same proportion as the actual structure.
2. Different team members may have different level of importance of different types of imperfections of the 3D reconstruction such as structure misalignment, scale of structure and colours of structure. This will lead to the teammates having different opinions of which 3D reconstruction is better.

That said, visual analysis still provided a good judgement of whether the 3D reconstruction is good or not.

3. Structure from Motion Algorithm

The team tested the structure from motion algorithm with reference to [1], [2] and [7]. The 3D reconstruction model is built on the images of the Hive from NTU and tested on the 'The Wind and Wings' sculpture from the NTU museum and the Nanyang University Memorial from the NTU Chinese Heritage Centre.

3.1 Feature Matching

Scale-Invariant Feature Transform (SIFT) is used to return key points and descriptors of key points for the image pairs. Key points are unique features of the object, and many key points can be identified even for small objects. The key points can be identified for all kinds of objects, making it very robust. SIFT allows the key points and descriptors of the images to be detected even if the camera is rotated or moved closer or further from the target object, making it suitable for multi-view 3D reconstruction. Descriptors are vector representation of the key points.

Table 1: Code Snippet of feature matching

```
def GetAlignedMatches(kp1,desc1,kp2,desc2,matches):

    #Sorting in case matches array isn't already sorted
    matches = sorted(matches, key = lambda x:x.distance)

    #retrieving corresponding indices of keypoints (in both images) from matches.
    img1idx = np.array([m.queryIdx for m in matches])
    img2idx = np.array([m.trainIdx for m in matches])

    #filtering out the keypoints that were NOT matched.
    kp1_ = (np.array(kp1))[img1idx]
    kp2_ = (np.array(kp2))[img2idx]

    #retrieving the image coordinates of matched keypoints
    img1pts = np.array([kp.pt for kp in kp1_])
    img2pts = np.array([kp.pt for kp in kp2_])

    return img1pts,img2pts,img1idx,img2idx
```

```

def GetImageMatches(img1,img2):
    surfer=cv2.SIFT_create()
    kp1, desc1 = surfer.detectAndCompute(img1,None)
    kp2, desc2 = surfer.detectAndCompute(img2,None)

    matcher = cv2.BFMatcher(crossCheck=True)
    matches = matcher.match(desc1, desc2)

    matches = sorted(matches, key = lambda x:x.distance)

    return kp1,desc1,kp2,desc2,matches

#Getting SIFT/SURF features for image matching
kp1,desc1,kp2,desc2,matches=GetImageMatches(img1,img2)

#Aligning two keypoint vectors
img1pts,img2pts,img1idx,img2idx=GetAlignedMatches(kp1,desc1,kp2,desc2,matches)

```

The key points and descriptors of each images are computed using `surfer.detectAndCompute`. The Brute Force Matcher (`cv2.BFMatcher`) compares the distance of the descriptors in both images using the Hamming Distance [3]. The smaller the distance, the more similar the descriptors and the more likely they are matched. Descriptors with large distances are not matched. The coordinates of the matching key points are retrieved.

3.2 Estimating Fundamental Matrix

After performing feature matching, we will need to estimate the fundamental matrix. The fundamental matrix is a 3 by 3 (rank 2) matrix that states the relationship between corresponding set of points in two images from different views. To be able to calculate the fundamental matrix, we will first need to understand what is epipolar geometry. Figure 5 illustrates the epipolar geometry.

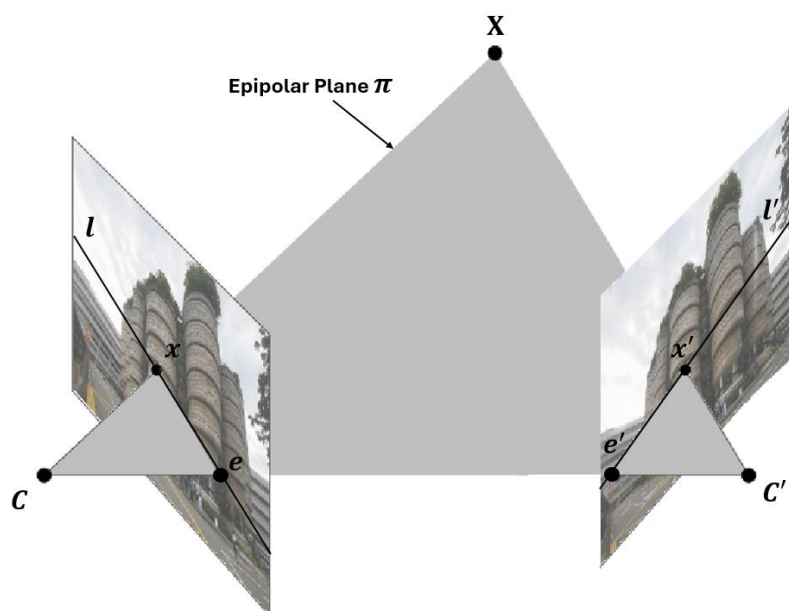


Fig 5: Epipolar geometry

C and C' are the camera center. The point X in the 3D space is projected onto two image planes, it produces corresponding point x in the first image and x' in the second image. Both points, x and x' lies on the same epipolar plane, π . The intersection of the epipolar plane with the image plane forms the epipolar lines, l in the first image and l' in the second image. The epipolar lines will intersect at the epipole, e in the first image and e' in the second image. The epipole is the point of intersection of the line joining the camera centers with the image plane. For example, given that point x is known and we want to find out where point x' lies. Because point x determines the epipolar line, l' which also means that point x' will lie along the epipolar line and on the intersection of l' and π . Similarly, point x' determines the epipolar line, l where point x should lie. With these concepts, we will be able to formulate the relationship between corresponding image points x and x' defined by the fundamental matrix. This is known as the epipolar constraint. We can formulate Eq.1 as follow:

$$x_i'^T F x_i = 0 \quad \text{where } i = 1, 2, \dots, m$$

Eq. 1

We can further expand Eq.1 as Eq.2. As the fundamental matrix is a 3 by 3 matrix, this means that we have 9 unknowns.

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

Eq.2

Eq.2 can be further transformed into Eq.3 for m number of correspondences.

$$\begin{bmatrix} x_1 x_1' & x_1 y_1' & x_1 & y_1 x_1' & y_1 y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_m x_m' & x_m y_m' & x_m & y_m x_m' & y_m y_m' & y_m & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

Eq.3

To solve this equation, each point contributes to one constraint as the epipolar constraint is a scalar equation. Therefore, we need at least 8 points to solve the Eq.3 also known as eight-point algorithm. To find F , we can further express the Eq.3 to Eq.4.

$$Ax = 0$$

$$A = \begin{bmatrix} x_1 x_1' & x_1 y_1' & x_1 & y_1 x_1' & y_1 y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_m x_m' & x_m y_m' & x_m & y_m x_m' & y_m y_m' & y_m & x_m' & y_m' & 1 \end{bmatrix}$$

$$x = \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix}$$

Eq.4

Using the concept of linear least squares, we can apply singular value decomposition (SVD) to A . A would be decompose into Eq.5.

$$A = U\Sigma V^T$$

Eq.5

- U and V are orthogonal matrices
- Σ is a diagonal matrix containing the singular values

The last column vector of V will be the solution of F . However, due to noise in the correspondences, F might be rank 3. If is rank 3, it means that the epipolar lines will not intersect at the same common point as shown in Figure 6. Therefore, we will need to ensure that the matrix is of rank 2 to have a common solution by setting the last singular value of F to zero.

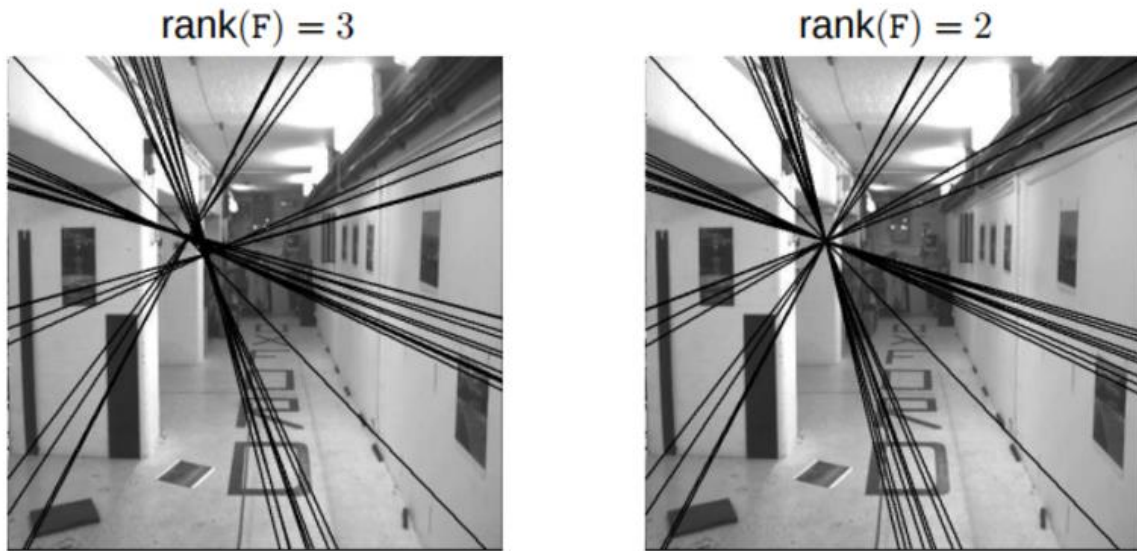


Fig 6: Visualization of epipolar lines of rank 3 matrix and rank 2 matrix. Retrieved from [6]

To improve the accuracy of the fundamental matrix F , the RANSAC algorithm is applied to eliminate outliers. The F matrix with the highest number of inliers is selected as the best estimate. RANSAC is used because feature matching often includes incorrect matches, referred to as outliers, which can negatively impact the computation of F . By identifying and excluding these outliers, RANSAC ensures a more robust and reliable

estimation of the fundamental matrix. Figure 7 shows the visualization of the epipolar lines for the Hive after computation of fundamental matrix.

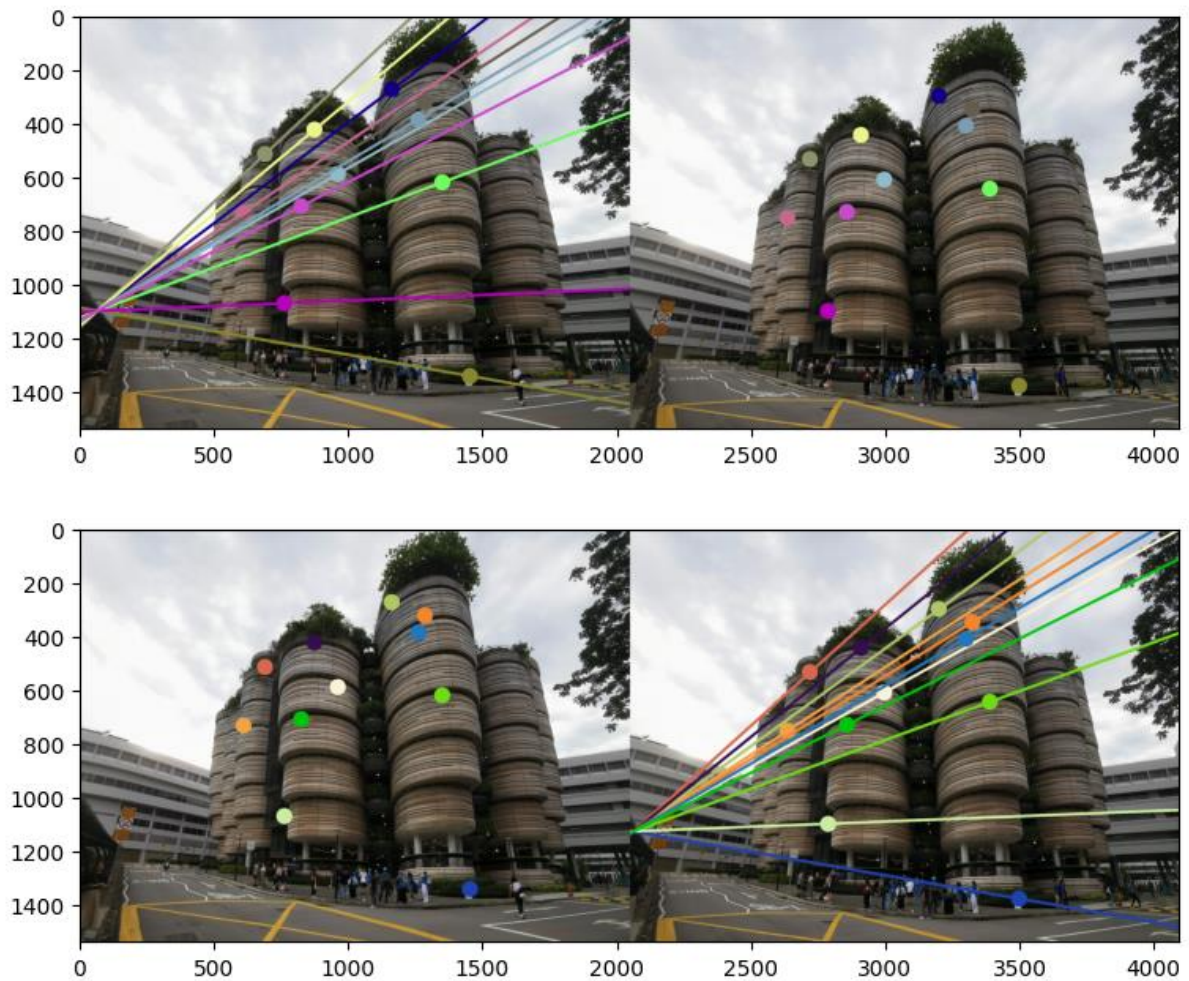


Fig 7: Visualization of epipolar lines for the Hive after computation of fundamental matrix

Table 2: Code Snippet for finding the intrinsic camera calibration matrix

```
def EstimateFundamentalMatrixRANSAC(img1pts,img2pts,outlierThres,prob=None,itors=None):
    if img1pts.shape[1]!=2:
        #converting to homogenous coordinates if not already
        img1pts = cv2.convertPointsToHomogeneous(img1pts)[:,:0,:]
        img2pts = cv2.convertPointsToHomogeneous(img2pts)[:,:0,:]

    bestInliers, bestF, bestmask = 0, None, None

    for i in range(itors):

        #Selecting 8 random points
        mask = np.random.randint(low=0,high=img1pts.shape[0],size=(8,))
        img1ptsiter = img1pts[mask]
        img2ptsiter = img2pts[mask]

        #Fitting fundamental matrix and evaluating error
        Fiter = EstimateFundamentalMatrix(img1ptsiter,img2ptsiter)
        err = SampsonError(Fiter,img1pts,img2pts)
        mask = err < outlierThres
        numInliers = np.sum(mask)
```

```

    #Updating best measurements if appropriate
    if bestInliers < numInliers:
        bestInliers = numInliers
        bestF = Fiter
        bestmask = mask

def EstimateFundamentalMatrix(x1,x2):
    if x1.shape[1]!=2: #converting to homogenous coordinates if not already
        x1 = cv2.convertPointsToHomogeneous(x1)[:,:0:]
        x2 = cv2.convertPointsToHomogeneous(x2)[:,:0:]

    A = np.zeros((x1.shape[0],9))

    #Constructing A matrix (vectorized)
    x1_ = x1.repeat(3,axis=1)
    x2_ = np.tile(x2, (1,3))

    A = x1_*x2_

    u,s,v = np.linalg.svd(A)
    F = v[-1,:].reshape((3,3),order='F')

    u,s,v = np.linalg.svd(F)
    F = u.dot(np.diag(s).dot(v))

    F = F / F[-1,-1]

    return F

F, mask = EstimateFundamentalMatrixRANSAC(img1pts,img2pts,.1,itters=20000)

lines2=sfmnp.ComputeEpiline(img1pts[mask],1,F)
lines1=sfmnp.ComputeEpiline(img2pts[mask],2,F)

import numpy as np
tup = ut.drawlines(img2,img1,lines2,img2pts[mask],img1pts[mask],drawOnly=10,
    linesize=10,circlesize=30)
epilines2 = np.concatenate(tup[:-1],axis=1) #reversing the order of left and right images

plt.figure(figsize=(9,4))
plt.imshow(epilines2)

tup = ut.drawlines(img1,img2,lines1,img1pts[mask],img2pts[mask],drawOnly=10,
    linesize=10,circlesize=30)
epilines1 = np.concatenate(tup,axis=1)

plt.figure(figsize=(9,4))
plt.imshow(epilines1)
plt.show()

```

3.3 Estimating Essential Matrix from Fundamental Matrix

With the computed fundamental matrix F , we can derive the essential matrix E . The essential matrix is another 3 by 3 matrix with additional properties that relates the corresponding points. E can be formulated as Eq.6.

$$E = K^T F K$$

- K is the intrinsic camera matrix
- F is the fundamental matrix

Eq.6

3.3.1 Finding Intrinsic Camera Calibration Matrix

The intrinsic camera matrix, K consists of the internal parameters of the camera that is unique to each camera. Once calculated, it can be reused on other images to determine the essential matrix taken by the same camera. The matrix is a 3 by 3 matrix as shown in Eq.7.

$$\text{Intrinsic Camera Matrix} = \begin{bmatrix} f_z & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Eq.7

- f_z and f_y are the focal length in terms of pixels along the x and y axis respectively
- c_x and c_y are the optical centers which is typically close to the center of the image
- s is the skew factor between the x and y axis, which is zero in most modern cameras. If it is non-zero, it means the image axes are not perpendicular

To determine these parameters, we can calibrate the camera using images of a well-defined pattern, such as a checkerboard. Specifically, a 9x7 checkerboard pattern was used, and images of this pattern were captured from various viewpoints, as shown in Table 3. From these images, the algorithm identifies object points and image points. The object points represent 3D coordinates in real-world space, while the image points correspond to their 2D projections on the image plane. Using the known object points, image points, and image frame size, these inputs are passed to the inbuilt OpenCV `calibrateCamera` function to compute the intrinsic camera matrix. An example of a generated camera matrix from a mobile camera is shown below.

$$\text{Intrinsic Camera Matrix} = \begin{bmatrix} 2862.07 & 0 & 1802.81 \\ 0 & 2860.28 & 1363.97 \\ 0 & 0 & 1 \end{bmatrix}$$

Table 3: Checkerboard pattern retrieved from [5] captured from different viewpoints with corners drawn during processing



Table 4: Code Snippet for finding the intrinsic camera calibration matrix

```
import numpy as np
import cv2 as cv
import glob
import pickle
import os

#FIND CHESSBOARD CORNERS - OBJECT POINTS AND IMAGE POINTS #

chessboardSize = (8,6)
frameSize = (3648,2736)

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....., (6,5,0)
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:chessboardSize[0], 0:chessboardSize[1]].T.reshape(-1, 2)

size_of_chessboard_squares_mm = 20
objp = objp * size_of_chessboard_squares_mm

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('cameraCalibration/images/*.jpeg')
output_dir = 'cameraCalibration/output_images'
```

```

for image in images:

    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)

    # If found, add object points, image points (after refining them)
    if ret == True:

        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)

        # Draw and display the corners
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(3000)

        # Save the processed image with corners drawn
        image_name = os.path.basename(image) # Get the image filename
        output_image_path = os.path.join(output_dir, image_name)
        cv.imwrite(output_image_path, img) # Save the image

cv.destroyAllWindows()

#CALIBRATION #

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, frameSize, None, None)

pickle.dump((cameraMatrix, dist), open( "calibration.pkl", "wb" ))
pickle.dump(cameraMatrix, open( "cameraMatrix.pkl", "wb" ))

```

With the known fundamental matrix, F and intrinsic camera matrix, K , we will be able to calculate the essential matrix, E .

Table 5: Code Snippet for calculating essential matrix

```

K = np.array([[2862.07,0,1802.81],[0,2860.28,1363.97],[0,0,1]])
E = K.T.dot(F.dot(K))

```

3.4 Estimate Camera Pose from Essential Matrix

With the essential matrix defined, we will be able to find the possible camera poses. The camera pose consists of the rotation (Roll, Pitch, Yaw) and translation (X, Y, Z) values of the camera with respect to the world. By decomposing the essential matrix, we will be able to identify four possible camera poses, each formed by different combinations of rotation and translation. To achieve this, the essential matrix, E is factorized into three matrices using singular value decomposition (SVD).

$$E = UDV^T$$

Eq.8

- U and V are orthogonal matrices

- D is a diagonal matrix containing the singular values. For example, $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

An additional helper matrix W is defined that helps to introduce a 90° rotation in the plane and is used to formulate the possible rotation equations.

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Eq.9

Using the formulated equations, we can determine the possible rotation and translation matrixes. The translation matrix, t can be derived by taking the last column of U , and its direction can be either positive or negative. For the rotation matrix, two possible rotation matrixes can be derived using W and W^T . These combinations lead to four possible camera poses, as illustrated in Table 6. The equations for the four possible camera poses are as follows:

$$C_1 = U[:,3], \quad R_1 = UWV^T$$

Eq.10

$$C_2 = -U[:,3], \quad R_2 = UWV^T$$

Eq.11

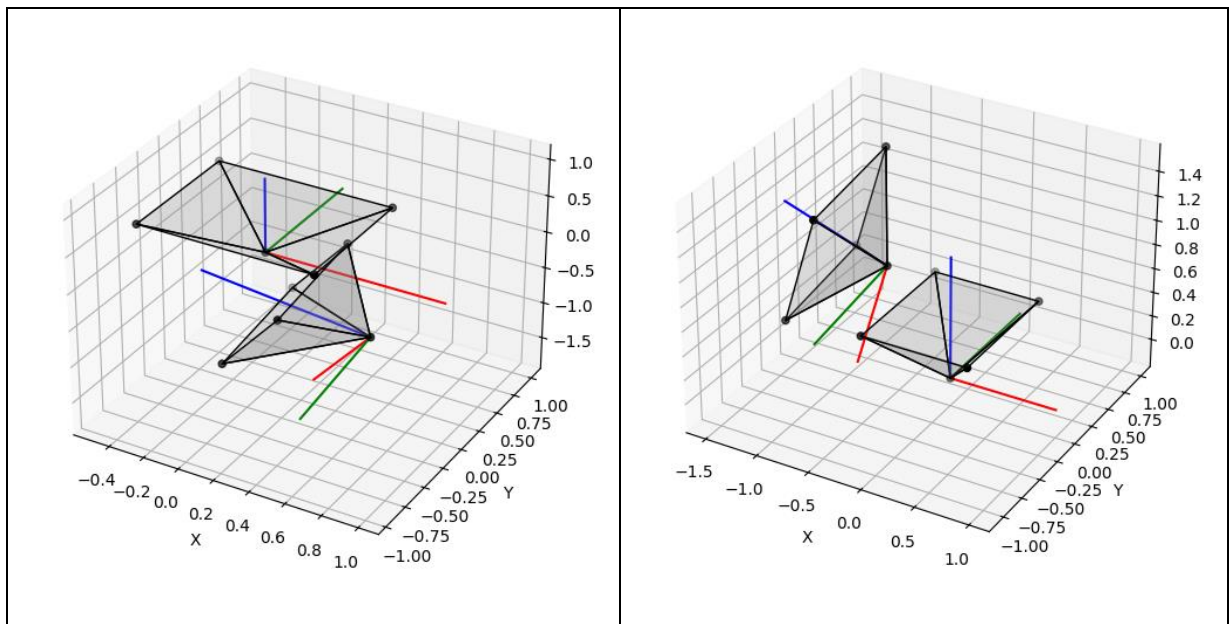
$$C_3 = U[:,3], \quad R_3 = UW^TV^T$$

Eq.12

$$C_4 = -U[:,3], \quad R_4 = UW^TV^T$$

Eq.13

Table 6: Four possible camera poses decompose from Essential Matrix



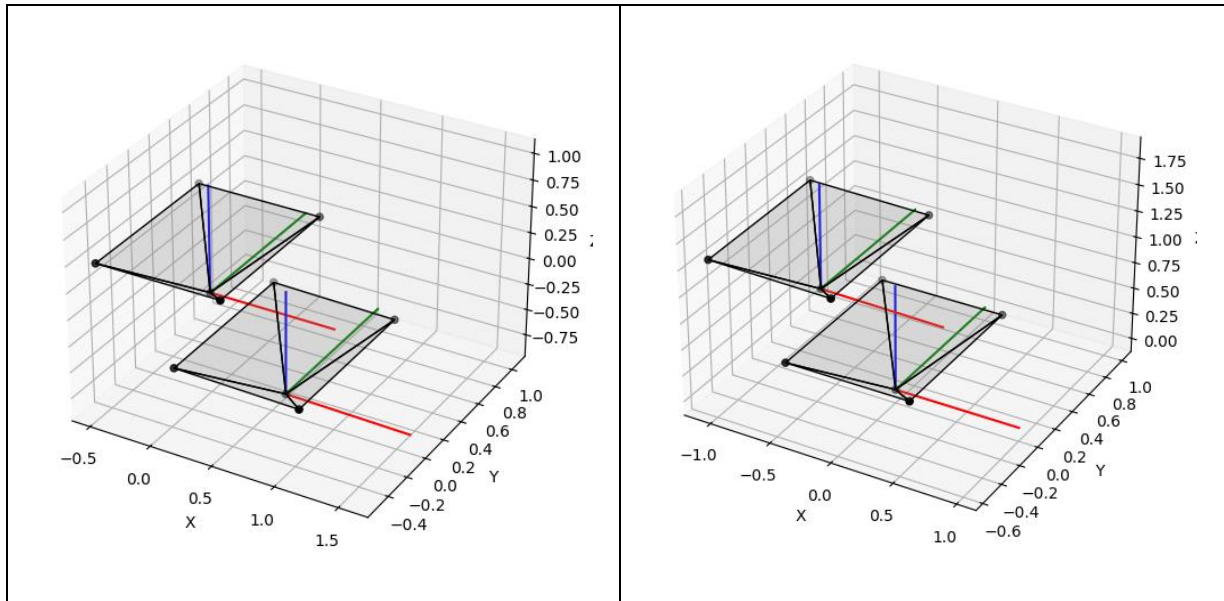


Table 7: Code Snippet for finding and visualizing the four possible camera poses

```
R1,R2,t = sfmnp.ExtractCameraPoses(E)
t = t[:,np.newaxis]

def ExtractCameraPoses(E):
    u,d,v = np.linalg.svd(E)
    W = np.array([[0,-1,0],[1,0,0],[0,0,1]])

    #Translation matrix
    t = u[:,-1]
    #Two Rotation Matrix
    R1 = u.dot(W.dot(v))
    R2 = u.dot(W.T.dot(v))

    if np.linalg.det(R1) < 0:
        R1 = R1 * -1

    if np.linalg.det(R2) < 0:
        R2 = R2 * -1

    return R1,R2,t

for R_ in [R1,R2]:
    for t_ in [t,-t]:

        fig = plt.figure(figsize=(9,6))
        ax = fig.add_subplot(111, projection='3d')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')

        ut.PlotCamera(np.eye(3,3),np.zeros((3,)),ax)
        ut.PlotCamera(R_,t_[:,0],ax)
```

3.5 Check for Cheirality Condition using Triangulation

Using the essential matrix, we have computed four different possible camera poses. However, there is only one camera pose that should be correct. To identify the correct pose, we need to first perform triangulation. Triangulation is the method in which we determine the 3D coordinates of a point in real world space given the 2D coordinates from our image plane and the camera positions. From Table 8, the triangulation plots illustrate the results for all four possible poses. The top-left and top-right plots show no 3D points, indicating invalid poses. The bottom-left plot displays a large number of points incorrectly located behind the camera, further eliminating it as a candidate. The bottom-right plot, on the other hand, shows a dense cluster of 3D points correctly positioned in front of the camera suggesting that this might be the correct camera pose.

Table 8: Triangulation plot of all four possible camera poses with the 3D points

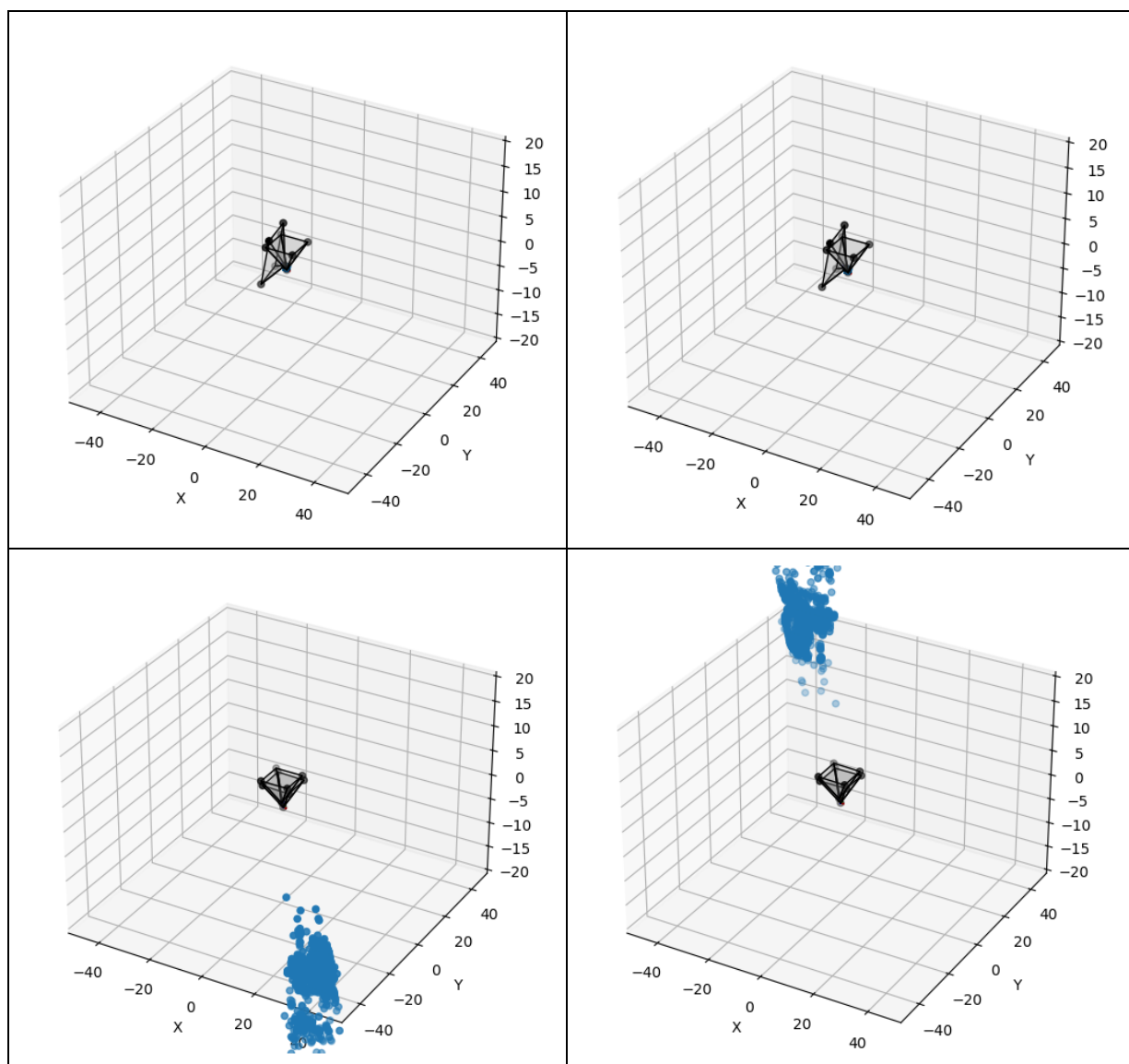


Table 9: Code Snippet for triangulation of 3D points and plotting the cameras with the 3D points

```
def GetTriangulatedPts(img1pts,img2pts,K,R,t,triangulateFunc):
    img1ptsHom = cv2.convertPointsToHomogeneous(img1pts)[:,:0,:]
    img2ptsHom = cv2.convertPointsToHomogeneous(img2pts)[:,:0,:]

    img1ptsNorm = (np.linalg.inv(K).dot(img1ptsHom.T)).T
    img2ptsNorm = (np.linalg.inv(K).dot(img2ptsHom.T)).T

    img1ptsNorm = cv2.convertPointsFromHomogeneous(img1ptsNorm)[:,:0,:]
    img2ptsNorm = cv2.convertPointsFromHomogeneous(img2ptsNorm)[:,:0,:]

    pts4d = triangulateFunc(np.eye(3,4),np.hstack((R,t)),img1ptsNorm.T,img2ptsNorm.T)
    pts3d = cv2.convertPointsFromHomogeneous(pts4d.T)[:,:0,:]

    return pts3d

configSet = [None,None,None,None]
configSet[0] = (R1,t,GetTriangulatedPts(img1pts[mask],img2pts[mask],K,R1,t,cv2.triangulatePoints))
configSet[1] = (R1,-t,GetTriangulatedPts(img1pts[mask],img2pts[mask],K,R1,-t,cv2.triangulatePoints))
configSet[2] = (R2,t,GetTriangulatedPts(img1pts[mask],img2pts[mask],K,R2,t,cv2.triangulatePoints))
configSet[3] = (R2,-t,GetTriangulatedPts(img1pts[mask],img2pts[mask],K,R2,-t,cv2.triangulatePoints))

for cs in configSet:
    fig = plt.figure(figsize=(9,6))
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    ut.PlotCamera(np.eye(3,3),np.zeros((3,)),ax,scale=5,depth=5)
    ut.PlotCamera(cs[0],cs[1][:,0],ax,scale=5,depth=5)

    pts3d = cs[-1]
    ax.scatter3D(pts3d[:,0],pts3d[:,1],pts3d[:,2])

    ax.set_xlim(left=-50,right=50)
    ax.set_ylim(bottom=-50,top=50)
    ax.set_zlim(bottom=-20,top=20)
```

To further verify, with the computed 3D points, we can check for the cheirality condition. This condition states that all 3D points should lie in front of the camera. We will need to compare all four camera poses by counting the number of points that satisfy the cheirality condition, the camera with the highest number of such points is the best camera pose. Based on the analysis, the bottom-right plot was indeed the best camera pose.

Table 10: Code Snippet for checking for chierality condition to find the best camera pose

```
def DisambiguateCameraPose(configSet):
    maxfrontpts = -1
    bestIndex = -1 # To store the index of the best configuration
    bestConfig = None # To store the best configuration set entry

    for idx, (R, t, pts3d) in enumerate(configSet):
        count = CountFrontOfBothCameras(pts3d, R, t)

        if count > maxfrontpts:
            maxfrontpts = count
            bestR, bestt = R, t
            bestIndex = idx
```

```

        bestConfig = (R, t, pts3d)

    return bestR, bestt, maxfrontpts, bestIndex, bestConfig

def CountFrontOfBothCameras(X, R, t):
    isfrontcam1 = X[:, -1] > 0
    isfrontcam2 = TransformCoordPts(X, R, t)[:, -1] > 0

    return np.sum(isfrontcam1 & isfrontcam2)

def TransformCoordPts(X,R,t):
    return (R.dot(X.T)+t).T

# Assuming configSet is already defined
R, t, count, bestIndex, bestConfig = DisambiguateCameraPose(configSet)

print(f"Best Camera Pose Index: {bestIndex}")

```

3.6 Perspective-N-Point

In the previous sections, we matched two images to perform 3D reconstruction. However, two images are often insufficient to fully reconstruct a structure in 3D. Typically, a larger number of images are required. For each new image, the camera pose must be estimated relative to the existing 3D model reconstructed from earlier images. This process introduces a concept known as perspective-n-points (PnP). The PnP algorithm helps to estimate the pose of the new camera relative to the existing 3D scene as illustrated in Figure 8. This step is crucial in extending the 3D scene reconstruction incrementally in an SfM pipeline. Before employing the PnP algorithm, two key inputs are required: corresponding 2D points from the new image and 3D points from the existing model. The code snippet shows that the new image undergoes feature detection using SIFT to generate keypoints and descriptors. These descriptors are matched with those from previous images to identify corresponding 2D and 3D points. The matched 2D and 3D points are then filtered and used as inputs for the PnP algorithm.

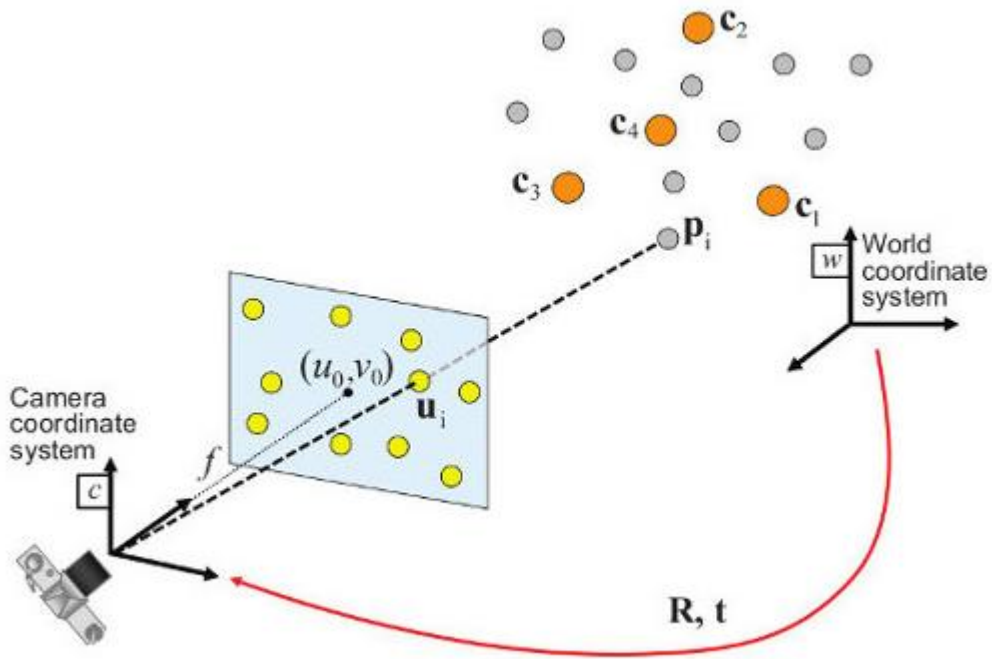


Fig 8: Concept of perspective-n-points. Retrieved from [4]

Table 11: Code Snippet for finding the 2D and 3D point correspondence from new image

```
img3 = cv2.imread('/content/drive/MyDrive/Structure7/03.jpeg')
img3 = img3[:, :, :-1]
surfer=cv2.SIFT_create()
kp3, desc3 = surfer.detectAndCompute(img3,None)

plt.figure()
plt.imshow(img3)

def Find2D3DMatches(desc1,img1idx,desc2,img2idx,desc3,kp3,mask,pts3d):
    #Picking only those descriptors for which 3D point is available
    desc1_3D = desc1[img1idx][mask]
    desc2_3D = desc2[img2idx][mask]

    matcher = cv2.BFMatcher(crossCheck=True)
    matches = matcher.match(desc3, np.concatenate((desc1_3D,desc2_3D),axis=0))

    #Filtering out matched 2D keypoints from new view
    img3idx = np.array([m.queryIdx for m in matches])
    kp3_ = (np.array(kp3))[img3idx]
    img3pts = np.array([kp.pt for kp in kp3_])

    #Filtering out matched 3D already triangulated points
    pts3didx = np.array([m.trainIdx for m in matches])
    pts3didx[pts3didx >= pts3d.shape[0]] = pts3didx[pts3didx >= pts3d.shape[0]] - pts3d.shape[0]
    pts3d_ = pts3d[pts3didx]

    return img3pts, pts3d_

img3pts,pts3dpts = Find2D3DMatches(desc1,img1idx,desc2,img2idx,desc3,kp3,mask,pts3d)
```

With the known 2D and 3D points, we employed OpenCV's solvePnPRansac algorithm to estimate the rotation and translation vectors for the new camera. The integration of the RANSAC method with PnP ensures robust handling of outliers because PnP is prone to errors if there are outliers in the set of point correspondences. Once the rotation and translation vectors are computed, we can use them to plot and visualize the new camera pose. Figure 9 illustrate the new camera pose highlighted in red with the existing two camera pose.

Table 12. Code Snippet for employing PnP to find the rotation and translation of new camera pose and visualizing the camera poses

```
retval,Rvec,tnew,mask3gt = cv2.solvePnPRansac(pts3dpts[:,np.newaxis],img3pts[:,np.newaxis],
                                             K,None,confidence=.99,flags=cv2.SOLVEPNP_DLS)
Rnew,_=cv2.Rodrigues(Rvec)
tnew = tnew[:,0]

fig = plt.figure(figsize=(9,6))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

ut.PlotCamera(np.eye(3,3),np.zeros((3,)),ax)
ut.PlotCamera(R,t[:,0],ax)
ut.PlotCamera(Rnew,tnew,ax,faceColor='red')
```

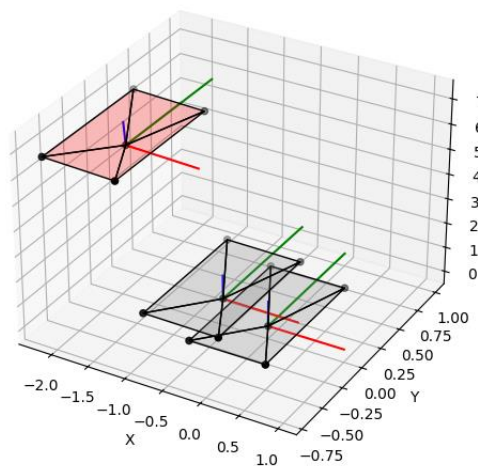


Fig 9: New camera pose (Highlighted in Red) plotted with existing two camera pose

With the new camera pose known, we can triangulate new 3D points and add it to the existing 3D model, incrementally building the scene. Table 14 shows the sparse 3D reconstruction base on three images of the hive. We can observe that the outline of the hive structure is visible, but the number of points is insufficient to capture the finer details of the hive. By incorporating more images, the 3D reconstruction can be incrementally refined, enabling us to capture most, if not all, details of the structure. The results of the hive is detailed in Section 4.1.

Table 13: Code Snippet for triangulating new points and adding the new points to the existing 3D model

```

kpNew, descNew = kp3, desc3

kpOld, descOld = kp1, desc1
ROld, tOld = np.eye(3), np.zeros((3,1))

accPts = []
for (ROld, tOld, kpOld, descOld) in [(np.eye(3), np.zeros((3,1)), kp1, desc1), (R, t, kp2, desc2)]:

    #Matching between old view and newly registered view..
    print('[Info]: Feature Matching..')
    matcher = cv2.BFMatcher(crossCheck=True)
    matches = matcher.match(descOld, desc3)
    matches = sorted(matches, key = lambda x:x.distance)
    imgOldPts, imgNewPts, _, _ = ut.GetAlignedMatches(kpOld, descOld, kpNew,
                                                    descNew, matches)

    #Pruning the matches using fundamental matrix..
    print('[Info]: Pruning the Matches..')
    F, maskt = cv2.findFundamentalMat(imgOldPts, imgNewPts, method=cv2.FM_RANSAC, ransacReprojThreshold=.1, confidence=.99)
    maskt = maskt.flatten().astype(bool)
    imgOldPts = imgOldPts[maskt]
    imgNewPts = imgNewPts[maskt]

    #Triangulating new points
    print('[Info]: Triangulating..')
    newPts = sfmnp.GetTriangulatedPts(imgOldPts, imgNewPts, K,
    Rnew, tnew[:, np.newaxis], cv2.triangulatePoints, ROld, tOld)

    #Adding newly triangulated points to the collection
    accPts.append(newPts)

#Adding the original 2-view-sfm point cloud and saving the whole collection
accPts.append(pts3d)

def pts2ply(pts, filename='out.ply'):
    f = open(filename, 'w')
    f.write('ply\n')
    f.write('format ascii 1.0\n')
    f.write('element vertex {}'.format(pts.shape[0]))

    f.write('property float x\n')
    f.write('property float y\n')
    f.write('property float z\n')

    f.write('property uchar red\n')
    f.write('property uchar green\n')
    f.write('property uchar blue\n')

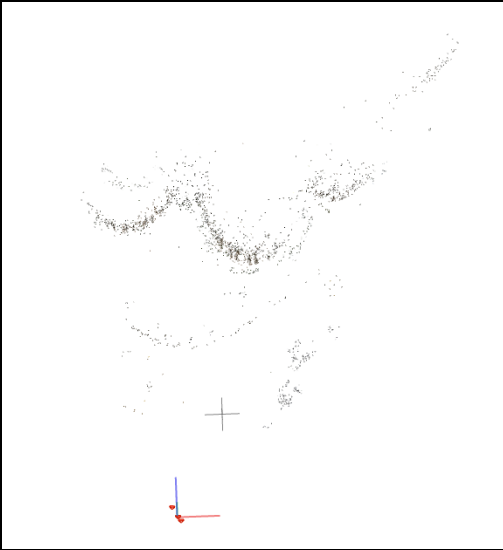
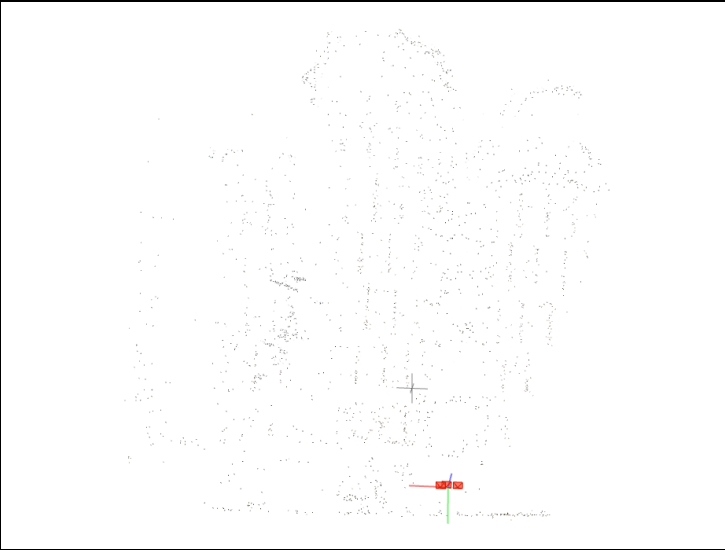
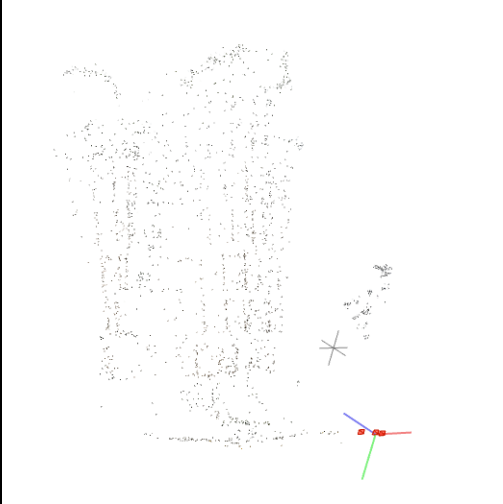
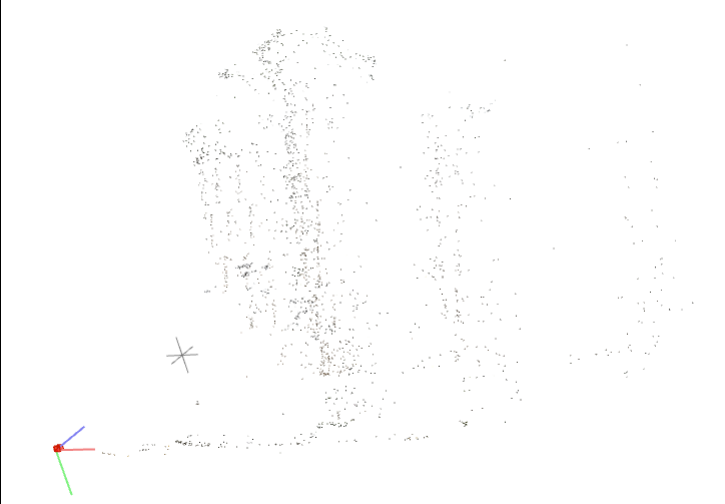
    f.write('end_header\n')

    for pt in pts:
        f.write('{} {} {} 255 255 255\n'.format(pt[0], pt[1], pt[2]))
    f.close()

pts2ply(np.concatenate((accPts), axis=0), 'hive.ply')

```


Table 14: Top, Front, 45° left, 45° right view of sparse 3D reconstruction of the Hive generated from 3 images. Red markers indicate camera positions.

Top View	Front View
	
45° Left View	45° Right View
	

4. Sparse Model Result and Discussion

4.1 Experiment Results of the Hive

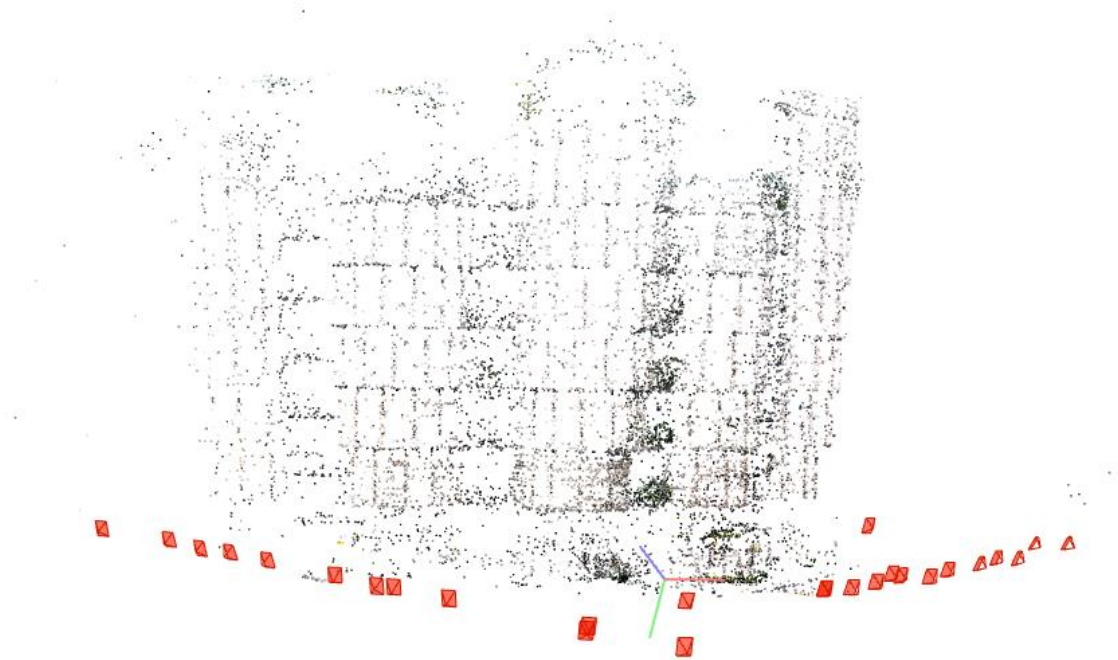


Fig 10: The Structure-from-Motion (SfM) reconstruction result derived from 28 photographs of the Hive.

The point cloud in the figure above depicts the building's three-dimensional structure, with individual points representing reconstructed spatial features. Red markers indicate camera positions and orientations used for capturing the images, while axes at the center represent the coordinate system's orientation.

For the SfM reconstruction of the Hive, we captured 28 photographs from various positions. However, due to physical and equipment limitations, we were restricted to taking photos at ground level along the two accessible roads, covering at most 180 degrees of the structure. The other half of the building could not be covered due to obstructions caused by adjacent buildings. Additionally, we did not use a drone to capture aerial views of the structure. The resulting SfM model accurately predicted our camera positions, as indicated by the red markers in Figure 10, which are limited to ground-level locations.

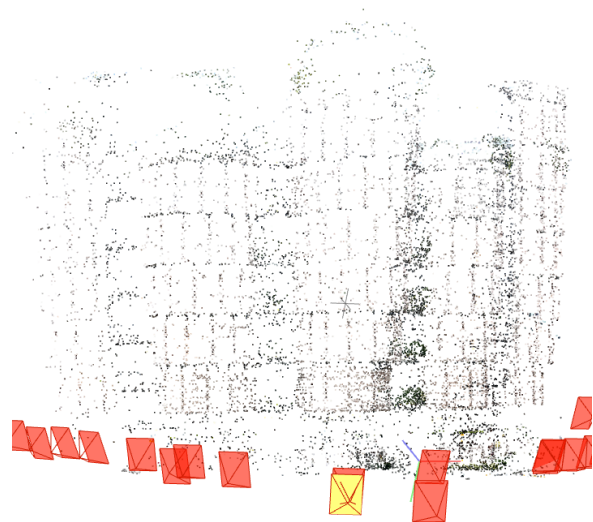


Fig 11: Composite image comparing the input photographs of the Hive (left) with their corresponding camera positions and orientations (yellow markers) estimated during the Structure-from-Motion (SfM) process (right). The top and bottom images showcase different perspectives of the Hive, with the dense point cloud reconstruction illustrating the building's unique architectural details.

As observed in Figure 11, the dense point cloud on the right effectively represents the Hive's structure and architectural features, including its distinctive cylindrical form and surrounding vegetation. The distribution of the red markers, representing additional camera positions, shows systematic coverage of the scene from multiple perspectives. This multi-view coverage likely contributed to the completeness and accuracy of the 3D reconstruction, capturing fine geometric details and spatial relationships within the scene.

Overall, the results highlight the strength of SfM in reconstructing complex structures like the Hive. The accurate alignment between the input images and the reconstructed model, along with the detailed representation of architectural features, underscores the robustness of the method. This process not only provides a faithful 3D representation but also validates the capability of SfM to handle intricate and geometrically challenging designs.

We further tested our SfM pipeline with other scenes within the NTU area.

4.2 Experiment Results of “The Wind and Wings” sculpture

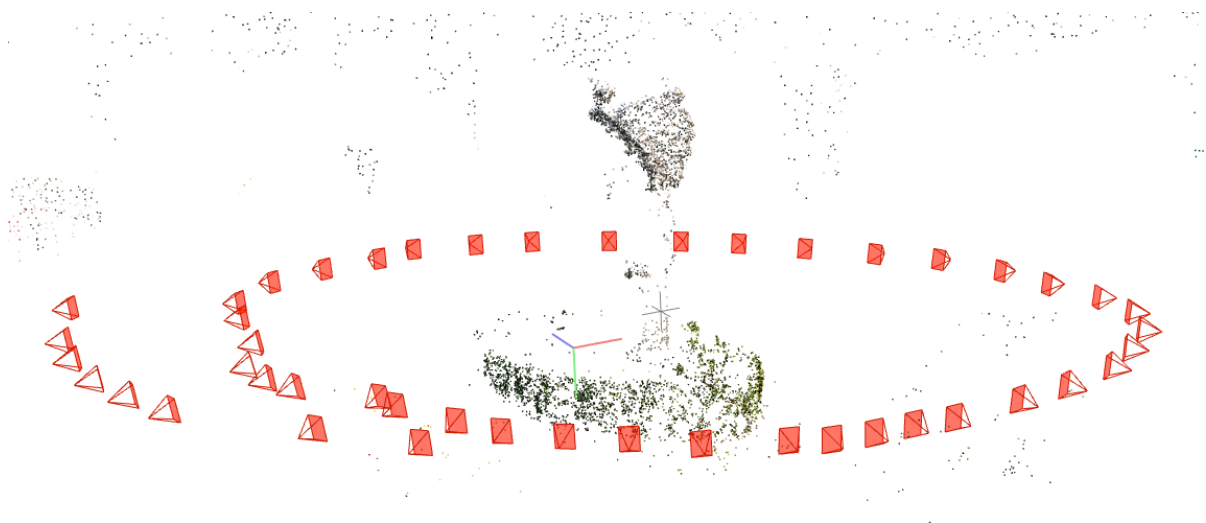


Fig 12: The Structure-from-Motion (SfM) reconstruction results generated from 47 photographs of the “The Wind and Wings” sculpture.

The point cloud in the figure above represents the sculpture's three-dimensional structure, with individual points corresponding to reconstructed spatial features. The red markers denote the camera positions and orientations used during image capture, while the axes at the center indicate the orientation of the coordinate system.

In this example, we experimented with the SfM pipeline using a smaller object where all photo positions, including the top section, could be covered without the need for a drone. The resulting model demonstrates that the pipeline successfully mapped the 3D positions of the features. As shown in Figure 12, top wing-like structure is accurately modelled, and the bottom bushes are also well-represented. However, the middle stem contains fewer feature points, possibly due to a lack of distinct features or colour contrast, which made it challenging for the pipeline to capture effectively.

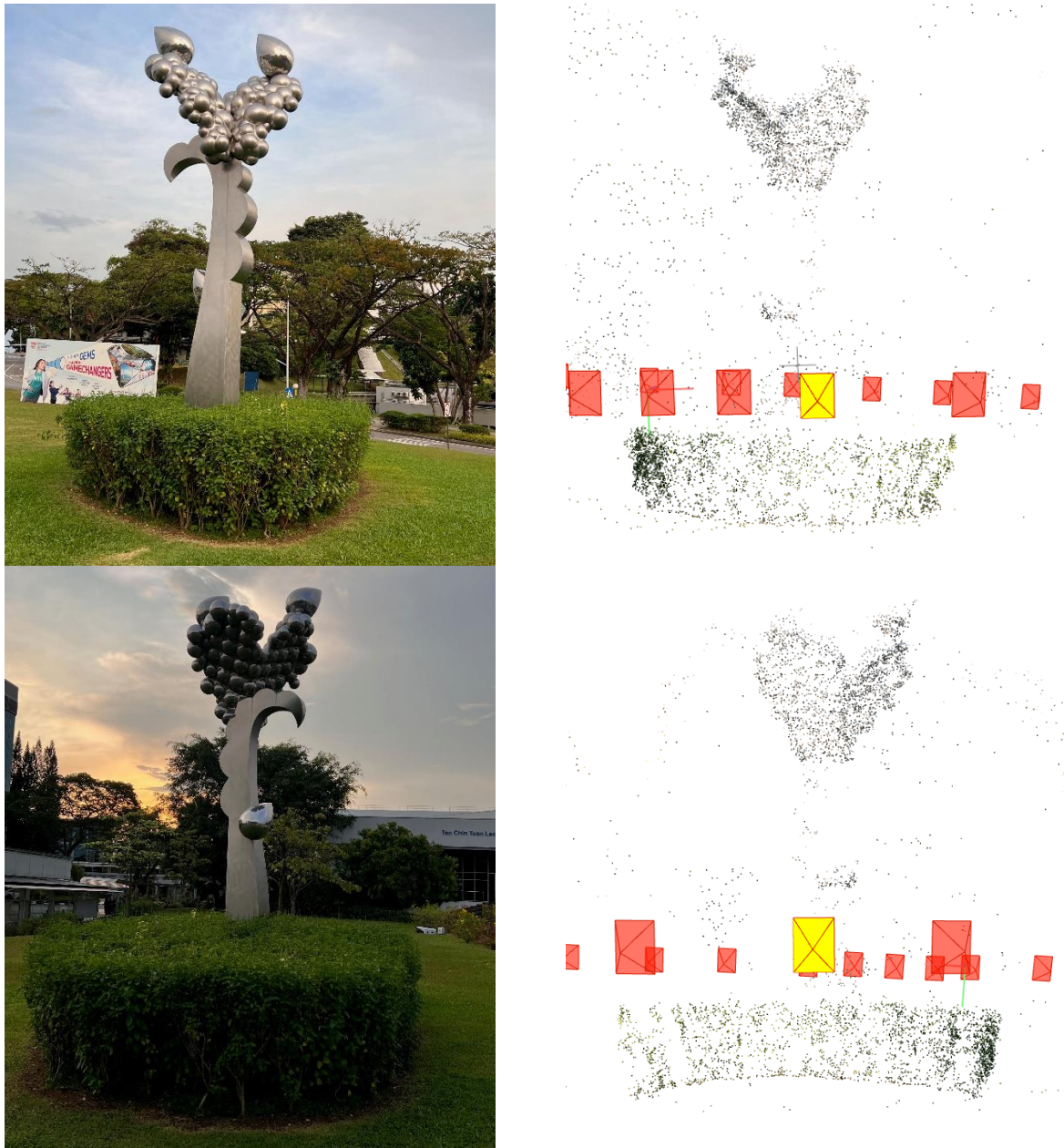


Fig 13: This composite image contrasts the input photographs of the sculpture (left) with the corresponding camera positions and orientations (yellow markers) estimated through the Structure-from-Motion (SfM) process (right). The top and bottom rows present different perspectives of the sculpture, with the dense point cloud reconstruction effectively capturing its intricate details.

The Structure-from-Motion (SfM) results demonstrate a successful reconstruction of the object, capturing key architectural and natural features. The top portion of the structure, characterized by its reflective and wing-like shape, is well-represented in the point cloud, indicating that the SfM pipeline handled these features with precision. Similarly, the rounded bush at the base is effectively reconstructed, showing the pipeline's capability in handling vegetative details. However, the central stem exhibits a sparser point distribution. This limitation is likely due to its smooth surface and lack of distinct texture or color contrast, which reduced the number of detectable feature points.

Overall, the results highlight the SfM pipeline's ability to create a detailed 3D representation when provided with comprehensive image coverage. The minor shortcomings in reconstructing the featureless central stem underscore the importance of surface texture and contrast in achieving higher point density. Despite this, the successful modeling of complex and reflective features, as well as natural elements, indicates the robustness of the pipeline in handling diverse objects.

4.3 Experiment Results of Nanyang University Memorial

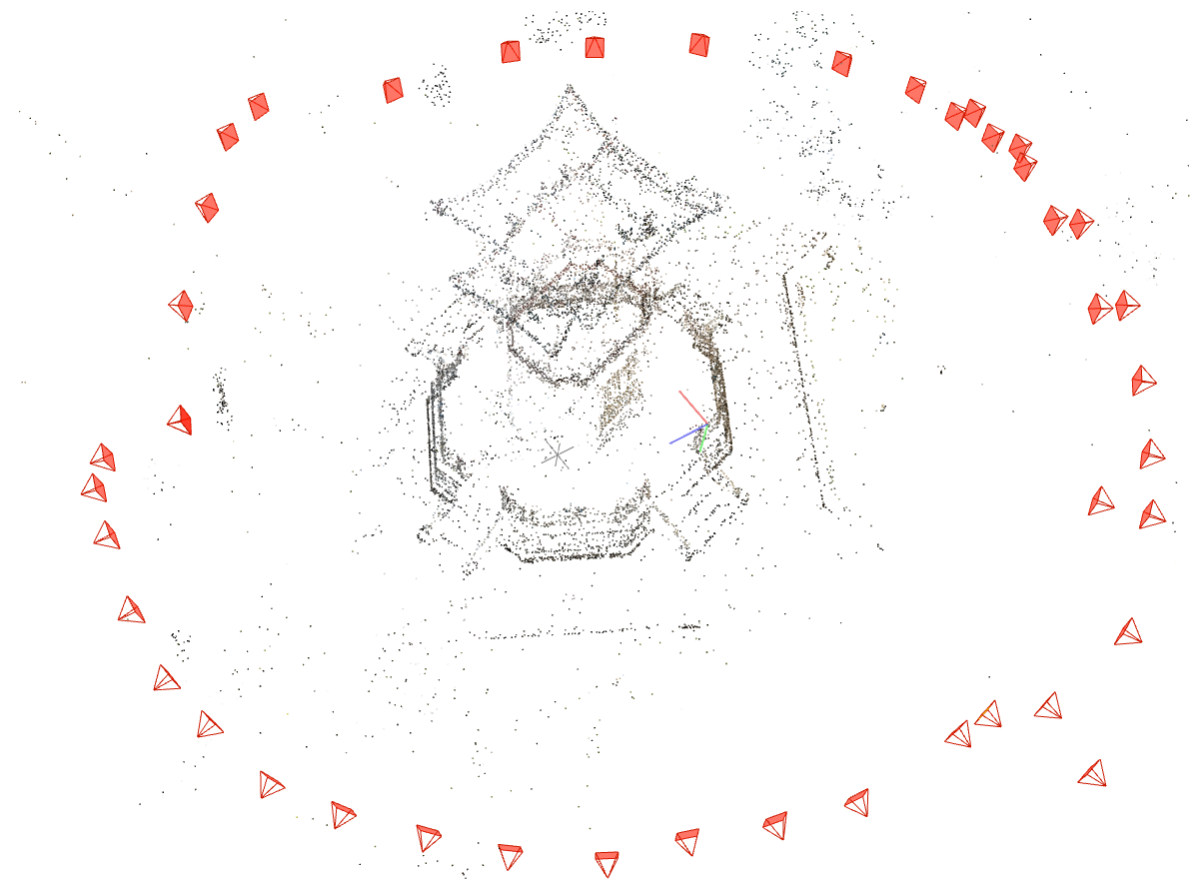


Fig 14: This image showcases the Structure-from-Motion (SfM) reconstruction result derived from 44 photographs of Nanyang University Memorial.

The point cloud in the figure above depicts the building's three-dimensional structure, with individual points representing reconstructed spatial features. Red markers indicate camera positions and orientations used for capturing the images, while axes at the center represent the coordinate system's orientation.

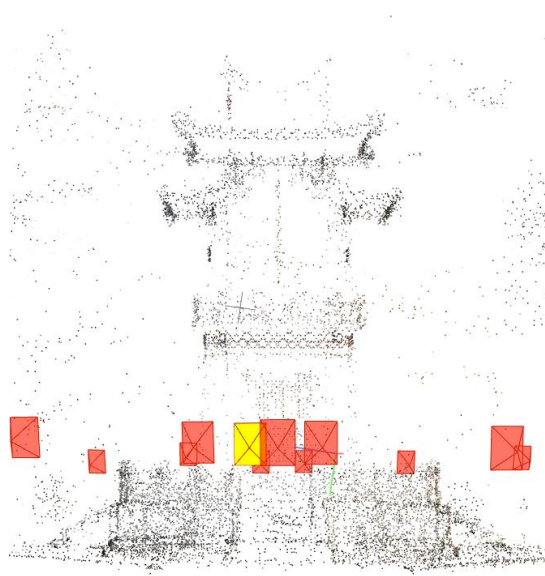
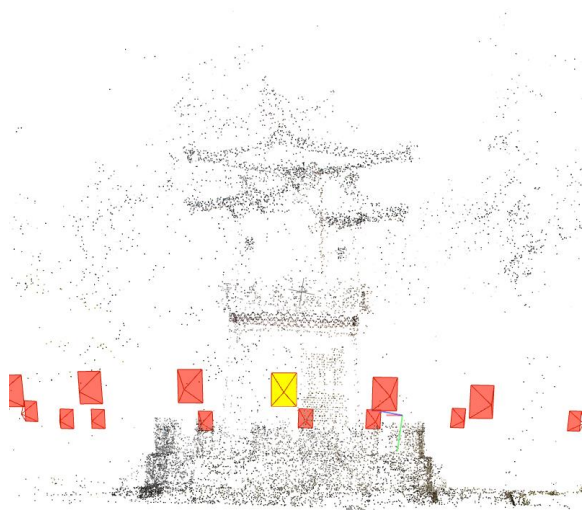


Fig 15: This composite image compares the input photographs of Nanyang University Memorial (left) with their corresponding camera positions and orientations (yellow markers) estimated during the Structure-from-Motion (SfM) process (right). The top and bottom rows showcase different perspectives of the monument, with the dense point cloud reconstruction illustrating the monument's unique details.

The Structure-from-Motion (SfM) results for the monument reveal several insights into the effectiveness and limitations of the reconstruction process. The circular placement of the cameras, as shown in the Figure 14, allowed the SfM pipeline to capture the monument from multiple angles, ensuring a comprehensive 360-degree coverage. This configuration enabled the pipeline to accurately reconstruct prominent architectural features, such as the intricate, multi-tiered roof, which includes curved edges and decorative ridges, and the stepped base with staircases on all sides. These details are critical in validating the pipeline's ability to handle geometric complexity in three-dimensional space.

Despite the robust reconstruction of the base and roof, some challenges were observed in the middle section of the monument, particularly around the smooth, low-texture panels and reflective surfaces. These areas exhibit sparse point coverage, likely because they lacked distinct visual features or sharp color contrasts, which are essential for effective feature matching in the SfM process. Similarly, the highly detailed ornamental roof, while well-represented in most areas, may have suffered from slight point sparsity due to its steep angles and potential occlusion from other sections of the structure.

Another notable feature of the reconstruction is the inclusion of the surrounding environment. Although the monument is the primary focus, elements like the pathways, vegetation, and nearby objects are partially visible in the point cloud, providing context and enhancing spatial accuracy. This inclusion is an indicator of the pipeline's ability to process scenes with multiple layers of depth. Overall, while the SfM results successfully model the monument's core architectural features, they also underline the importance of feature-rich surfaces, diverse viewpoints, and sufficient texture contrast for achieving even more detailed and complete reconstructions.

5. COLMAP Dense Model

After completing the sparse reconstruction process using Structure from Motion (SfM), we extend the project works into building a dense 3D model. By leveraging the COLMAP engine, specifically its pixelwise view selection for unstructured Multi-view Stereo (MVS) functionality, to create a detailed representation of the scene and target object. It is a method uses both photometric (color) and geometric information to accurately and efficiently build dense 3D models from sets of images. The algorithm employs a variant of the Generalized Expectation-Maximization (GEM) algorithm to iteratively refine the depth, normal, and occlusion estimates.

Key concepts:

- **Pixelwise View Selection:** Instead of pre-selecting source images, the algorithm performs pixel-level view selection based on both photometric and geometric priors. This allows for more robust and accurate reconstruction, especially in unstructured datasets where viewing geometry and image resolutions can vary significantly.
- **Joint Estimation:** The algorithm jointly estimates depth and surface normals, using the estimated normals to improve the accuracy of depth estimation through the use of slanted plane-induced homographs.
- **Image-Based Fusion and Filtering:** A directed graph of consistent pixels is created based on photometric and geometric support. This graph is then used to filter outliers and fuse depth and normal estimates from multiple views into a consistent 3D point cloud


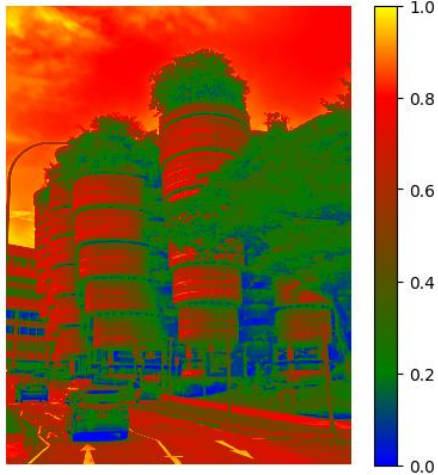

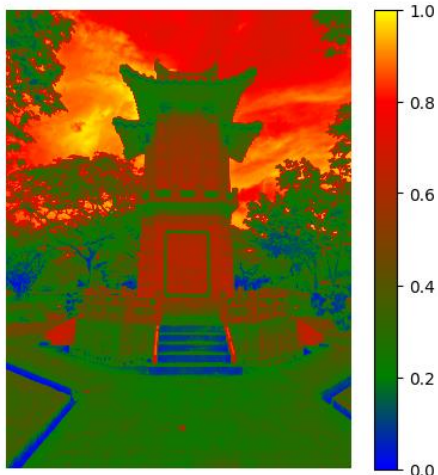
5.1 Photometric Information

Color Similarity: The algorithm compares patches in the reference image and source images using a method called Normalized Cross-Correlation (NCC). This measures how similar the colors are, helping to identify which source images are best for estimating depth.

Improved Accuracy with Weighted NCC: To handle areas with sharp depth changes (like edges), a special version of NCC is used. It gives more weight to pixels based on both their color similarity and spatial closeness, reducing blur near depth boundaries.

Occlusion Handling: The algorithm uses indicators (binary variables) to check if parts of the source image are hidden (occluded). This ensures only visible areas are used for accurate depth estimation.

Table 15: Comparison of original image and photometric information

Original image	Photometric Information
	
	



5.2 Geometric Information

Depth and Normals: The algorithm calculates both the depth and surface orientation (normals) for each pixel in the reference image. These are used to create better alignments (homographies) between images, improving accuracy compared to simpler methods.

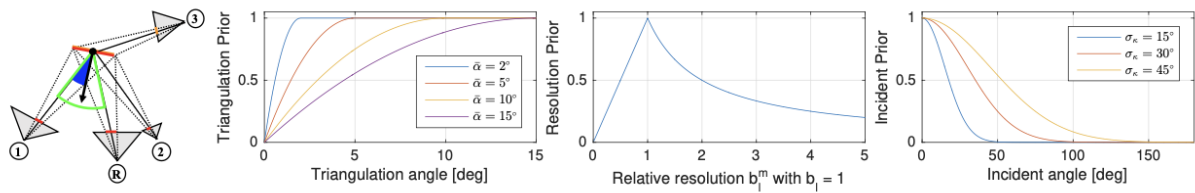


Fig 16: Left: Illustration of geometric priors for reference view (R) and three source views (1-3). View 1 has similar resolution (red), and good triangulation (green) and incident angle (blue), while view 2 is oblique and has lower resolution. View 3 cannot see the patch. Right: Geometric prior likelihood functions with different parameters. (Screenshot of the paper, pixelwise view selection for MVS by Schonberger et al)

Geometric Priors for Image Selection (as show as Figure 16)

- **Triangulation Prior:** Chooses source images with good viewing angles for stable 3D reconstruction.
- **Resolution Prior:** Prefers source images with resolution similar to the reference image to avoid quality issues.
- **Incident Prior:** Selects images with direct (non-oblique) views to ensure the surface is clearly visible.

Geometric Consistency: A consistency check compares depth and surface orientation estimates across multiple views, reducing errors and ensuring the 3D reconstruction is complete and accurate.

5.3 Filtering and Fusion (Final step)

The final process involves filtering out outliers and fusing consistent pixel information to create a robust 3D point cloud from multiple images. Initially, depth and surface normals are estimated for each pixel, but outliers can occur due to noise, occlusions, or ambiguous textures. Filtering removes these outliers based on geometric and photometric consistency across multiple views. Pixels that lack sufficient support are eliminated, ensuring only reliable data is used. The remaining consistent pixels are fused into clusters, which are then used to generate a 3D point cloud. This process effectively leverages information from multiple views to create a detailed and robust 3D model, suitable for further processing such as coloring and meshing.

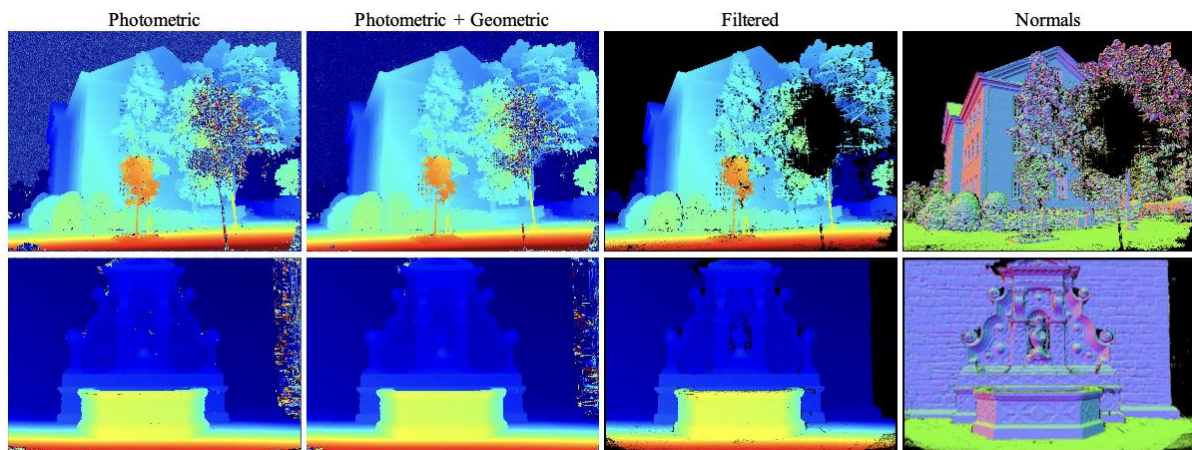


Fig 17: Process of robust 3D reconstruction: (1) Initial photometric depth estimates, (2) Combined photometric and geometric consistency, (3) Filtered results after outlier removal, (4) Final surface normals visualization, which can be further used for surface colouring and detailed 3D reconstruction. (Screenshot of the paper, pixelwise view selection for MVS, Schonberger et al)

5.4 Experiment Results

Table 16: The top row displays three sample input images of: (1) the Hive; (2) the "Wind and Wings" sculpture; and (3) the Nanyang National Memorial. The bottom row showcases the corresponding MVS dense models of these three scenes.





For all three scenes, the MVS pipeline utilized the same set of input images and the sparse point cloud generated by the Structure-from-Motion (SfM) model. The input images served as the primary data source for feature matching and depth estimation, while the sparse point cloud provided an initial geometric framework for reconstructing the dense 3D models. This consistent approach ensures that variations in reconstruction quality are attributable to the scene's characteristics.

The MVS dense reconstruction of the Hive captures the overall cylindrical shapes quite well, emphasizing the layered, textured façade of the building. The repetitive nature of the building's design, with its consistent horizontal grooves and vertical alignment, lends itself well to feature matching in the MVS process. This results in a relatively complete reconstruction, preserving the iconic form of the building. The vegetation at the top of the structure is less detailed in the dense reconstruction, likely due to the complexity and lack of distinct features in the greenery.

There are certain challenges in the reconstruction. Textureless or shadowed regions appear sparse or noisy, indicating areas where the MVS struggled to extract sufficient information. The base of the structure, which may include smaller architectural details or vehicles, appears less resolved compared to the upper portions of the building. These results highlight the strengths of MVS for geometrically repetitive and textured surfaces, but also its limitations in capturing fine elements without a higher density of input images.

The “Wind and Wings” sculpture, shaped like metallic grapes on a tall, curved stem, creates specific challenges for MVS dense reconstruction. Its smooth and shiny surface makes it harder for the system to detect and match features, resulting in a noisier and less detailed point cloud. The overall shape of the sculpture, including the grape-like clusters and the stem, is recognizable, but finer details, such as the smooth curves or small connections between parts, are not captured clearly.

Another issue is the impact of the surrounding environment on the reconstruction. The nearby plants and ground add noise to the point cloud, making it harder to focus on the sculpture itself. The reflective surface of the sculpture further confuses the MVS system, which depends on clear and consistent textures.

The MVS dense reconstruction does a good job capturing the main structure of the Nanyang National Memorial, including the multi-tiered roof, the plaques, and the steps leading to the base. The overall shape and key features are clear, making it easy to see the monument's design. However, smaller details, like the carvings on the roof and the writing on the plaques, are not as clear. This is likely because the system had trouble detecting and matching features that are small or lack of contrast.

The environment around the monument, such as trees and grass, adds some extra noise to the reconstruction, making it harder to focus only on the monument itself. This slightly affects the clarity of the model. To improve the results, using higher-quality images and capturing more angles of the monument would help. Reducing distractions from the background, like vegetation, would also make the reconstruction cleaner.

6. Conclusion

In this project, we successfully utilized Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipelines to reconstruct dense 3D models of three distinct structures on the NTU campus: the Hive, the “Wind and Wings” sculpture, and the Nanyang National Memorial. Each reconstruction highlighted the strengths and limitations of the employed methods, offering insights into their effectiveness under varying conditions. The results demonstrate the capability of SfM and MVS to produce detailed representations of complex geometries, while also emphasizing the impact of environmental factors, surface textures, and input image quality on the final outcomes.

The dense reconstructions captured the main architectural features of all three scenes, with the Hive’s cylindrical design, the sculpture’s fluid curves, and the monument’s intricate roof details being notable successes. However, challenges such as sparse coverage in textureless or reflective regions, environmental noise, and occlusions underline the need for improved input datasets and optimized pre-processing. Future work could focus on acquiring higher-resolution images, increasing image diversity, and exploring advanced algorithms to enhance feature matching and depth estimation. These improvements will enable more robust and comprehensive 3D reconstructions across diverse and complex scenes.

References

- [1] Johannes L. Schonberger, Enliang Zheng, Marc Pollefeys, Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo <https://demuc.de/papers/schoenberger2016mvs.pdf>
- [2] Schonberger J L, Frahm J M. Structure-from-motion revisited. CVPR, 2016: 4104-4113.
- [3] *OpenCV: Feature Matching*. (n.d.). Opencv.org. Retrieved November 19, 2024, from https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html
- [4] *OpenCV: Perspective-n-Point (PnP) pose computation*. (n.d.). Opencv.org. Retrieved November 19, 2024, from https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html
- [5] Jones, M. H. (2018, April 15). Calibration checkerboard collection. *Mark Hedley Jones*. <https://markhedleyjones.com/projects/calibration-checkerboard-collection>
- [6] Hartley, R. and Zisserman A., *Multiple view geometry in Computer Vision*, Cambridge University Press, New York. 2000
- [7] Aadil, M. (n.d.). *structure-from-motion: Structure-from-Motion from scratch, using Numpy and OpenCV*.