CSE 586: Computer Vision II

Computer Project # 2:

Augmented Reality through Multiview Stereo and SfM

Liming Gao, Chengyu Hu, Zekai Liu, Mingzhao Yu

Date:4/30/2020

A. Motivation:

Structure from Motion (SfM) is a common method of 3D reconstruction by utilizing the spatial and geometric relationship of the same target from cameras of different angles. Humans perceive a lot of information about the three-dimensional structure in their environment by moving around it. When the observer moves, objects around them move different amounts depending on their distance from the observer. This is known as motion parallax, and from this depth information can be used to generate an accurate 3D representation of the world around them. Inspired by SfM technique, we construct 3D representation of a specific scene and place virtual box in it in a well-designed way. Consequently, the projection of the virtual box in the camera will look realistic.

B. Approach:

1) Take photos

We used a cell phone camera to collect a set of images that capture a 3D scene in the corner of the living room from different angles, we manually placed a lot of features, such as boxes, paper-cuts, and posters on the scene, so as to facilitate COLMAP software to work easily.

Figure 1. A set of real-world Images

2) COLMAP

Fed all six images into the COLMAP software and did feature extraction, matching, incremental, and bundle adjustment to export the sparse 3D points in the interface, also exported the “cameras.txt”, “image.txt” and “points3D.txt” files for reconstructed model.

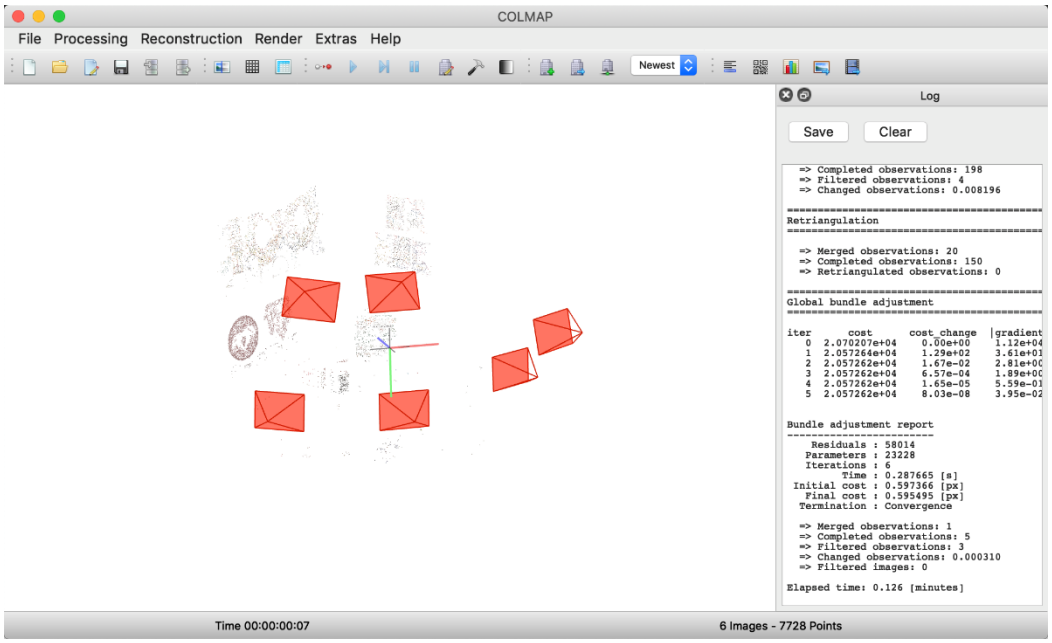


Figure 1 Sparse 3D points in scene

3) Read 3D point cloud stored in points3D.txt

A python code (Figure 2) is used to extract the needed X,Y,Z coordinates from points3D.txt for each point and store them into pointsXYZ.txt file. Thus, MATLAB can load the point cloud data directly.

```
with open('points3D.txt', 'r') as f:
    for i in f.readlines():
        n = list(map(float, i.split()))
        s = n[1], n[2], n[3]
        s = (' '.join(map(str, s)))
        print(s)

```

Figure 2 Python code parsing point cloud data

Final, 3747 3D points are obtained.

4) Plane Dominant Fit using RANSAC

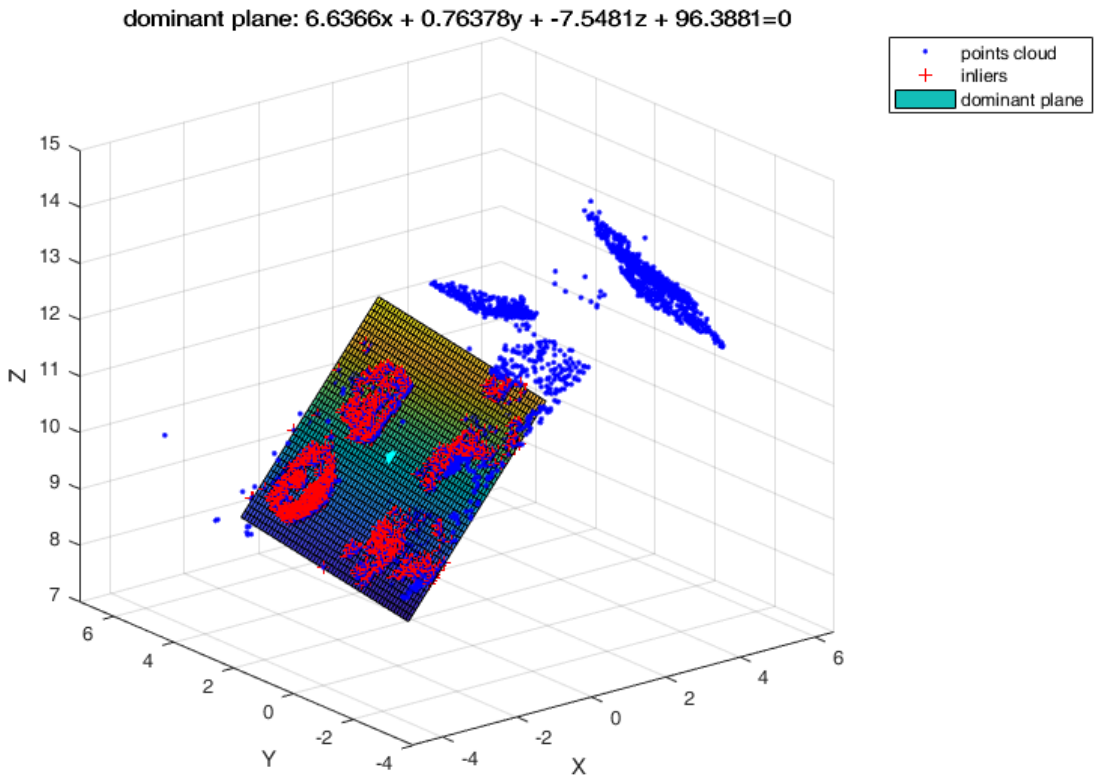
The main objective of this step is to write a RANSAC routine to find the largest subset of inliers after that find the dominant plane in this point cloud with sample consensus.

First, load the data in the above text “point3D.txt”, we only need the X, Y, Z coordinates for each world points. Instead of calculating the number of loops with a few parameters by the mathematical equation, we straightforward set the iterations to 10 times the total number of points. Then randomly select 3 points by using MATLAB function “randsample”, which arbitrary pick 3 index of points in the dataset to fit a 3D plane in one iteration. Take the cross product to get the normal vector of P2-P1 and P3-P1 (parameter A, B, C), then dot P1 to get the inverse (parameter D), and use these parameters to create the plane Ax + By + Cz + D = 0, which is the temporary plane we fit. By calculating the Euclidean distance between each point in the dataset and the plane, the batch of points whose distance below the threshold sigma is selected. Continually updated the dominant plane and the maximum number of inliers until the iteration stops.

5) Display the 3D point cloud and inlier points

Next, we use the MATLAB function “plot3” to plot the 3D point cloud generated by COLMAP, and denote the inliers by finding the index of corresponding points that

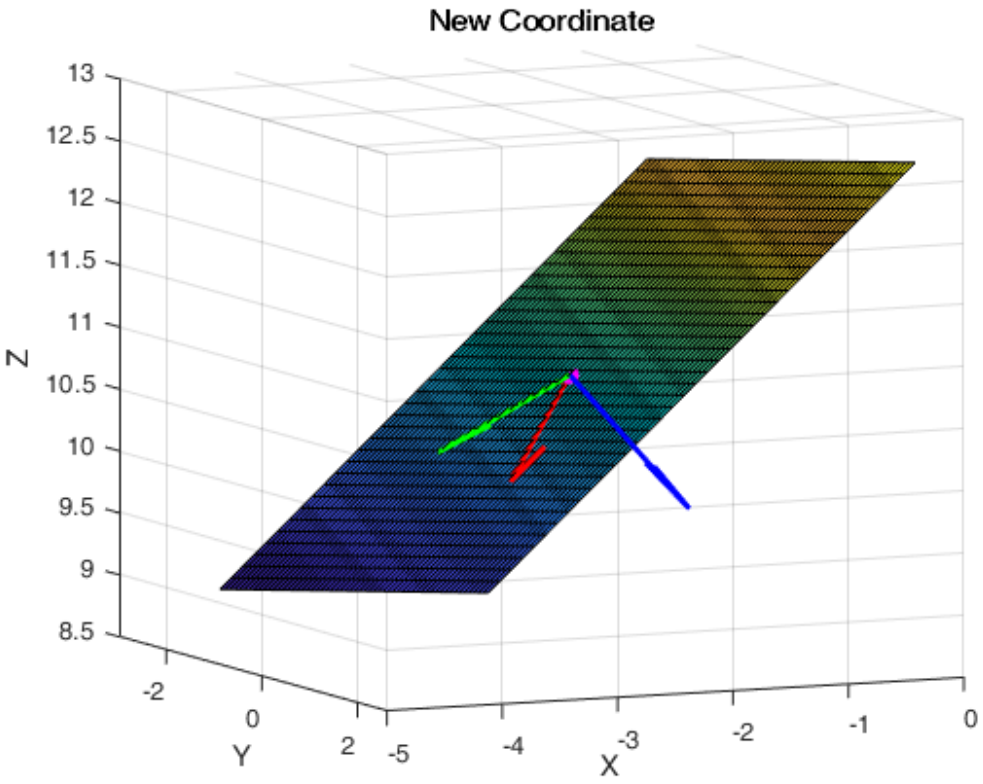
matches our constraint (inliers) in the point cloud, and colored them red. We set up 3-D grids to display the dominant plane, the range of its x and y is the minimum and maximum value of the inliers, and the corresponding z is obtained by the equation Ax + By + Cz + D = 0.



6) New Coordinate

The main objective of this project is to place a virtual box on the dominant plane, which is the wall in our case. A box is determined by 8 corners. Hence, if the positions of all corners are determined, the shape of box is formed. One way of achieving this is to directly add coordinates of 8 corners to the corresponding reasonable locations in the real-world coordinates system. However, computing the locations is not straightforward due to the inclination of the dominant plane. An alternative way and easier way to handle this is to transform the XYZ coordinates system to a new coordinate system in which the dominant plane lies in z=0 plane. We call the

new coordinate system Z0 system, the original system XYZ system. In the figure below, the XYZ coordinate system is the ordinary system labeld by each axis. The new coordinates system is formed by the three arrows.



To find the transformation T: Z0--->XYZ, we first need to find T1: XYZ-->Z0. Once T1 is obtained, T is simply the inverse process of T1. For any point s0 = (x,y,z)^T in XYZ system, T1 involves several steps as shown below:

1) s1 = s0 + k, where k = (d/a, d/b, d/c)^T according to plane equation P0: ax + by +cz + d =a(x+d/a) + b(y+d/b) + c(z+d/c) = 0. Hence, we have a new plane P1: ax1 + by1+ cz1 = 0, which has original point in it. In other words, P1 cross original point in XYZ coordinate system.

2) Find an orthonormal basis (v1, v2) of the P1 plane. To achieve this, we first randomly choose a unit vector v1 in P1 plane and then find another unit vector orthogonal to v1. Besides, v = (a, b, c)^T and v3 = v/||v||. As a result, we have V = (v1, v2, v3) as an orthonormal basis of XYZ coordinate which has two bases lying in the P1 plane.

3) s2 = (x2, y2, z2)^T, where x2 = <s1, v1>, y2 = <s1, v2>, y3 = <s1, v3>. Put it in a simpler

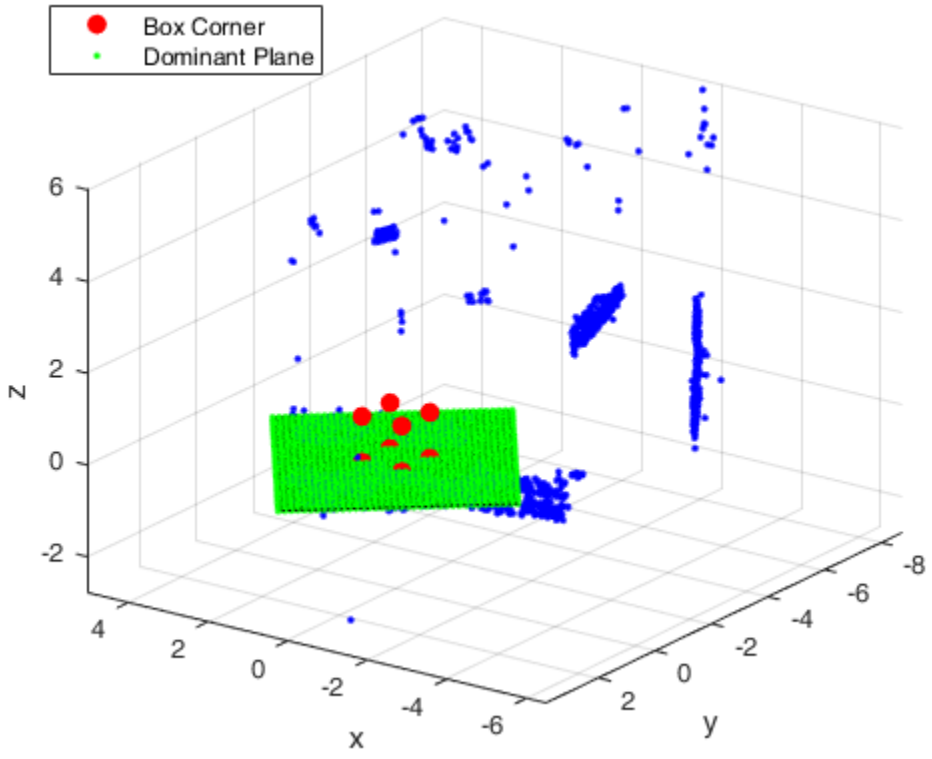
form, s2 = transpose(V)*s1. By inner product s1 obtained from step 1) with v1, v2, v3 respectively, we get s2 which is in Z=0 plane. Thus, s2 is the representation of s0 in Z0 coordinates system.

4) s3 = s2 – c, where c is the center of transformed dominant plane (the wall) in z=0 plane. The purpose of this step is to put the wall's center in the original point.

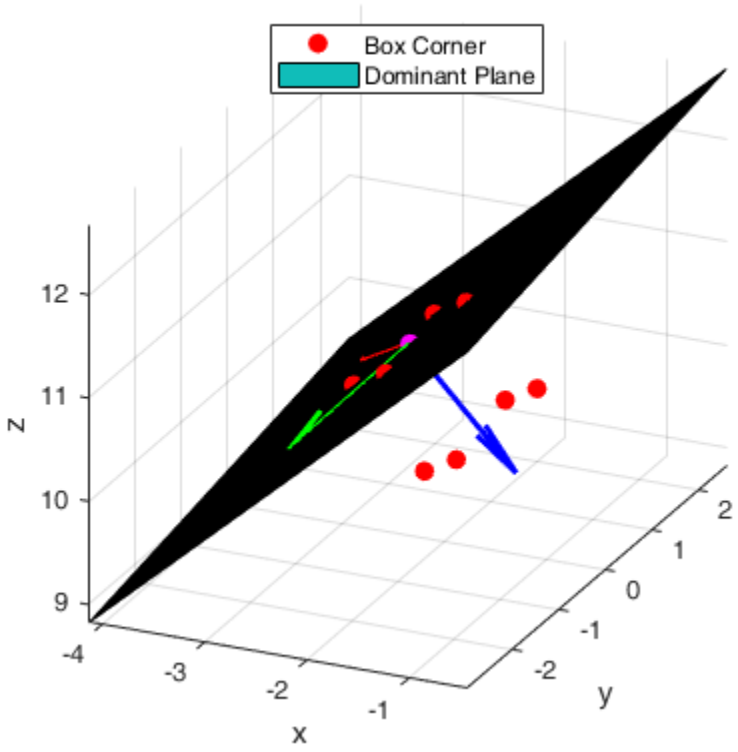
The four steps above transform a point from XYZ coordinate system to Z0 coordinate system, denoted by T1 transformation. With T1 determined, T is simply its inverse process. Given a point in Z0 coordinates system s3 = (x3, y3, z3)^T, s0 = V*(s3 + c) - k.

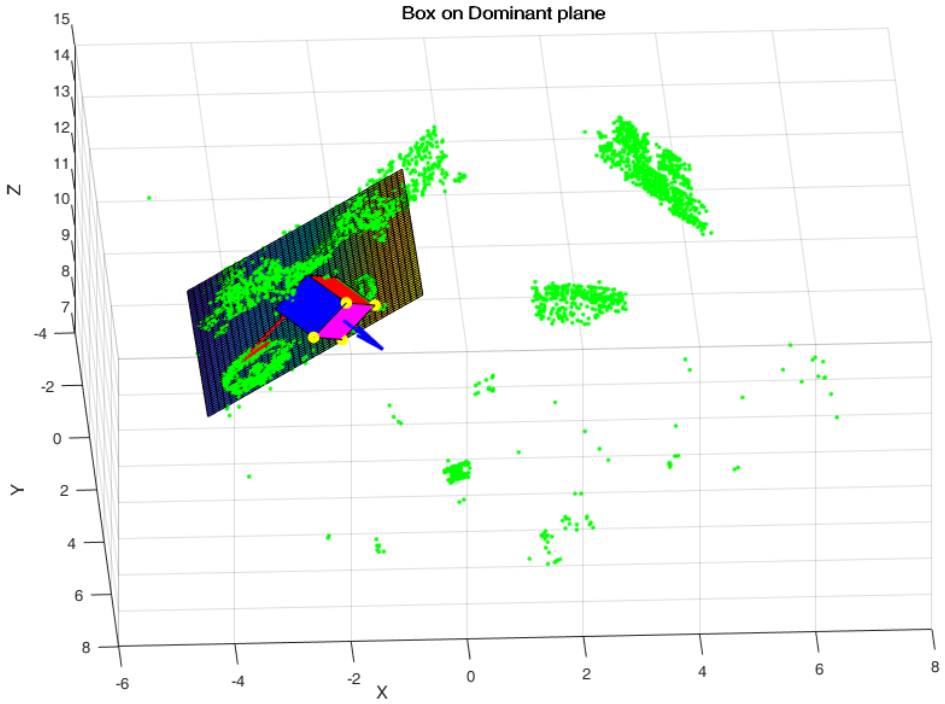
7) Create a 3D virtual box

The virtual box in Z0 coordinate was created as shown in figure below. We transformed coordinates of 8 corners of the box and obtained the results shown below. The box is located on the orignal point.



After transformation T, the box is placed on the dominant plane, which is shown below:





The complete virtual box after transformation is shown in the above figure.

8) Read External and internal camera parameters

To project 3D points into the image plane using a perspective transformation.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = K[R \mid t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

The internal parameters are in cameras.txt file. The parameters value for our model is shown in Figure 3. Form it we can see that we are using the SIMPLE_RADIAL camera model, which has four parameters: "f, cx, cy, k1"[1].

```
# Camera list with one line of data per camera:
# CAMERA_ID, MODEL, WIDTH, HEIGHT, PARAMS[]
# Number of cameras: 1
1 SIMPLE_RADIAL 4032 3024 3338.87 2016 1512 0.0217406
```

Figure 3 Camera model and parameters

The external parameters are in images.txt file. The parameters value for each image are shown in Figure 4. There are many data in this file, the parameters we concern are QW, QX, QY, QZ, TX, TY, TZ, CAMERA_ID, NAME. In the parameters, QW, QX, QY, QZ represent the rotation matrix R in (1). But it is the quaternion form, we need a function to convert it to 3x3rotation form. TX, TY, TZ correspond the t in (1). [1].

```
# Image list with two lines of data per image:
#   IMAGE_ID, QW, QX, QY, QZ, TX, TY, TZ, CAMERA_ID, NAME
#   POINTS2D[] as (X, Y, POINT3D_ID)
# Number of images: 6, mean observations per image: 2261.17
1 0.979263 0.202555 -0.00106734 0.00368678 -0.459095 1.82157 0.618126 1 image1.jpg
67.3665 23.1665 -1 75.9314 72.4692 -1 397.081 41.2597 -1 390.352 55.3784 -1 420.739 18.3504 -1 437.279
2 0.977508 0.0178535 -0.208185 -0.0286142 3.46842 -2.30138 0.941556 1 image2.jpg
301.685 52.7231 3209 340.084 19.4851 -1 371.309 28.0709 -1 358.142 63.7748 -1 416.307 19.109 -1 489.12
3 0.999995 -0.00104219 -0.00312473 -0.00032687 -0.606498 -2.21618 0.386019 1 image3.jpg
260.427 30.0599 -1 463.382 44.4322 -1 522.245 41.117 -1 505.309 50.9526 -1 526.215 47.0533 -1 552.109 4
4 0.979744 -0.0107324 0.198886 0.0207606 -5.38868 -1.97346 0.668188 1 image4.jpg
263.191 26.369 -1 271.59 29.4179 -1 310.457 46.1832 -1 310.457 46.1832 -1 376.384 63.4291 -1 444.3 12.15
5 0.967941 0.170931 -0.161207 -0.0887994 2.39997 1.02125 2.27773 1 image5.jpg
110.087 52.4594 -1 194.013 23.5531 -1 206.529 44.8199 -1 257.5 7.49413 -1 278.05 10.1669 -1 289.094 38.7
6 0.974184 0.0568479 0.210735 0.0576596 -6.10197 -0.970516 2.03217 1 image6.jpg
309.454 18.0612 -1 336.098 70.1414 -1 336.098 70.1414 -1 398.74 69.8619 -1 398.74 69.8619 -1 388.232 70
```

Figure 4 Camera model and parameters

represent the rotation matrix R in (1). But it is the quaternion form, we need a function to convert it to 3x3rotation form. TX, TY, TZ correspond the t in (1). [1].

Code:

In the main script, function [images,cameras] = ***fnc_readModel(path)*** is called to read the needed data from those two files. All the data are stored into to two struct, shown as Figure 5.

image_id	R	t	camera_id	name
1	[1.0000,-0.0077,-5.9686e-04;0.0068,0.9179,-0.3967;0.0036,0.3967,0.9179]	[-0.4591;1.8216;0.6181]	1	'image1.jpg'
2	[0.9117,0.0485,-0.4080;-0.0634,0.9977,-0.0230;0.4060,0.0468,0.9127]	[3.4684;-2.3014;0.9416]	1	'image2.jpg'
3	[1.0000,6.6025e-04,-0.0062;-6.4722e-04,1.0000,0.0021;0.0063,-0.0021,1.0000]	[-0.6065;-2.2162;0.3860]	1	'image3.jpg'
4	[0.9200,-0.0449,0.3893;0.0364,0.9989,0.0293;-0.3902,-0.0128,0.9207]	[-5.3887;-1.9735;0.6682]	1	'image4.jpg'
5	[0.9323,0.1168,-0.3424;-0.2270,0.9258,-0.3023;0.2817,0.3595,0.8896]	[2.4000;1.0213;2.2777]	1	'image5.jpg'
6	[0.9045,-0.0884,0.4171;0.1363,0.9869,-0.0865;-0.4040,0.1351,0.9047]	[-6.1020;-0.9705;2.0322]	1	'image6.jpg'

Field	Value
camera_id	1
model	'SIMPLE_RADIAL'
width	4032
height	3024
params	[3.3389e+03;2016;1512;0.0217]

Figure 5 intrinsic and external parameters

Basically, given the file path, the code can parse the file and pick the needed data. The only thing we would like to mention is that we call function r***otmat*** =

fnc_quat2rotmat(qvec) (shown as Figure 6) to convert the quaternion to 3x3 rotation form when read the images.txt file.

```
function rotmat = fnc_quat2rotmat(qvec)

rotmat = [1 - 2 * qvec(3).^2 - 2 * qvec(4).^2, ...
        2 * qvec(2) * qvec(3) - 2 * qvec(1) * qvec(4), ...
        2 * qvec(4) * qvec(2) + 2 * qvec(1) * qvec(3); ...

        2 * qvec(2) * qvec(3) + 2 * qvec(1) * qvec(4), ...
        1 - 2 * qvec(2).^2 - 2 * qvec(4).^2, ...
        2 * qvec(3) * qvec(4) - 2 * qvec(1) * qvec(2); ...

        2 * qvec(4) * qvec(2) - 2 * qvec(1) * qvec(3), ...
        2 * qvec(3) * qvec(4) + 2 * qvec(1) * qvec(2), ...
        1 - 2 * qvec(2).^2 - 2 * qvec(3).^2];

end
```

Figure 6 Matlab function *rotmat = fnc_quat2rotmat(qvec)*

9) Project a 3D(X, Y, Z) point into image

The principle behind this step is equation (1). In the main script, we call *fnc_ProjectPointToImage(Pworldpts_3D,Pose_External, Camera_Intrinsic)* to complete the transformation.

In the function, we take two steps to realize the projection.

a) Step1: transform the 3D point from world coordinate to camera coordinate

In this step, point coordinate in camera frame can be obtained by left multiplication the matrix [R | t] to $[X, Y, Z, 1]^T$ in equation (1). We can also get the depth information for each point in this step which is pretty useful in the problem 10. In addition, at the end of this step, the coordinates are normalized by the depth.

```
% Step1: World 3D coordinate to camera 3D coordinate
numpts = size(Pworldpts_3D,2);
Point3D_Homogeneous =  [Pworldpts_3D; ones(1,numpts)];
world_point =  Pose_External * Point3D_Homogeneous;   % 3x4 * 4xN = 3*N
Depth = world_point(3,:); % 1*N,  distance along the principle axis OF THE CAMERA (aka depth) to each point
world_point_hnormalized = world_point./world_point(3,:);   % 3*N → 2*N
```

b) Step2: transform the point from camera coordinate to image coordinate

In this step, point coordinate in image frame can be obtained by left multiplication the matrix [K] to the result [*x'*, *y'*] in step1 according to equation (1). Of course, we need to consider the distortion defined by SimpleRadialCameraModel [2].

```
% Step2: Camera 3D coordinate  to Image coordinate (WorldToImage)
% SimpleRadialCameraModel, refer to https://github.com/colmap/colmap/blob/master/src/base/camera_models.h
f = Camera_Intrinsic(1); % focal lengths expressed in pixel units
cx = Camera_Intrinsic(2);  %(cx, cy) is a principal point that is usually at the image center
cy = Camera_Intrinsic(3);
cxy = [cx;cy];
% distortion
r2 = vecnorm(world_point_hnormalized).^2;
k1 = Camera_Intrinsic(4);
world_point_distortion = world_point_hnormalized + k1*world_point_hnormalized.*r2;  % x'',y''

% Transform to image coordinates
Pimagepts = f*world_point_distortion(1:2,:) + repmat(cxy,1,numpts);
```

To calculate the distortion effectiveness:

$x'' = x'(1+k*r^2)$

$y'' = y'(1+k*r^2)$

$r^2 = x'^2 + y'^2$

Code:

In the main script, call function to get the image coordinates and depth information for each point. *[Pimagepts, Depth] = fnc_ProjectPointToImage(Pworldpts_3D,Pose_External, Camera_Intrinsic)*

```
%%Step3: load scene points coordinates
Pworldpts_3Dbox = C_box_XYZ';

%%Step4: WorldPointToImagePoint
num_images = length(images);
Pimagepts_3Dbox = zeros(2,length(Pworldpts_3Dbox),length(images));
Depth = zeros(length(images), length(Pworldpts_3Dbox));
for image_i=1:num_images
    Pose_External = [images(image_i).R images(image_i).t];
    %Camera_Intrinsic = [cameras(images(image_i).camera_id).params]';
    Camera_Intrinsic = [cameras(1).params]';

  [Pimagepts_3Dbox(:,:,image_i), Depth(image_i,:) ]= fnc_ProjectPointToImage(Pworldpts_3Dbox,Pose_External, Camera_Intrinsic);

end
```

Results:

We project the 8 corners created in problem 7 to the image, which are shown through Figure 7 to Figure 12. The red dots are the projection of corners in the dominant plane. Therefore, their projection should be in the same scene location in all images. And from the results, we can see the red dot are in the same location relative to the poster scene in the all. And the center of the corners square is in the middle of the wall posters. The results are exactly

as expected. So our projection function works well!

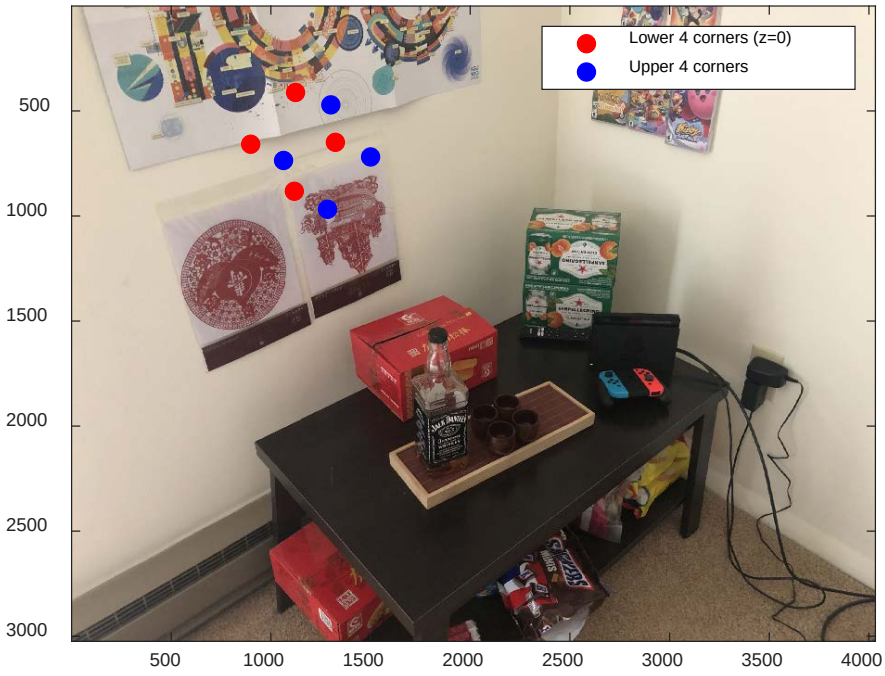


Figure 7 Corners projection in image1

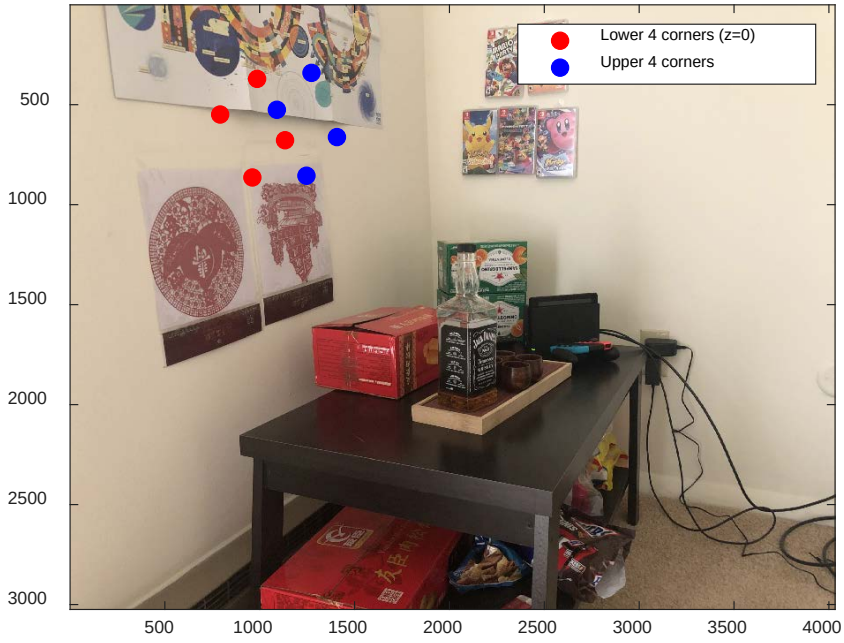


Figure 8 Corners projection in image2

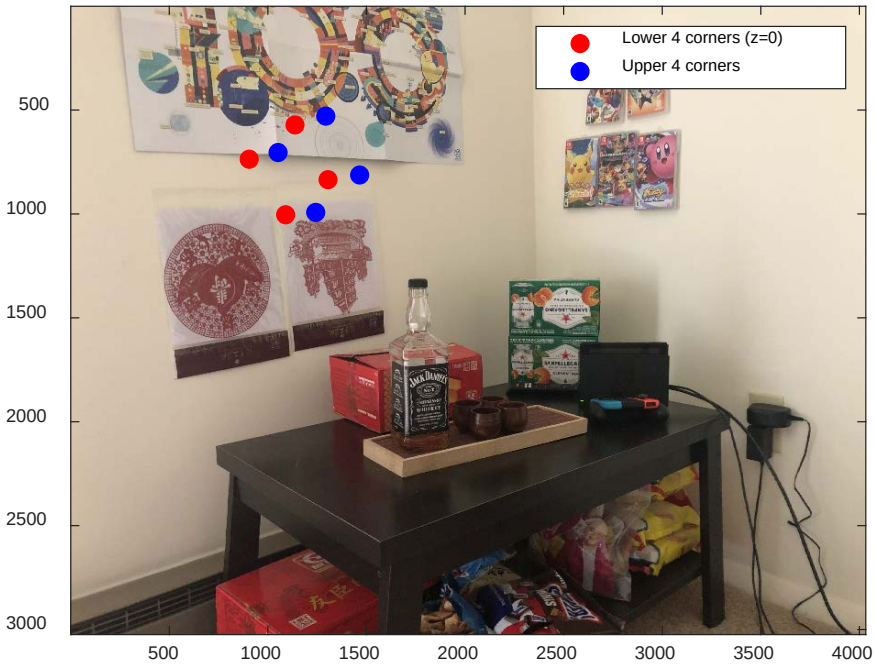


Figure 9 Corners projection in image3

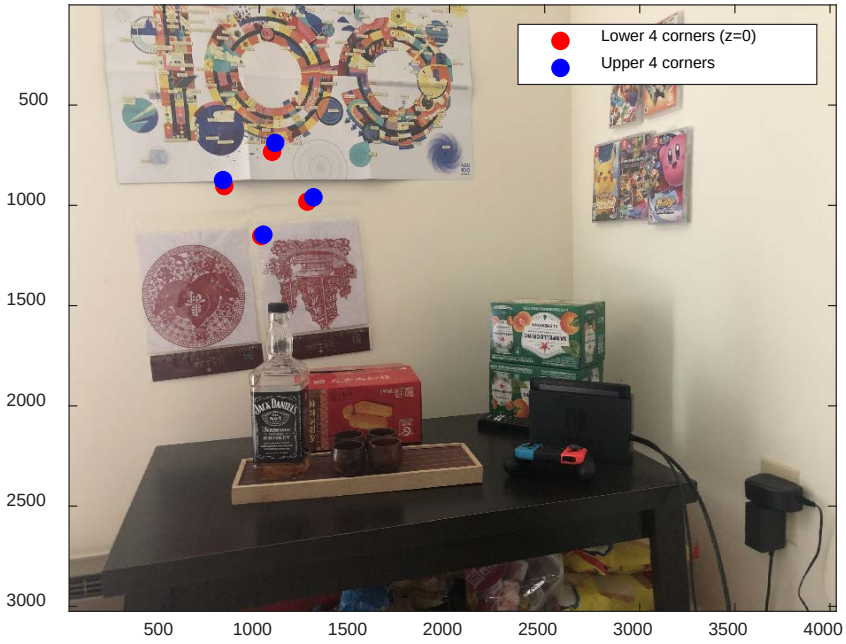


Figure 10 Corners projection in image4

Figure 11 Corners projection in image5

Figure 12 Corners projection in image6

10) Project the 3D box into the image and overlaid over the original pixel values.

For a cubic like box, at most 3 faces can be seen from a single point of view. And the visible faces have a common corner which is nearest to the camera along the principle axis of the camera. We use this principle to project the visible faces and corners to original image.

Code:

We first define the six face patches and assign distinct color to each of them.

```
%define patch for each face, four corners for each face
face(1,:) = [1 2 3 4];
face(2,:)  = [5 6 7 8];
face(3,:)  = [1 2 6 5];
face(4,:)  = [1 5 8 4];
face(5,:)  = [3 4 8 7];
face(6,:)  = [2 3 7 6];
face_color = ['y','m', 'c', 'b','r','g'];
```

Then find the visible faces and corners for each projection and drawn them.

```
I = (imread([images_path,'image',num2str(image_i),'.jpg']));
% find the nearest point
[~,nearest_corner_index ]= min(Depth(image_i,:));
% find the three nearest faces
[nearest_faces_index,~] = find(face ==nearest_corner_index);

h_fig = figure(1000+image_i);
set(h_fig,'Name',['Image',num2str(image_i),' and projection' ]);
colormap(gray);
clf;
% subplot(2,1,1);
imagesc(I);
hold on
for face_i = 1:length(nearest_faces_index)
   hold on
   patch(Pimagepts_3Dbox(1,face(nearest_faces_index(face_i),:),image_i),   Pimagept
   plot(Pimagepts_3Dbox(1,face(nearest_faces_index(face_i),:),image_i), Pimagepts_
   % plot(Pimagepts_3Dbox(1,5:8,image_i),Pimagepts_3Dbox(2,5:8,image_i),'b.','Mark
end
```

Results:

For a box in the dominant plane shown in 3D plot Figure 13, the box projection to each image are shown through Figure 14 to Figure 19. If we check the camera position in problem1, we can see that for the box in the dominant plane, only magenta, blue, and cyan can be viewed for all six projection. The results are also exactly as expected!

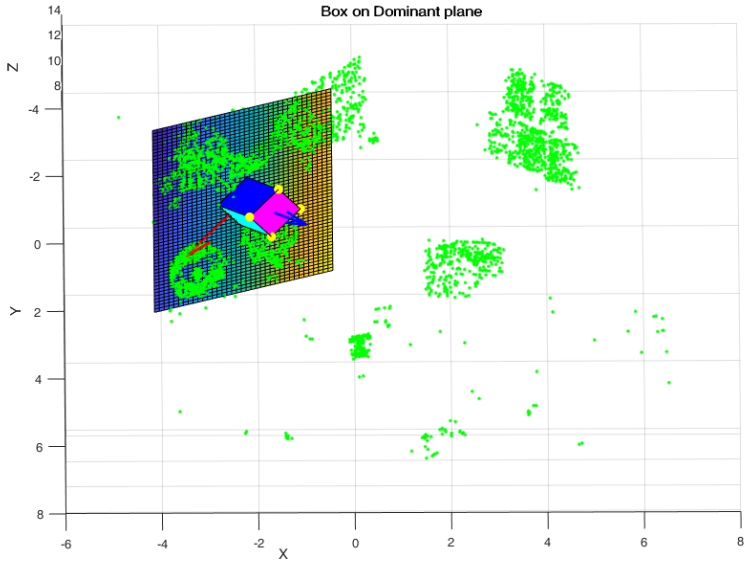


Figure 13 Box in dominant plane

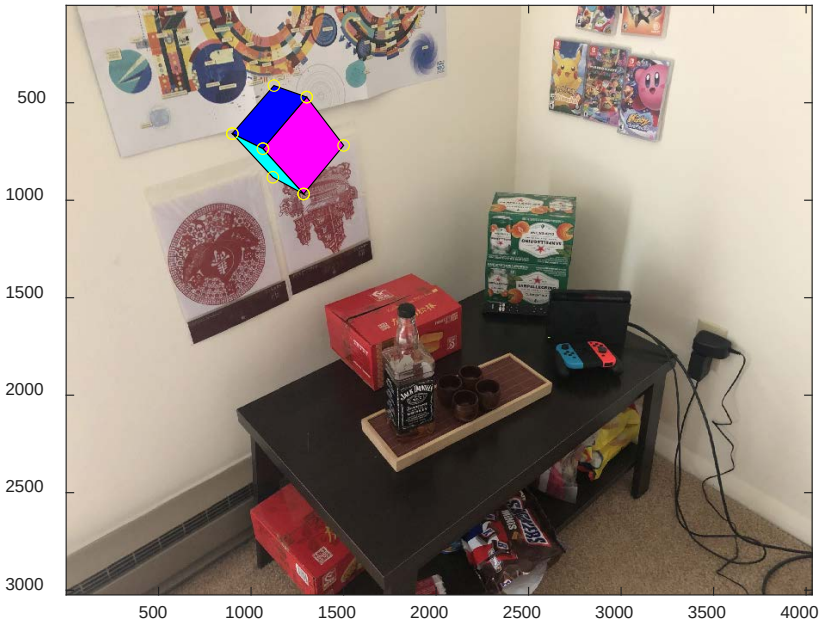


Figure 14 Corners projection in image1

Figure 15 Corners projection in image2

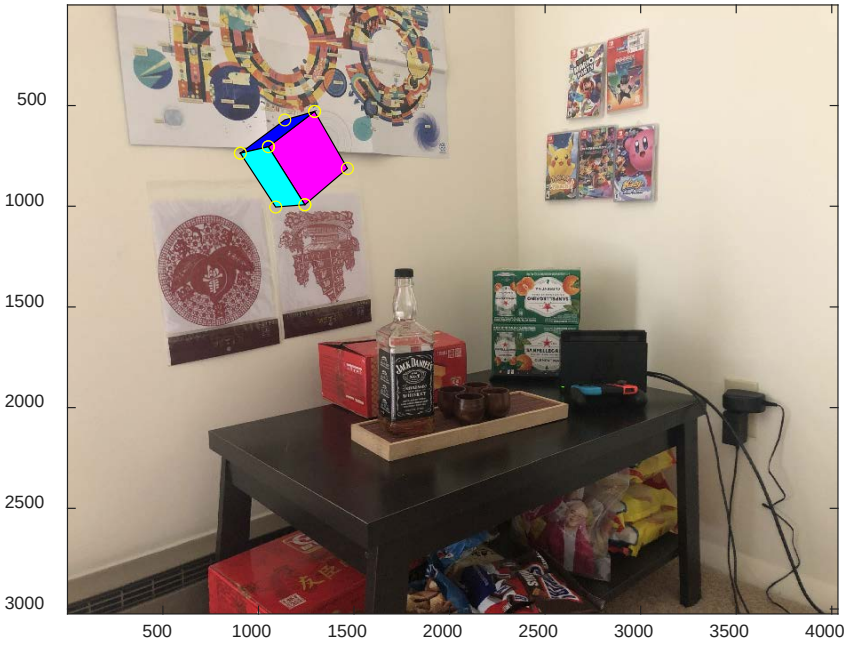


Figure 16 Corners projection in image3

Figure 17 Corners projection in image4

Figure 18 Corners projection in image5

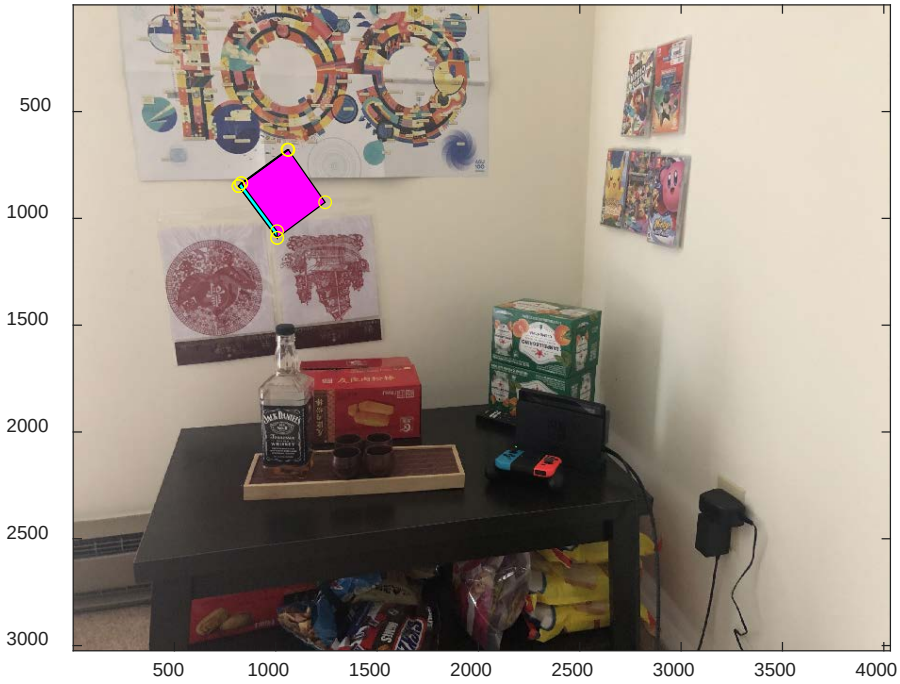
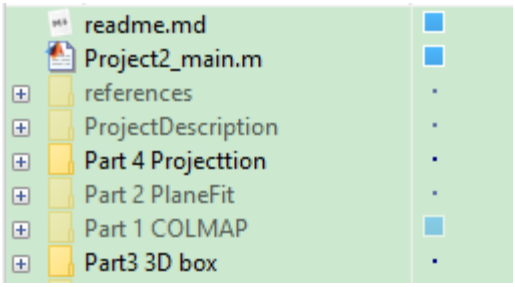


Figure 19 Corners projection in image6

C. Code organization

We put all the needed data and code into a folder. Just run Project2_main, and then we can get all the results for this project.



D. Work division

We divide the project into 4 tasks, and each member are responsible for each of them, include the code and report.

Task1&Task2: Chengyu Hu, Zekai Liu

Problem 1,2,3): take pictures, use COLMAP to generate point cloud, and camera\image parameters

Problem 4,5): fit a plane using RANSAC based on the point cloud and display the result.

Task3: Mingzhao Yu

Problem 6,7): 3D coordinate transformation, XYZ to xyz, create a virtual object(cubic) in xyz,find its XYZ

Task4: Liming Gao

Problem 8,9,10): read camera and images parameters. project a set of 3D(XYZ) poitns to image, project a 3D virtual box to image

Code organization

E. Conclusion:

By utilizing SfM, we place a virtual box on photos taken from a real-world scene. The boxes on photos of different angles are consistent with their real-world location and angle. From this project, we conclude that we can project properly any object that we know its function to the camera views through several consecutive transformations.

F. Reference:

[1] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[2] https://github.com/colmap/colmap/blob/master/src/base/camera_models.h