Zheng Qi, Yuchen Sun
Professor Robert Collins
CSE 586
April 9th

Project 1 Report

Introduction

In this project, we reconstruct a 3D scene from a bunch of 2D images taken in the same scene from several different positions. We also would like to create an object in our 3D scene, and then project it back to the 2D images. To approach our goal, we first use COLMAP to collect the features and reconstruct the 3D model. Then we use RANDSAC to find our desired plane in the 3D scene. Then we adjust plane using projection and rotation to make it parallel to the xy plane. After that we put a cubic on that plane, and finally project the virtual box object onto 2D image.

Approach

Step 1:

We first tried to use the photos of the living room of an apartment as the light is abundant there and the quality of the photos is pretty nice. However, after several experiments of step 1 to 5, we found that we could not detect the plane surface that we wanted successfully. The reason was that the furniture around contain much more features than the floor itself, even when we put a carpet and lots of other things on the floor to increase its features. After that we tried the photos from several different scenes, including our desks and corridors in the department building. Finally, we found the table for printer in lab W205 is a good choice. The wood texture of the table offered abundant features, while the purely grey printer blocked other features in the 3D scene. Thus, we took 15 pictures of this table from different positions using the same camera on cell phone with same focus. Three samples with different shooting angles are shown in Figure 1.

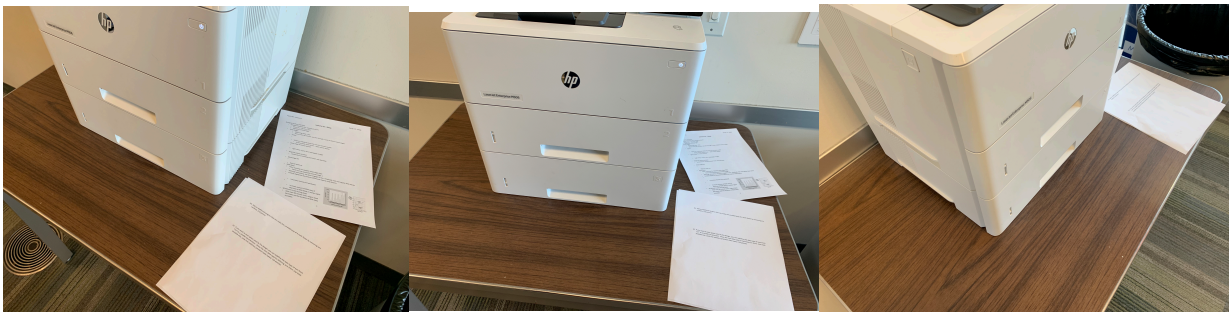

Figure 1 – Pictures of the table and printer in lab Westgate W205

Step 2:

The second step is using COLMAP to reconstruct the 3D model. We preceded feature extraction, feature matching, and reconstruction on images from step 1. Result is shown below in Figure 2. The red symbols are the positions and locations of the camera, and the grey dots are the

feature points. We export the model as text for further analysis.

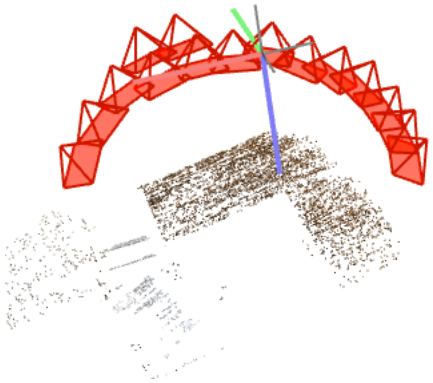

Figure 2 – 3D Reconstruction in COLMAP

Step 3:

To read in information of 3D feature points, we write a function named ”getDict” in our Python program. We read the x,y,z coordinates for each point and store them in a dictionary.

Step 4:

In step 4, we tried to find a plane that could describe largest subset of 3D points from the result of the previous step. To do this, we used RANSAC, an iterative algorithm estimates parameters of a model from a set of data that contains outliers. The number of iterations in defined an input. In each time of iteration, we randomly chose three points from our samples and calculated the formula of the plane defined by them. Then we computed the distance between all other points to this plane. We counted the number of points that have the distances less than a given threshold to the target plane. We would pick the plane that has the maximum number of points that have the distance less than the threshold as the final result. We tried 500,1000,2000,3000 as the number of iterations of sample. The result shows that 1000 iterations gave a good enough plane in a comparatively short computing time. We also tried different values for the error threshold. If it was too large, points that were not really in the plane would still be counted as a member of the plane. If it was too small, it would be hard to find a plane that many points agreed with. After testing, we finally use as 0.1 the threshold. From the function we learned that the equation of our plane is: -0.0155x + -2.0033y + -2.4230z + 19.0511 = 0.

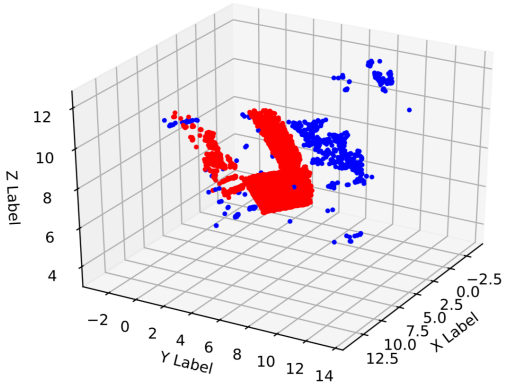

Figure 3 – 3D Plot of Scene in Python

Step 5:

In step 5, the function "plot3D" took the coordinates of all the points, the parameters of the desired plane and the threshold as input and outputted a 3D scatterplot of our model. Result is shown in Figure 3. For the points whose distance is smaller than the threshold, we use red points to represent them. For the points whose distance is higher than the threshold, we use blue points to represent them. From Figure 3 we can see that the plane of the table surface has been clearly detected. There are some red outliers at the left side that may cause by the papers on the table.

Step 6:

After we got the plane, the next step was to rotate this plane to become the plane that parallelled to xy plane and had z=0. According to theory, we could define a plane by its normal vector and a point on it. Thus, we computed the normal vector, projected it to the xy plane, and computed the rotation matrix to rotate it to the z-axis. Then we could use the rotation matrix to process all the points and rotating them to the plane where z = 0. Unfortunately, our projection and rotation in this step were not very successful, and did not get the plane we wanted. The judgment of the plane also failed, so the plots did not show in two different colors. Result is shown in Figure 4.

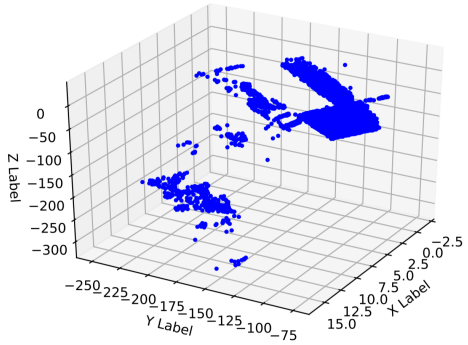

Figure 4 – 3D Plot of Scene parallel to xy plaine in Python

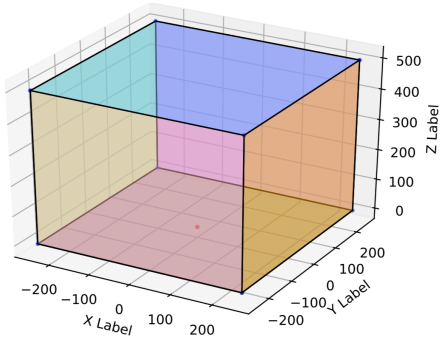

Figure 5 – Cube in 3D scene

Step 7:

In this step, we used the function "drawBox" to build a virtual cubic. The bottom of the cube is on the xy plane, and the center of the bottom surface is at coordinate (0, 0, 0). The length of edges is taken in as an input. For debug purpose, we used different color for each surface. A box with edge length 500 is shown in Figure 5. (Because the plots in Step 6 failed, we just show the cube here.)

Step 8:

For the read the carmera.txt and images.txt, we used the function in script "readModel.py" given by the COLMAP team. We called the function "read_model" in this script in our code for the next two steps.

Step 9:

To convert 3D coordinates to 2D coordinates, we used the function of Pinhole Camera Model learned in class, which is shown as below:

$$\lambda P_u = K[R|t]P_w$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} \pm 1/s_x & 0 & o_x \\ 0 & \pm 1/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

where Pu is the pixel location. K is the camera matrix. R is the rotation matrix. t is the translation vector . Pw is the world point. We implemented this function in function "to2D". The camera focus f was learned from "camera.txt", and the rotation matrix was learned from quaternion values in "images.txt". The center of 2D image (o_x, o_y) should be (width/2, height/2) = (4032/2, 3024/2) = (2016, 1512) according to "camera.txt". We set scale factors s_x and s_y to 1. c_x, c_y, and c_z are coordinates of a point on our target plane. Here we choose (1, 1, z).

We used another function "modifyImages" to compute the corresponding coordinates of vertexes in 2D images for all images. In the for loop in the function, method "to2D" was called to find the vertexes, and "draw2D" described in the next section was called to draw the cube in an image.

Step 10:

We drew back the cube to a 2D image by function "draw2D". The function plotted the eight 2D coordinates computed by "to2D" in the previous step, and connected them in certain orders to build six surfaces. We set the back and bottom surfaces opaque in order to prove that the cube covered the objects behind it, while we set the other surfaces semitransparent in order to show the structure of the cube more clearly. For the debug purpose, we used different color for each surface similar to that in step 7. Because of the incompletion of step 6, our images are not good enough, yet it still give some sense of 3D to 2D projection.

Distribution of Work

Yuchen: Step 2, 5, 7, 8, 9, 10
Zheng Qi: Step 1, 3, 4, 6