Augmented Reality Viewer

Mudit Garg Vijay Vamsi Nadella Arun Teja Muluka
Project 2 Report, CSE 586 Computer Vision II, Spring 2020
{mxg5783, vvn5075, avm6604}psu.edu

April 30, 2020

Introduction

The idea of the project is to implement a augmented reality viewer to display artificial objects on a real 3D scene. To get to this target, we have used COLMAP initially, which is used to generate the 3D points cloud by reconstructing the 3D scene. The output from this step would just be a 3D set of points which will be used for pre-processing in the subsequent steps. After this, RANSAC [2] routine is custom built to fit the plane and get the dominating plane. After, getting the dominant plane, the routine to get the 2D camera projection of the real-world points is implemented.

Methods

COLMAP

For this step, we have clicked images with a single camera with the same focal length (giving same camera intrinsic properties) with a dominant planar surface. After installing COLMAP, we used it to extract features by providing the set of images, later have extracted features, matched and incrementally reconstruct to get the sparse reconstruction, this sparse reconstruction has given out 3D cloud of points corresponding the the images inputted. This $3Dpoints.txt$ file generated will be having additional properties of the image. For this project, we have extracted the x, y, z values of each point only into a separate variable to be used in the subsequent steps. The other two files are, $camera.txt$ and $images.txt$. After these 3 files are generated using the images, COLMAP is not extensively used in further steps. Points 1, 2 and 3 are done till here. The snippet of $points3D.txt$ generated is shown in figure 1:

```
# 3D point list with one line of data per point:
#   POINT3D_ID, X, Y, Z, R, G, B, ERROR, TRACK[] as (IMAGE_ID, POINT2D_IDX)
# Number of points: 3154, mean track length: 3.6766
1008 -4.55016 0.738633 6.91494 73 135 238 0.318494 5 3724 3 2079 1 2748
65 -3.92333 -0.975256 8.09497 199 203 241 0.469614 4 445 5 520 1 416
944 -4.13009 0.197176 7.79645 115 165 255 0.836444 5 593 3 1890 1 754 6 324 2 384
1 -3.80797 1.35504 8.57513 13 73 110 1.93875 4 45 5 59 3 43 1 41 2 45
952 -3.21367 -0.544681 9.69734 74 147 255 0.888918 4 1572 3 1902 1 5463 2 2802
9 -4.3151 1.50172 7.65008 111 158 194 1.49579 4 187 5 333 3 159 1 366 6 169 2 184
2 -3.53054 1.27745 9.4121 0 52 78 1.61658 4 100 5 103 1 74 6 112 2 95
393 -3.56239 0.0012032 8.99747 137 141 170 0.495851 4 6308 5 7213 3 6182 1 5588
1054 -2.90042 0.717458 10.6204 34 129 195 0.637871 4 6538 5 1478 3 2153 1 1708 2
5180
1970 -3.53591 0.0778308 9.03675 58 112 187 2.33341 3 1871 1 5619 6 3483 4 1611 2
```

Figure 1: Snippet of $points3D$ file.

Only the X, Y, Z coordinates are taken from this file and stored into another variable using the code snippet shown in figure 2.

```
function [points]=readPoints()
fileID = fopen('points3D_1.txt','r');

output=textscan(fileID,'%f %f %f %f %*[^\n]','Delimiter',' ','CommentStyle','#');

points(:, 1) = output{2};
points(:, 2) = output{3};
points(:, 3) = output{4};
fclose(fileID);
%points_hor
return
```

Figure 2: $readPoints.m$

Steps 1, 2 and 3 of the problem description have been completed till here.

RANSAC

RANSAC(Random Sample Consensus) [2] is a non-deterministic algorithm to estimate parameters by randomly sampling from a set of observed data points. We use these parameters to determine the model that best describes the set of points. We adopt this routine to the current task of determining the dominant plane in the scene under consideration. This is an iterative algorithm which produces descent result in reasonable amount of time.

We iterate through the observed set of points in each iteration we sample a random subset of points. For our use case, we took a minimum of 3 random points as want to estimate a plane. A

plane is fit to these sampled points using least squares estimation [7] and collected all the points which have distance less than the threshold assumed. After each iteration, the number of points which form a dominant plane is updated is they are more than what was previously obtained. After all the iterations we probably end up with the dominant plane.

After this step is complete, we end up with the the inliers and the coordinates needed to create the plane equation. The algorithm for implementing RANSAC is shown in procedure 1.

Algorithm 1 RANSAC Routine

$bestSupport = 0$; $bestPlane(3,1) = [0,0,0]$
$bestStd = \infty$; $i = 0$
$\epsilon = 1 - forseeable\text{-}support/length(point\text{-}list)$
$N = round(\log(1-\alpha)/\log(1-(1-\epsilon)^3))$
while $i \leq N$ **do**
    $j =$ pick 3 points randomly among $(point\text{-}list)$
    $pl = pts2plane(j)$
    $dis = dist2plane(pl, point\text{-}list)$
    $s = find(abs(dist) \leq t)$
    $st = Standard\text{-}deviation(s)$

    if (length(s)$> bestSupport$) or (length(s)$= bestSupport$ and $st < bestStd$) **then**
        $bestSupport = length(s)$
        $bestPlane = pl$;
        $bestStd = st$
    end if
    $i = i + 1$
end while

The visualization of the dominating plane and the virtual box plotted on the dominating plane is reported in the Results section below.
Step 4 is completed with this.

World Coordinates to Camera Coordinates

Now, after obtaining the dominant plane using RANSAC, we converted the 3D-world coordinates into 3D camera coordinates. The dominant plane obtained was converted to $z = 0$ plane. The center of the plane was obtained and it was assumed that the local origin lies at that center. Thus, the camera location becomes the center of plane. In order to obtain the camera coordinates, equation of the form $P_C = R(P_W - C)$ was used. Here, P_C are the camera coordinates, P_W are the world coordinates, R is the rotation matrix and C is the camera location. In matrix

form, this equation is given by as shown in eq. 1.

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{1}$$

The method to obtain camera location is already stated previously. Now, in order to obtain the rotation matrix, it is obvious that the Y axis is very near to $[0, 0, 1]$. Z axis is normal to the relative distance between camera and world coordinates. X axis is ideally normal to the Y and Z axis. Using this logic, we obtained the rotation matrix, which is an orthonormal matrix and, thus, calculated the 3D local coordinates. The code snippet for this functionality is shown in figure 3.

```
function [camera_coor]=getCameraCoordinates(plane,worldPoints)

c=[0 0 0];
%c= mid point of pslane
for i=1:length(plane)
    c(1)=c(1)+plane(i,1);
    c(2)=c(2)+plane(i,2);
    c(3)=c(3)+plane(i,3);
end
c=c./length(plane);

[m,n]=size(worldPoints);
camera_coor=zeros(m,n);
for i=1:n
    P=worldPoints(:,i);
    Z=P-c';
    Zaxis=Z/norm(Z);
    X=cross([0,0,1],Zaxis);
    Xaxis=X/norm(X);
    Yaxis=cross(Zaxis,Xaxis);
    rot(1,:)=Xaxis;
    rot(2,:)=Yaxis;
    rot(3,:)=Zaxis;
    camera_coor(:,i)=rot*Z;
end


return
```

Figure 3: $getCameraCoordinates.m$

3D to 2D projection

Now, in order to convert 3D local coordinates into 2D image coordinates, we need intrinsic and extrinsic parameters of the camera. COLMAP gave us two files $camera.txt$ and $images.txt$.

the former contains the information regarding the intrinsic parameters of camera and later contains information regarding the extrinsic parameters i.e. rotation and translation. We iterated over these files and obtained the required matrices. The images file has the rotation matrix as unit quaternion. We converted to our required form using $quat2rotm$ function of matlab. The screenshots of sample $cameras.txt$ and $images.txt$ files are shown in figure 4 and 5 respectively. The order of arguments was obtained from [6].

```
# Camera list with one line of data per camera:
#   CAMERA_ID, MODEL, WIDTH, HEIGHT, PARAMS[]
# Number of cameras: 6
1 SIMPLE_RADIAL 1200 1600 1245.65 600 800 0.0577095
2 SIMPLE_RADIAL 1200 1600 1224.99 600 800 0.0274735
3 SIMPLE_RADIAL 1200 1600 1769.75 600 800 0.100056
4 SIMPLE_RADIAL 1200 1600 1280.31 600 800 0.0534728
5 SIMPLE_RADIAL 1200 1600 1220.87 600 800 0.0522953
6 SIMPLE_RADIAL 1200 1600 1226.84 600 800 0.0294545
```

Figure 4: $cameras.txt$ file

```
# 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: 1932.67
1 0.803317 -0.0451533 0.58139 0.12095 -5.33985 -0.999243 8.85397 3 WhatsApp Image 2020-04-24 at 5.12.56 PM (1).jpeg
488.727 1105.34 -1 812.748 777.27 -1 811.21 1120.47 -1 208.702 577.779 -1 424.001 903.404 -1 339.171 823.74 -1 642.165 1118.44 -
2 0.998277 0.00636613 0.0583254 0.000375681 2.70606 -1.47872 1.30778 5 WhatsApp Image 2020-04-24 at 5.12.57 PM (1).jpeg
528.56 594.576 -1 387.419 1038.32 -1 905.813 683.194 -1 793.702 1136.56 -1 908.973 1431.77 -1 848.061 250.501 -1 1046.86 339.633
3 0.921418 0.0229529 0.348463 0.170398 -1.86308 -1.04572 1.28571 1 WhatsApp Image 2020-04-24 at 5.12.55 PM (1).jpeg
922.695 559.533 -1 403.276 910.689 -1 612.411 1136.8 -1 874.002 1083.38 -1 482.586 1424.45 -1 210.961 1475.18 -1 358.908 676.033
4 0.99994 0.00253517 -0.00529759 0.00926307 2.722 -1.3408 0.325946 6 WhatsApp Image 2020-04-24 at 5.12.57 PM.jpeg
342.676 576.945 -1 642.764 1228.79 -1 126.56 481.965 -1 755.637 682.079 -1 403.619 1024.33 -1 1024.74 135.335 -1 912.412 338.33
5 0.795356 0.0504405 0.556887 0.23396s8 -6.59177 -1.77755 3.66385 4 WhatsApp Image 2020-04-24 at 5.12.56 PM.jpeg
```

Figure 5: $images.txt$ file

The code to extract the both the parameters is shown in figure 6. In this, pair of image and it corresponding camera was obtained using $ImageID$ and $CameraID$. For each camera, intrinsic and external parameters were stored in a separate data structure.

```
function [intrinsic_map_keys,intrinsic_map_values,extrinsic_map_values,extrinsic_map_keys,image_camera_map]=getCameraParameters()

    fileID = fopen('cameras.txt','r');

    output=textscan(fileID,'%d %s %f %f %f %f %f %f','Delimiter',' ','CommentStyle','#');
    fileID2 = fopen('images.txt','r');
    img=textscan(fileID2,'%d %f %f %f %f %f %f %f %d %s %*[^\n]','Delimiter',' ','CommentStyle','#');
    [m,n]=size(output{1});
    imgId=zeros(1,m);
    cameraId=zeros(1,m);
    intrinsic_map_values=zeros(3,3,m);
    intrinsic_map_keys=zeros(m,1);

    extrinsic_map_values=zeros(3,4,m);
    extrinsic_map_keys=zeros(m,1);


    %intrinsic_map=
    intrinsic=zeros(3,3);
    j=1;
    for i=1:m
        intrinsic(1,1)=output{5}(i);
        intrinsic(2,2)=output{5}(i);
        intrinsic(1,3)=output{6}(i);
        intrinsic(2,3)=output{7}(i);
        intrinsic(1,2)=output{8}(i);
        intrinsic(3,3)=1;
        intrinsic_map_values(:,:,i)=intrinsic;
        intrinsic_map_keys(i)=output{1}(i);

        quat = [img{2}(j) img{3}(j) img{4}(j) img{5}(j)];
        trans=[img{6}(j);img{7}(j);img{8}(j)];
        rotm = quat2rotm(quat);
        rot(:,1:3)=rotm(:,1:3);
        rot(:,4)=trans;
        extrinsic_map_values(:,:,i)=rot;
        extrinsic_map_keys(i)=img{9}(j);
        imgId(i)=img{1}(j);
        cameraId(i)=img{9}(j);
        j=j+2;

    end

    image_camera_map=containers.Map(cameraId,imgId);


return
```

Figure 6: $getCameraParameters.m$

After extracting the camera parameters, we used pinhole camera equation [4] to obtain the 2D coordinates.

$$\lambda P_u = K \, [\, R \mid t \,] \begin{bmatrix} P_W \\ 1 \end{bmatrix} \tag{2}$$

In equation 2, λ is an unknown scale factor. P_u is a 2D coordinates matrix of size 3×1. K is an intrinsic parameter matrix of size 3×3. R is a rotation matrix of size 3×3 and t is a translation matrix of size 3×1. Another row is added to P_w to make it of size 4×1. This equation in matrix form is shown in figure 7.

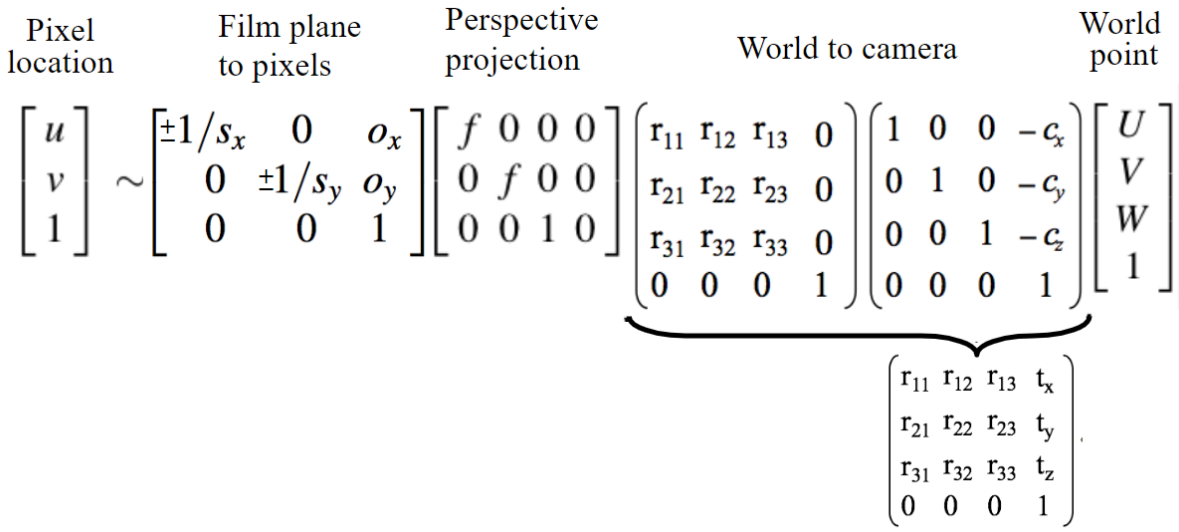

Figure 7: Equation system used to get the 2D coordinates.

As seen from equation given in 7, the resultant vector is of size 3×1. The first 2 rows gives the x and y coordinate in 2D plane.

Final Projection

After finding the 2D pixel co-ordinates by using the transformation matrices that we got from intrinsic and extrinsic parameters of the camera, we plot the a virtual box onto the dominant plane using the functions given in [3]. Then the modified co-ordinates of this cube are transformed from 3D world co-ordinates to 2D-pixel co-ordinates and are plotted on the corresponding image.

In order to verify the orientation, we have used different color coding for different cube surfaces that can be seen in the output figures in Results section. Then we used the z-ordering function to calculate the order to in which the virtual cube surfaces are plotted. This gives the solid cube effect which is achieved by adopting a z-buffer which stores the z-coordinate of each cube surface which is measured from the perspective of the camera. Let $Z = \{z_i\}$ represent the z-coordinates of the centers of all the surfaces. We sort Z in descending order and collect the corresponding surfaces after sorting. The farthest planes are plotted first and the nearest planes are plotted last. The code snippet for this logic is shown in figure 8.

```
function plot_and_project(img_id,translated_custom_box)
[ik,iv,ek,ev,ic_map]=getCameraParameters();

plane_ids = [1 2 3 4;
    5 6 7 8;
    1 2 6 5;
    4 1 5 8;
    3 4 8 7;
    2 3 7 6
    ];

c1_id = ic_map(img_id);

iv_id = find(ik==c1_id);
ev_id = find(ek==c1_id);

A = [iv(:,:,iv_id) zeros(3,1)];
R_t = [ev(:,:,ev_id); [0 0 0 1]];

trans_box_tmp = zeros(length(translated_custom_box),1) + 1;
translated_custom_box_new = [translated_custom_box trans_box_tmp];

proj_pixel_cord = zeros(length(translated_custom_box),3);

temp = (R_t*translated_custom_box_new')';
[zz, zi] = get_z_ordering(plane_ids, temp);

for i = 1:length(translated_custom_box)
    proj_pixel_cord(i,:) =  A*R_t*translated_custom_box_new(i,:)';
    proj_pixel_cord(i,1) =  proj_pixel_cord(i,1)/ proj_pixel_cord(i,3);
    proj_pixel_cord(i,2) =  proj_pixel_cord(i,2)/ proj_pixel_cord(i,3);
end
proj_pixel_cord = proj_pixel_cord(:,1:2);

tt = imread(strcat('./original_images/ori_img',int2str(c1_id),'.jpeg'));
```

Figure 8: Code Snippet for translating virtual box from dominant plane into pixel coordinates.

Results

We have taken the following set of images as input and passed them to COLMAP. Input images are shown in figure 10.

Figure 10: Input Images

Once we fed the images shown in figure 10 into COLMAP, we got the following image (shown in Fig. 11) and we exported the features/ parameters generated as txt files. The files $cameras.txt$, $images.txt$ and $points3D.text$, thus generated, can be found in the submitted zip file.

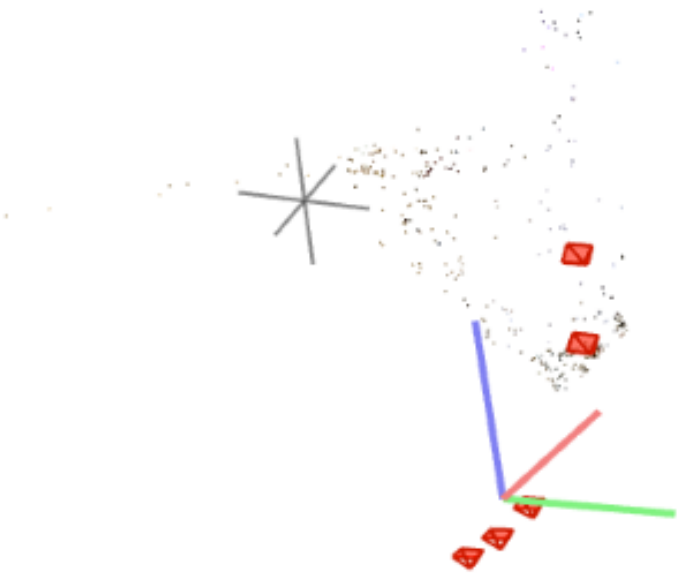

Figure 11: COLMAP after feeding the images

We extracted the world coordinates from $points3D.txt$ file using the method discussed in Methods section. These points were then fed to our RANSAC routine. The results obtained are shown in figure 12. These are images from different angles. We ran RANSAC for 1500 iterations. Also, in order to obtain the plane, minimum of 3 points were selected at random at each step. Threshold of 1 was selected assuming there are 40% outliers. On further analyzing, we found that we obtained wall as the dominant plane.

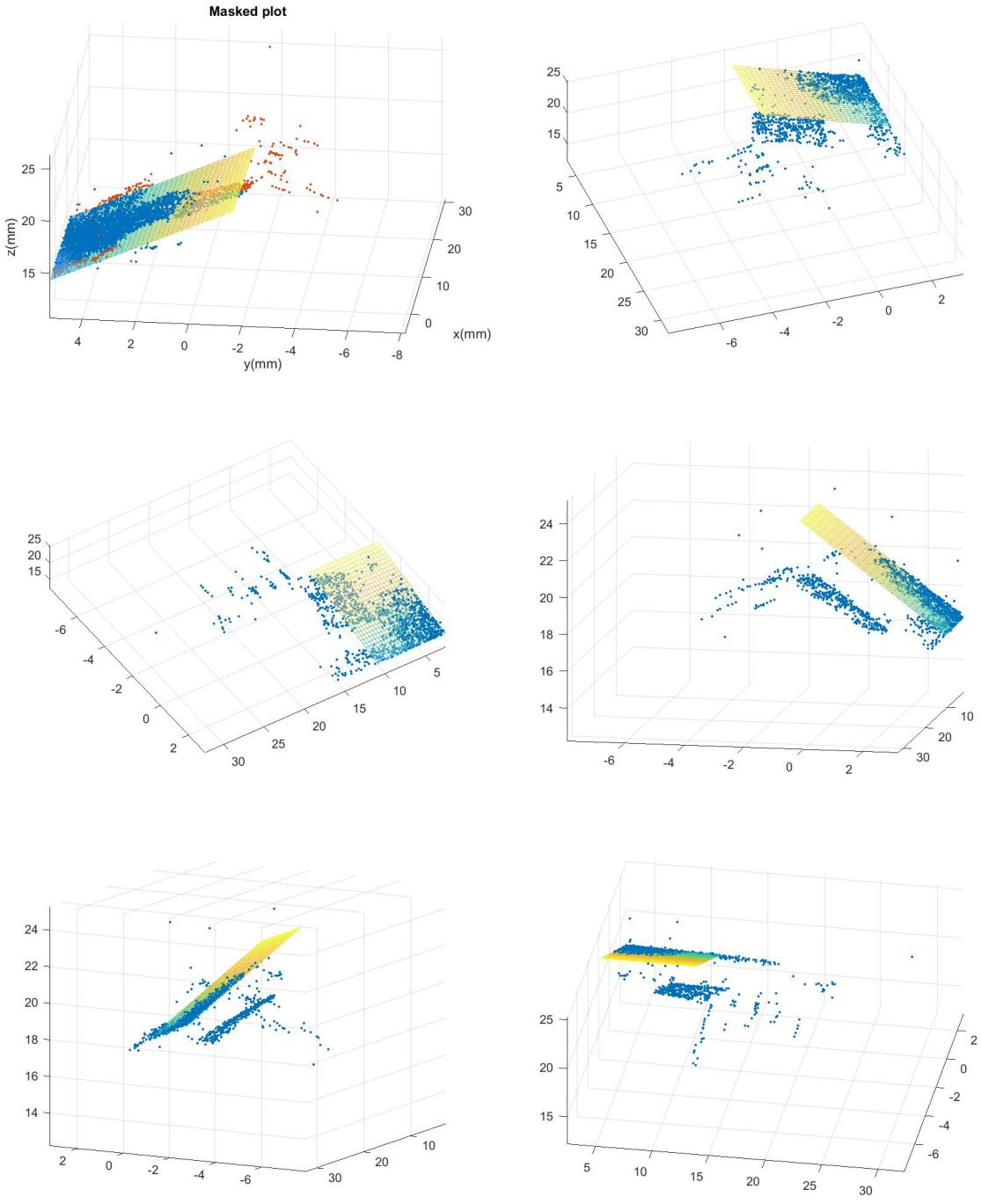

Figure 12: Plane obtained after executing RANSAC

The results shown in figure 12 seem correct. The red points are the inliers. We rotated the

plane and took screenshots at different angles. We see that the plane obtained lies where the majority of points present. Thus, it seems to be the dominant plane. We used this plane in further steps.

After obtaining the dominant plane, we converted 3D world coordinates into 3D camera coordinates. It was done in such a way that the local origin lies at the dominant plane. It is explained i the Methods section. Now, we put a 3D box on the dominant plane such that its bottom surface lies in the local $z = 0$ plane. We transformed the box similar to the method used for getting local camera coordinates. The projected box images are shown in figure 13.

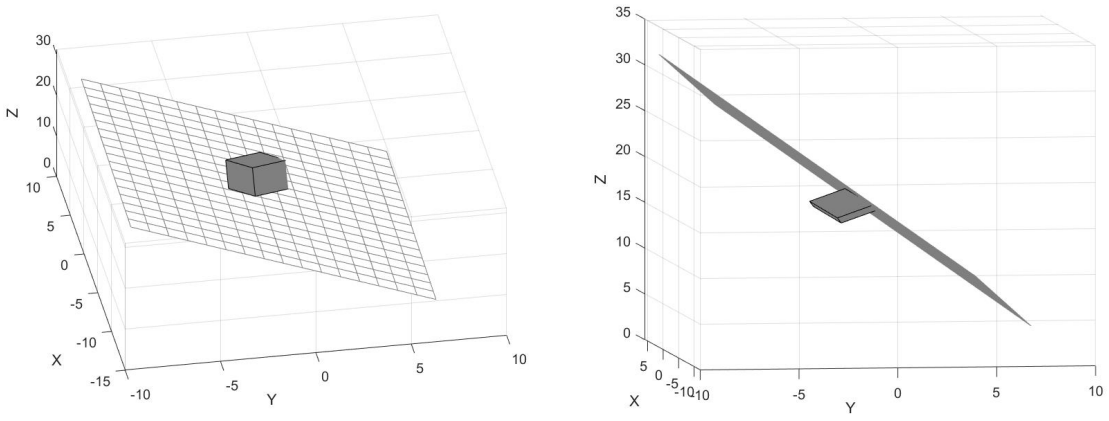

Figure 13: Visualization of the virtual box on a dominating plane.

The extrinsic and intrinsic parameters were further obtained (already discussed in). They were used to get the 2D coordinates of the image. For each image, we have got a separate set of parameters. We used each of them on their corresponding image and projected the final object using the method described in the final section.

The final results of the project depicting the virtual object on the scene across all the images are shown in the figures 14 and 15. As expected, the virtual object is present in the dominant plane obtained i.e. the wall. The object also occludes the pixels in the background. The object also orients according to the camera orientation and stays at the same point irrespective of the position of the camera.

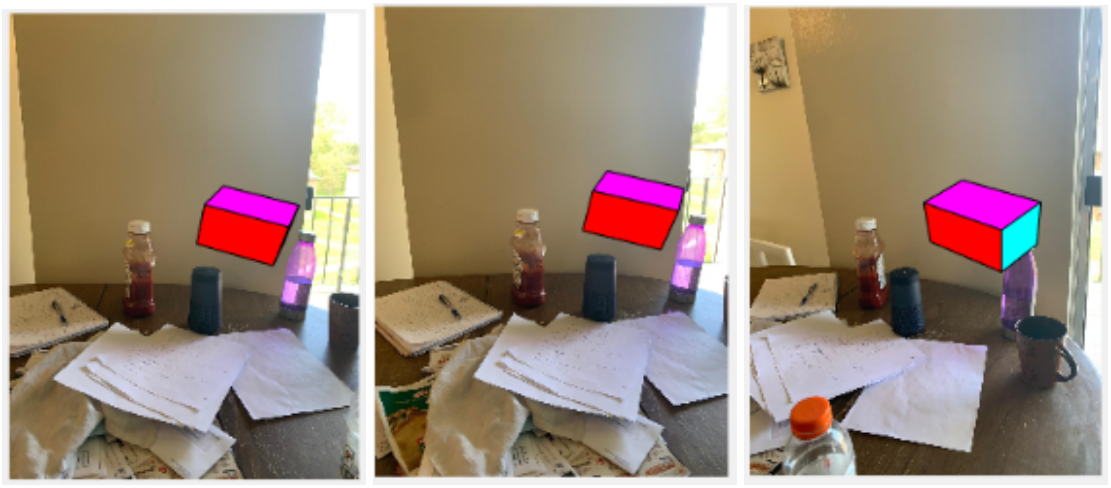

Figure 14: Results after placing the virtual box in the scene.

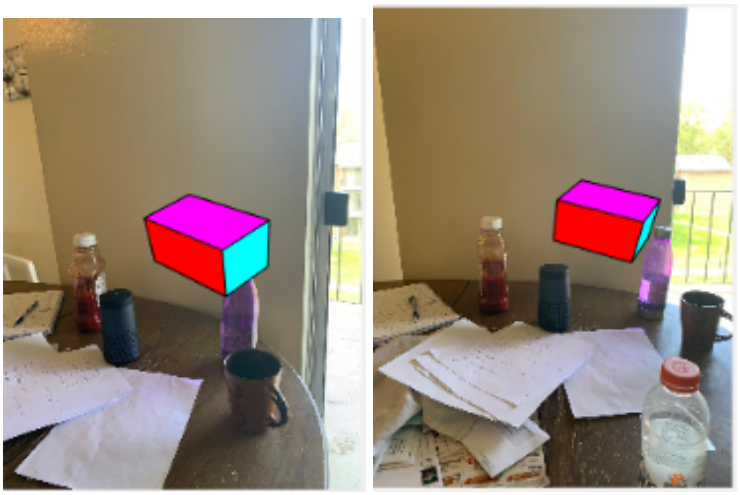

Figure 15: Results after placing the virtual box in the scene.

Code Details & Execution

The code of our entire work can be found in the same zip file submitted.
In order to run the code, one just need to execute the $main.m$ file in the code submitted. It will take the world coordinates generated by COLMAP and find a dominant plane using RANSAC. It will convert these world coordinates into 2D coordinates and project them on the obtained dominant plane. All the figures shown in the Results section will be generated.

Drawbacks and Challenges

During the implementation of this project, we faced some challenges. Also, there were some drawbacks that we observed from the approach used.

1. There is no upper bound on the time that RANSAC takes. It might be possible that the solution we have obtained is not optimal.
2. COLMAP is computationally expensive. It took lot of time to generate features and world coordinates for just a set 5 images.
3. The initial conversion of 3D world coordinates to 3D local coordinates was tricky. It consumed our lot of time.
4. Implementing RANSAC seems to be easy at first. But least square estimation was not an easy task.
5. We need to be very careful while transforming any of the matrix. A single wrong transformation will yield wrong results.

Contribution

All the members of the team contributed roughly equally to the project. The details of the task implementation are listed below:

1. **Mudit Garg**: Generation of 3D world points, coordinate transformations.
2. **Arun Teja Muluka**: RANSAC routine.
3. **Vijay Vamsi Nadella**: Projecting object onto image.

All the members contributed equally in report writing.

Conclusion

In conclusion, we were successfully able to project the virtual object on the 3D scene we created using the images we clicked by implementing the key concepts learned in the second module of the semester.This project was quite insightful in terms of coordinate geometry knowledge. Also, it was very fascinating when the theoretical knowledge matches with the practical implementation. In the results section, it can be seen that a 3D object was successfully projected onto the image. It is also observed that the orientation of the box also changes with the change of direction of the camera from image to image.

References

[1] J. L. Schönberger and J. Frahm, "Structure-from-Motion Revisited," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV,* 2016, pp. 4104-4113.

[2] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". *ACM*, 1981

[3] "3D Polygon - draw planes with Matlab," *matrixlab-examples*, Retrieved April 28, 2020, http://www.matrixlab-examples.com/3d-polygon.html

[4] "Pinhole Camera Model," (February 24, 2020). *Wikipedia*, https://en.wikipedia.org/wiki/Pinhole_camera_model

[5] "Transform a plane to the xy plane," *Stack-Exchange*, Retrieved April 29, 2020, https://math.stackexchange.com/questions/1167717/transform-a-plane-to-the-xy-plane

[6] J. L. Schönberger, "Output Format - COLMAP," https://colmap.github.io/format.html

[7] "Least Square Fitting," *Wolfram Research Inc.*, Retrieved April 29, 2020, https://mathworld.wolfram.com/LeastSquaresFitting.html

[8] Kaucic, R., R. Hartley, and N. Dano. "Plane-based projective reconstruction", *Computer Vision, IEEE International Conference on* 1, 420, 2016