Advanced Vision Practical (2018) Report

Yu, Jianmeng   Yi, Ruitao

March 29, 2018

1 Introduction

The task is to construct a complete box form 50 3D point clouds acquired by a Kinect sensor. One can get a complete box if all the 3D points captured in each frame could be processed and transformed into points in the initial frame.

2 Algorithms and implementation

2.1 Extract the relevant data from each point cloud

2.1.1 Description

The key idea for data extraction is how to construct a mask to eliminate the irrelevant data. First, we need to erase the data points that are distant from the box we concerned. Then, the following two type of irrelevant data are removed:

- **Background Blue Data Points** - using `removeBlue.m`, to remove these data points coming from the lab window.
- **Skin-coloured Data Points** - using `removeBlue.m`, to remove the detection on the hand holding the box.

During the faraway data eliminating process, we only retain data which is within a interval of *[Center* $\pm$ *Offset]*, where the center is provided by the material and the offset is estimated by our group. This process result in a binary mask of size 424x512, marking the area of valid box points. Also, the technique of Erode and Dilate is used to remove noise points found at edge of the box. Occasionally, this process will fail to filter some hand data points. However, these points are filtered out at the plane extraction stage of the project.

Using this binary mask, the project could use the extracted points to construct a new Point Cloud, filtering out the irrelevant data points. Result of this process on point cloud is shown in Fig 1 and Fig 2. And the effect on original image is shown in Fig 3 and Fig 4.

2.1.2 Algorithm

The Pseudo-code of the algorithm is shown in Algorithm 1. The original Matlab code is in Appendix C.

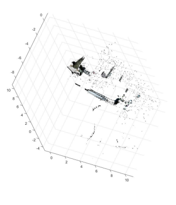
Figure 1: Raw data from frame 19

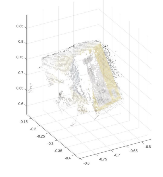
Figure 2: Extract the relevant data from frame 19

Algorithm 1 Relevant data extraction

```
Input: $Frame\ number$
Output: $Extracted\ point\ cloud$
function GETPOINTCLOUD($frameNum$)
    $Colour \leftarrow$ GETCOLOUR($frameNum$)
    $Location \leftarrow$ GETLOCATION($frameNum$)
    $Mask \leftarrow$ GETMASK($frameNum$)
    $Count \leftarrow 1$
    for $i = 1 : maxRow$ do
        for $j = 1 : maxColumn$ do
            if $valid\ point(i, j, :)$ && $mask(i, j, :) \neq 0$ then
                $RGB\{Count\} = Colour(i, j, :)$
                $XYZ\{Count\} = Location(i, j, :)$
            end if
        end for
    end for
    $results \leftarrow$ $Point\ cloud\ reconstructed\ using\ RGB\ and\ XYZ$
    return $result$
end function
function GETCOLOUR($frameNum$)
    $Colour \leftarrow pc\{frameNum\}.Colour$
    $results \leftarrow flip(rot90(reshape(Colour, [\max Row,\ \max Column, 3])))$
    return $result$
end function
function GETLOCATION($frameNum$)
    $Location \leftarrow pc\{frameNum\}.Location$
    $results \leftarrow flip(rot90(reshape(Location, [\max Row,\ \max Column, 3])))$
    return $result$
end function
function GETMASK($frameNum$)
    $Center \leftarrow [-0.71, -0.30, 0.81]$
    $Offset \leftarrow [-0.15, 0.20, 0.25]$
    $\text{Mask1} \leftarrow \begin{cases} 1 \text{ if Location} \in [\text{Center} - \text{Offset}\ ,\ \text{Center} + \text{Offset}] \\ 0 \text{ otherwise} \end{cases}$
    $Mask2 \leftarrow removeSkin\{$GETCOLOUR($frameNum$)$\}$
    $Mask3 \leftarrow removeBackground\{$GETCOLOUR($frameNum$)$\}$
    $results \leftarrow denoise(Mask1)$ && $denoise(Mask2)$ && $denoise(Mask3)$
    return $result$
end function
```

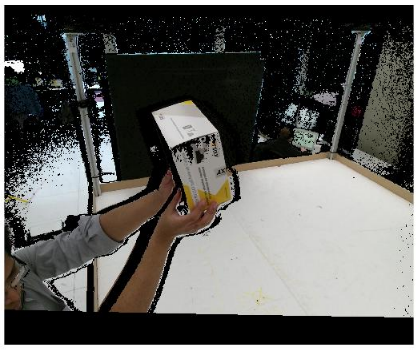
Figure 3: Raw Image from Frame 36

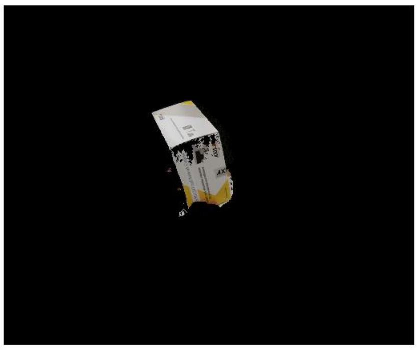
Figure 4: Masked Image from Frame 36

2.2 Extract the planes from each point cloud

2.2.1 Description

The main idea of this step is that we extract raw planes with the supplied ”Patch-growth” algorithm. And by doing it multiple times, we filter out the misfit planes and obtain a clean set of plane.

1. Raw plane extraction
   First, we define the maximum number of planes we want per frame. Then, for each iteration, we select a patch from the remaining points, and generate a plane with the patch. After that, for each point left, we add it to the patch if the distance to this patch plane is smaller than the tolerance. Once the loop is complete enough points, we remove the points from the list, and store the plane normal.

2. Get cleaned plane
   The key idea in this part is that how to de-noise the plane got from the first stage and how to re-mark the intersection points lying on the intersecting planes. Firstly, for each round, we get the raw planes and sort them by the number of points on it in descending order. The plane with the maximum number of points is selected. For each remaining possible planes, we average the planes within 15 degree angle of the selected plane. And we drop all other plane within 60 degrees angle (We assumed the box surface are orthogonal to each other). Finally, for all cleaned planes, we first remove intersection points from their originally marked planes and then we assign them to the plane with minimum distance.

The Pseudo-code of the algorithm is shown in Algorithm 2. The original Matlab code is in Appendix D.

And the result of this process on point cloud is shown in Fig 5 and Fig 6.

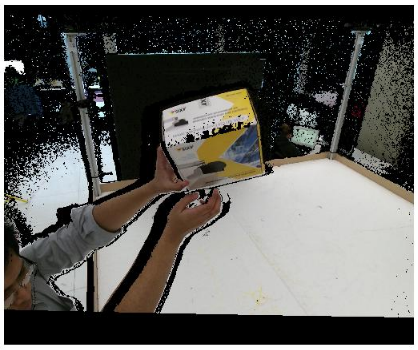
Figure 5: Raw data from frame 6

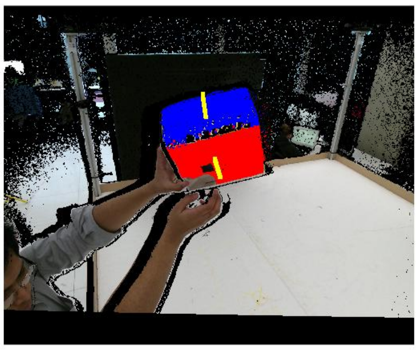
Figure 6: Extract the planes from frame 6

Algorithm 2 Clean planes extraction

```
Input: $Frame\ number,\ Plot(boolean, optional)$
Output: $Planesout : (a, b, c, d)$ which corresponds to $ax + by + cz + d = 0$
Output: $Pointsout :\ points\ on\ each\ plane$
 1: function GETCLEANPLANE($frameNum, plot$)
 2:     $Location \leftarrow$ GETPC($frameNum$).$Location$
 3:     $Planes \leftarrow \emptyset$
 4:     $Points \leftarrow \emptyset$
 5:     for $each\ round$ do
 6:         $[Tempplane, Temppoint] \leftarrow$ GETPLANES($Location$)
 7:         $Planes \leftarrow Catenate(Planes, Tempplane)$
 8:         $Points \leftarrow Catenate(Points, Histogram\ of\ Temppoint)$
 9:     end for
10:     for $each\ iteration$ do
11:         $Sort\ the\ Planes\ by\ the\ number\ of\ Points\ assigned\ to\ it\ in\ descending\ order$
12:         $MaxPlane \leftarrow Planes\{1\}$
13:         for $i = 1 : num\ of\ remaining\ Planes$ do
14:             $Normal1 \leftarrow MaxPlane(1 : 3)$
15:             $Normal2 \leftarrow Plane\{i\}(1 : 3)$
16:             $Angle \leftarrow arccos(Normal1, Normal2)$
17:             if $angle$ <=MAXANG then $Mark\ Plane\{i\}\ as\ candidate\ plane$
18:             else if MAXANG < $angle$ <=REJANG then $Remove\ Plane\{i\}$
19:             end if
20:         end for
21:         $NewPlane \leftarrow Average\ all\ candidate\ planes$
22:         $PossiblePlaneNum \leftarrow PossiblePlaneNum + 1$
23:     end for
24:     for $all\ Possible\ Planes$ do
25:         $Store\ points\ on\ intersection\ lines$
26:         $Remove\ intersection\ points\ from\ their\ originally\ marked\ plane$
27:     end for
28:     for $each\ intersection\ point$ do
29:         $Re - assign\ it\ to\ the\ plane\ with\ minimum\ orthogonal\ distance$
30:     end for
31:     $results \leftarrow updated\ possible\ planes$
32:     return $result$
33: end function
34:
35: function GETPLANES($Location$)
36:     $Remaining \leftarrow Location$
37:     for $i = 1 : MaxIteration$ do
38:         $[Plane\{i\}, remaining] \leftarrow Select\_Patch\{remaining\}$
39:         for $each\ remaining\ point$ do
40:             $measure\ distance\ from\ this\ point\ to\ Plane\{i\}$
41:             if $distance < threshold$ then
42:                 $Add\ this\ point\ to\ Plane\{i\}$
43:             end if
44:         end for
45:     end for
46:     $results \leftarrow [Plane, points]$
47:     return $result$
48: end function
```

2.3 Estimate the 3D positions of the corners where planes meet

2.3.1 Description

There could be a maximum of 3 cleaned planes for each frame. If a frame have less than 2 planes, there should be no corners, so we ignore frames. The algorithm has two core steps:

- Obtain reasonable points on the intersection lines

  To get reasonable data points, first, we need to calculate the unit vector pointing to the direction of the intersection line of two intersecting planes. Then, we project all the points on the vector and select all data points between quantiles 0.04 and 0.96 before winsorizing all reasonable data points.

- Forming a corner

  If there are two planes in this frame, then for each plane, we project all points of this plane in the direction of the normal of another plane. After that, we select the furthest projected point as a new corner point which should lie on the edge of that plane. If there are three planes, assume that we have done the first step with plane1 and plane2. For each pair (Plane1, Plane3) and (Plane2, Plane3), first, we project all points of this plane on the direction of the normal of the remaining plane and select the furthest projected point as a new corner point which should lie on the edge of that plane afterwards.

The Pseudo-code of the algorithm is shown in Algorithm 3. The original Matlab code is in Appendix E. And the result of this process on point cloud is shown in Fig 7 and Fig 8.

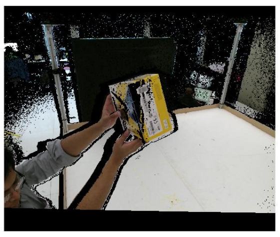
Figure 7: Raw data from frame 19

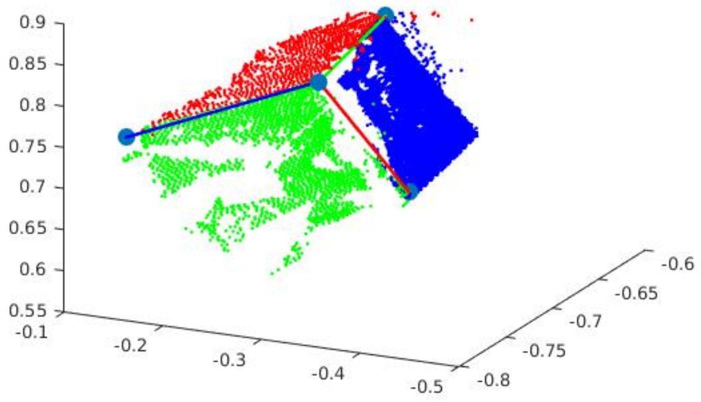

Figure 8: Extracted corners from frame 19

Algorithm 3 Corners 3D estimation

```
Input: $Planes, Points(get from GETCLEANPLANE)$
Output: $Corners$
 1: function GETCORNER($arg1, planes, points$)
 2:     $NPS \leftarrow Num\ of\ Clean\ Planes\ extracted$
 3:     if $NPS == 1$ then
 4:         $Corners \leftarrow \emptyset$
 5:     else if $NPS == 2$ then
 6:         $[Points\_12, Vector\_12] \leftarrow$GETPLANEINTERSECTION(Planes{1}, Planes{2})
 7:         $Tps \leftarrow project\ all\ points\ on\ Vector\_12$
 8:         $Reasonable\ points \leftarrow$ PRCTILE(Tps,[lowerbound,upperbound])
 9:         $Winsorize\ Tps$
10:         $Corners\{1\} \leftarrow Nearest\ reasonable\ point$
11:         $Corners\{2\} \leftarrow Furthest\ reasonable\ point$
12:         $Corners\{3\} \leftarrow$ADDANOTHERPOINT($Normal1, Plane2, Corners\{1\}$)
13:         $Corners\{4\} \leftarrow$ADDANOTHERPOINT($Normal2, Plane1, Corners\{1\}$)
14:     else if $NPS == 3$ then
15:         $[Points\_12, Vector\_12] \leftarrow$GETPLANEINTERSECTION(Planes{1}, Planes{2})
16:         $Tps \leftarrow project\ all\ points\ on\ Vector\_12$
17:         $Reasonable\ points \leftarrow$ PRCTILE(Tps,[lowerbound,upperbound])
18:         $Intersection\ point \leftarrow$GETPLANELINEINTERSECTION($Planes\{3\}, Points\_12, Vector\_12$)
19:         $Shift\ Reasonable\ points\ in\ the\ direction\ of\ the\ Intersection\ Points$
20:         $Winsorize\ Tps$
21:         $Corners\{1\} \leftarrow Nearest\ reasonable\ point$
22:         $Corners\{2\} \leftarrow Furthest\ reasonable\ point$
23:         $Corners\{3\} \leftarrow$GETANOTHERINTERSECTION($Normal1, Planes\{2\}, Planes\{3\}, NP$)
24:         $Corners\{4\} \leftarrow$GETANOTHERINTERSECTION($Normal2, Planes\{1\}, Planes\{3\}, NP$)
25:     end if
26:     $results \leftarrow Corners$
27:     return $result$
28: end function
29:
30: function ADDANOTHERPOINT($Normal, Plane, Startpoint$)
31:     $Points \leftarrow Project\ all\ points\ on\ Plane\ to\ Normal$
32:     if $Mean(Points) < 0$ then
33:         $Corner \leftarrow Points\ assigned\ with\ Negative\ Largest\ Value$
34:     else
35:         $Corner \leftarrow Points\ assigned\ with\ Largest\ Value$
36:     end if
37:     $results \leftarrow Corner$
38:     return $result$
39: end function
40:
41: function GETANOTHERINTERSECTION($Normal, Plane1, Plane2, Startpoints$)
42:     $Points \leftarrow Project\ all\ points\ on\ Plane1\ and\ Plane2\ to\ Normal$
43:     if $Mean(Points) < 0$ then
44:         $Corner \leftarrow Points\ assigned\ with\ Negative\ Largest\ Value$
45:     else
46:         $Corner \leftarrow Points\ assigned\ with\ Largest\ Value$
47:     end if
48:     $results \leftarrow Corner$
49:     return $result$
50: end function
```

2.4 Corners and point clouds normalization

2.4.1 Description

In order to fuse the images to the same system, we need to translate the planes onto XY, XZ, YZ planes for easier fusion. This is possible due to the 3D object we are fusing is a box.

To do the normalization, we need to estimate the rotation and translation separately. For the translation part, we shift the point cloud and the corner in each frame, in order to make the axis point of corner extracted from the previous stage locating at the origin of the new reference frame. For the rotation step, we need to make sure the three axes of the corner coinciding with the positive X, Y, and Z axis respectively.

To do this, we estimate the first rotation matrix by rotating the first axis so that it coincides with the positive X-axis, then the second rotation matrix is estimated by rotating the third axis so that coinciding with the positive Y-axis. Aggregate rotation matrix is estimated by multiplying the first and second rotation matrix.

After this rotation, we project all marked points onto their corresponding plane, forming a smooth plane surface.

The Pseudo-code of the algorithm is shown in Algorithm 4. The original Matlab code is in Appendix F.

And the result of this process on point cloud is shown in Fig 9.

Algorithm 4 Corners and point clouds normalization

```
Input: $Frame\ Number$
Output: $Location : pointsmappedtoplane$
Output: $Corners : swappedcorners$
Output: $newPoints : newmarkedpoints(removingpointsoutsidethebox)$
1: function GETROTATEDPOINTS($frameNum$)
2:     $[Planes, Points] \leftarrow$ GETCLEANPLANE($frameNum$)
3:     $Corners \leftarrow Translate($ GETCORNER($frameNum, Planes, Points$)$)$
4:     $Location \leftarrow Translate(pc.Location)$
5:     $X \leftarrow [1, 0, 0]$
6:     $Y \leftarrow [0, 1, 0]$
7:     $Z \leftarrow [0, 0, 1]$
8:     $\%\ First, select\ a\ corner\ vector\ and\ make\ it\ coincide\ with\ the\ X$
9:     $project\_XY \leftarrow Project\ Corners2\ on\ XY\ plane$
10:    $rotation1 \leftarrow$ROTATEZ(Angle(projection_XY, X))
11:    $rotation2 \leftarrow$ROTATEY(Angle(rotation1 * Corners{2}, X))
12:    $\%\ Then, select\ another\ corner\ vector\ and\ make\ it\ coincide\ with\ the\ Y$
13:    $rotation3 \leftarrow$ROTATEX(Angle(rotation1 * rotation2 * Corners{4}, Y))
14:    $\%\ Finally, the\ aggregate\ rotation\ matrix$
15:    $rotation \leftarrow rotation1 * rotation2 * rotation3$
16:    for $i = 1 : numberofpoints$ do
17:        $Remark\ points\{i\}$
18:        $Points \leftarrow$ PROJECTPOINTONLINE(Location{i},[0 0 0],Corners{PN})
19:        $Location\{i\} \leftarrow rotation * Location\{i\} - Points$
20:        $Corners \leftarrow rotation * Corners;$
21:        $result \leftarrow [Location, Corners, Points]$
22:        return $result$
23:    end for
24: end function
```

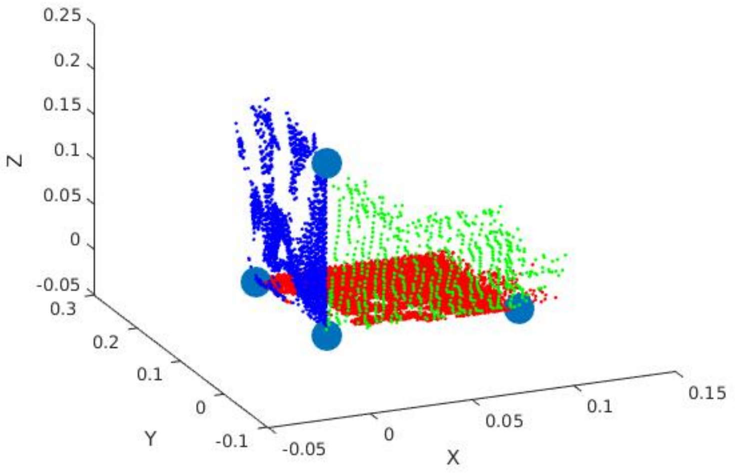

Figure 9: Frame 19 corners, rotated onto the main axis system.

2.5 Rotation and Translation Estimation and Box Fusion

2.5.1 Description

To make sure we have corner points and vectors matched correctly between two consecutive frames, we do this manually by marking the change to the red plane and the position of the blue plane in each frame. Then, giving the reference frame transformation from the previous stage, we map all of the points from the frame (other than 1,2,3,50 because we do not get any planes from these frames) to frame 4. Finally, we reconstruct the point clouds consecutively to fuse the whole box.

2.5.2 Marking System

This project uses manual marking for plane matching due to the similarity between planes, and the most of the plane points on original point cloud image is absent.

The provided `model.m` script is executed when Matlab starts, it sets several global variables used in plane extraction and model fusion. The description of detailed marking method are in the comment section of Appendix H.

2.5.3 Algorithm

Algorithm 5 shows the Pseudo-code of this step. Figures of the step-by-step fusion is included in Section 3. The Fig 10 shows the final fused model.

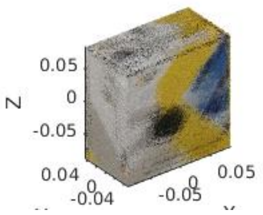

Figure 10: Final fused box

Algorithm 5 Rotation and Translation Estimation and Box Fusion

```
Input: $-$
Output: $PointClouds$
 1: function FUSEMODEL
 2:     $estimated\_size \leftarrow [0.1700, 0.090, 0.1700]$
 3:     $\%To\ make\ the\ box\ face\ you$
 4:     $rotation \leftarrow RotateX(270) * RotateZ(180)$
 5:     $\%Calculate\ the\ center\ of\ the\ box$
 6:     $Center \leftarrow -(estimated\_size/2)$
 7:     for $i = 1 : MaxFrameNum$ do
 8:         $pc \leftarrow getPC(i)$
 9:         $[planes, \sim] \leftarrow$ GETCLEANPLANE(i)
10:         $[location, \sim, points] \leftarrow$ GETROTATEDPOINTS(i)
11:         $Colour \leftarrow pc.Colour$
12:         $\%Consider\ the\ red\ plane$
13:         if $Change\ to\ the\ red\ plane$ then
14:             if $Change\ to\ top$ then
15:                 $rotation \leftarrow rotation * RotateX(90)$
16:             else if $Change\ to\ right$ then
17:                 $rotation \leftarrow rotation * RotateY(90)$
18:             else if $Change\ to\ bottom$ then
19:                 $rotation \leftarrow rotation * RotateX(270)$
20:             else if $Change\ to\ left$ then
21:                 $rotation \leftarrow rotation * RotateY(270)$
22:             end if
23:         end if
24:
25:         $\%Consider\ the\ blue\ plane$
26:         if $Blue\ plane\ exists$ then
27:             if $on\ top$ then
28:                 $rotation2 \leftarrow rotation * RotateZ(270)$
29:             else if $on\ right$ then
30:                 $rotation2 \leftarrow rotation * RotateZ(0)$
31:             else if $on\ bottom$ then
32:                 $rotation2 \leftarrow rotation * RotateZ(90)$
33:             else if $on\ left$ then
34:                 $rotation2 \leftarrow rotation * RotateZ(90)$
35:             end if
36:         end if
37:
38:         if $Only\ one\ plane\ in\ this\ frame$ then
39:             $Continue$
40:         end if
41:
42:         for $j = 1 : number\ of\ points$ do
43:             $\%Rotate\ back\ to\ original\ box$
44:             $Location(j) \leftarrow rotation2 * Location(j)$
45:             $this\_Center \leftarrow Center. * sign(rotation2 * [1; 1; 1])$
46:             $\%Translation$
47:             $Location(j) \leftarrow Location + this\_Center$
48:         end for
49:         $Remove\ data\ points\ outside\ the\ box$
50:         $Reconstruct\ the\ point\ clouds$
51:     end for
52: end function
```

3 Model evaluation

3.1 Images of Fused Point Clouds

The following images shows the fusing of the model at different stages of the fusing. Note that there are a total of 33 frames used for the fusing of the final model show in Figure 10 in Section 2.5.3.

The following frames are used in the fusion of the model, the later ones are not plotted due to similarity: 4,5,6,9,11,12,13,15,16,17,18,19,20,21,22,24,26,27,28,30,31,33,34,35,36,39,40,42,43,44,46,47,49.

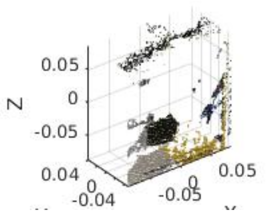

Figure 11: Frame 4

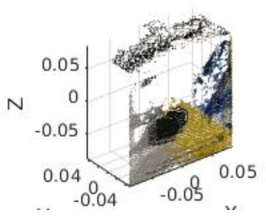

Figure 12: Frame 5

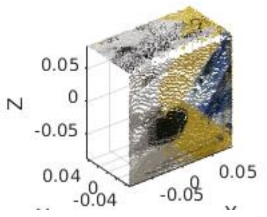

Figure 13: Frame 6

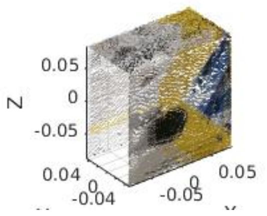

Figure 14: Frame 9

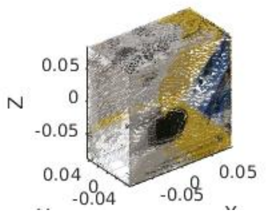

Figure 15: Frame 11

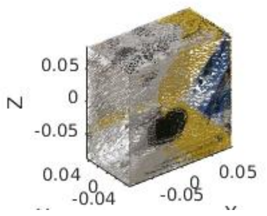

Figure 16: Frame 12

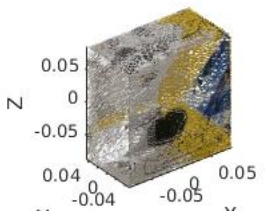

Figure 17: Frame 13

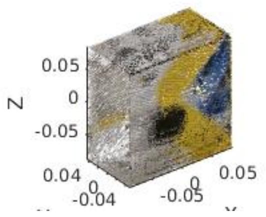

Figure 18: Frame 15

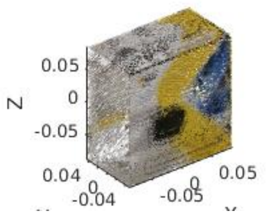

Figure 19: Frame 16

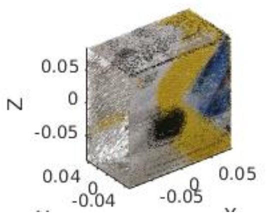

Figure 20: Frame 18

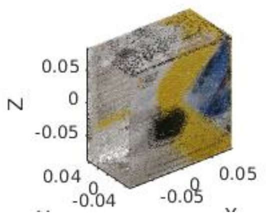

Figure 21: Frame 22

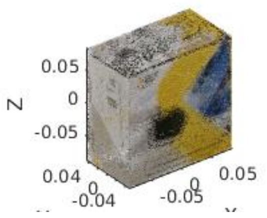

Figure 22: Frame 24

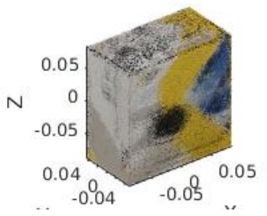

Figure 23: Frame 28

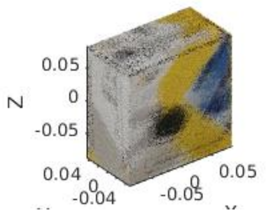

Figure 24: Frame 36

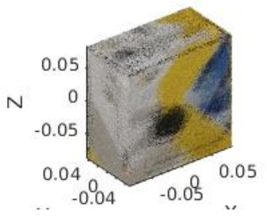

Figure 25: Frame 40

3.2 Large Plane Fusion Errors

Due to the assumption on the box shape, the plane normals obtained are orthogonal to each other. This change has led to a drastic decrease in the fused planes.

From the figures above in Section 3.1, the large planes are strictly fixed onto each faces of the box. This change have led to almost no angular error and corner error.

The only possible error from the plane fusion can come from the transformation in `getRotatePoints`. In this section, the points are projected onto the XYZ coordinate system. However, due to the erraticity caused during recording of the frames, the planes all have an angle of intersection greater than 90. This intersection angle may lead to the shrinking or rotation error on the extracted planes. The following Figure shows such error, note that the error is barely noticeable when comparing the images.

The blurry planes in the final image are caused by this effect, even in the case when large error has occured, the frame could be removed easily by modifying `model.m` file.

Fig 26 and Fig 27 shows the difference caused by the error. Note the shadow added at top of the plane, the increase in brightness, and the image is slightly shifted upward.

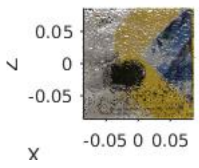

Figure 26: Fused Box at frame 13

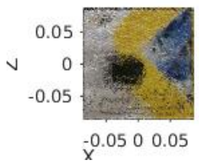

Figure 27: Fused Box at frame 15

3.3 Angular and Corner Position Error

Due to the assumption of box shape, such errors are eliminated during the fusion of the box. Hence the cumulative error required could not be plotted. Corner Positions are fixed by the model.m, so no position error could be made in the project.

Instead, this project will plot a chart of the angular error eliminated during each step.

Figure 28 shows the (average) angular error in each frame. Red bar shows the average error between 3 planes are used, blue bars shows the error for 2 planes.

Also, in planes with higher angular error eliminated, the planes usually have a higher noise. However, these errors are not removed due to the overall low noise value.

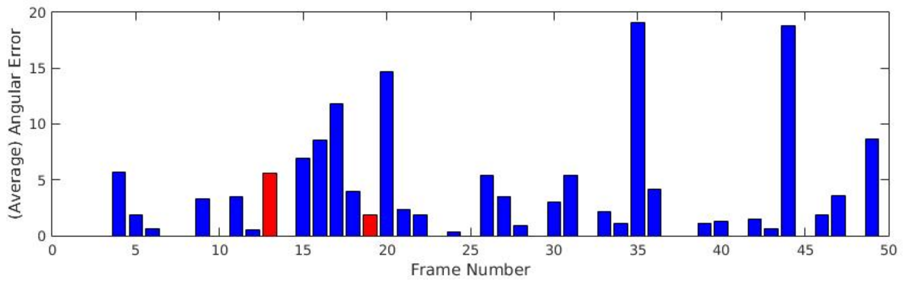

Figure 28: (Average) Angular Error versus Frame Number

3.4 Root Mean Square Error (RMSE)

Figure 29 shows the plotted average Root Mean Square Error of each frames. The RMSE error metric is very sensible to outliers. But with the outlier removal in the plane cleaning process, a lot of the outliers are removed, this leads to a rather stable distribution of the error values.

Note that some frames with higher RMSE (namely frame 35 and 45). This have lead to a high angular error shown in Figure 29 above. This is not reflected in other frames due to low amount of planes obtained after cleaning plane. As planes with high errors are usually rejected.

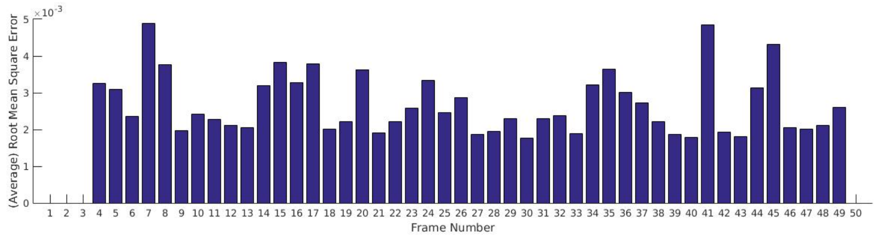

Figure 29: Average Root Mean Square Error of Planes in each image.

For deeper evaluation of the error, Root Mean Absolute Error (RMAE) is also plotted for the project, where RMAE are sensible to small errors instead of outliers. However, a similar plot (Figure 30) is obtained, this essentially means that most of the outliers are in fact removed. Combining with the Angular Error, this high error rate might be caused by the the misfit in the orientation of the planes.

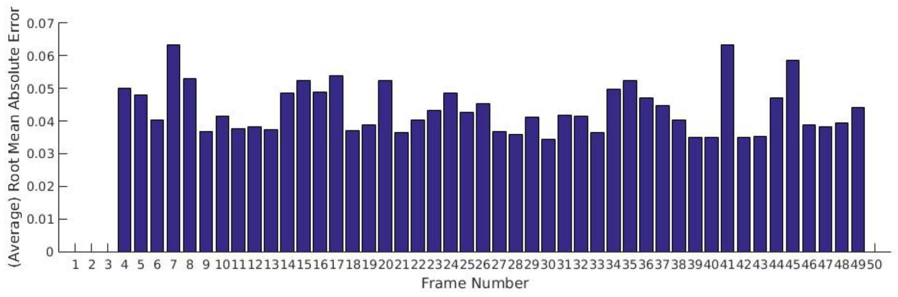

Figure 30: Average Root Mean Absolute Error of Planes in each image.

3.5 Overall Performance and Possible Failures

Overall, the project obtained a relatively clean box with clear surfaces. However, the time cost of fusing the box is still very time-consuming, which took about 2 minute to form the final model.

One of the reason that leads to the poor speed may be the cleaning plane part. Where multiple times of this time-consuming algorithm is performed, in hind-sight this procedure may not be necessary due to the frame could be ignored directly from `model.m` file. Also, a lot of repetitive operation is used in the project. The point are repeatedly marked and projected throughout the project. This also makes the code's structure hard to understand and created a lot of misunderstanding during both coding and creating of the report.

During the mask extraction stage, all the light blue colors are removed. If the new box contains such color, it would be removed incorrectly.

Due to the nature of the frame set, not enough plane contains 3 planes. This could potentially causes the estimation of location of third plane to be wrong in the Rotate Point step in the project. Also, the handling of this case is not thoroughly tested. Fig 28 shows that there are only 2 frame obtained actually contains 3 useful plane. This limit of test case could probably make the rotation calculated to be 90 degrees wrong. However, this error could be fixed with the `model.m` by changing the orientation.

3.6 Table of average RMSE in each frame

Image name	number of pixels	number of planes	average RMSE
1	0	0	N/A
2	0	0	N/A
3	0	0	N/A
4	6468	2	0.00325704261814754
5	8353	2	0.00308286767798611
6	11609	2	0.00235842065238900
7	2260	1	0.00488807060338564
8	2416	1	0.00376793312492517
9	11247	2	0.00196333671561034
10	9528	1	0.00241252775742022
11	9532	2	0.00227214416179784
12	7451	2	0.00211971044822282
13	3341	3	0.00204100727700222
14	2445	1	0.00320396658035265
15	11642	2	0.00381647358422069
16	6976	2	0.00328236431691562
17	9214	2	0.00378691254638582
18	11634	2	0.00200234314733579
19	7983	3	0.00220995176978304
20	2547	2	0.00363094831433755
21	10677	2	0.00189908399042462
22	11031	2	0.00221423739207694
23	9124	1	0.00258059149024027
24	11679	2	0.00332895449278854
25	4814	1	0.00246903180420358
26	5594	2	0.00287693104549114
27	10454	2	0.00187307024748445
28	10662	2	0.00194914242237185
29	9014	1	0.00228872457202993
30	10989	2	0.00176204488054514
31	8884	2	0.00228634363984092
32	5190	1	0.00236850564785098
33	7983	2	0.00187994303893711
34	5424	2	0.00320881566106691
35	3990	2	0.00363480957548286
36	7874	2	0.00300882760716597
37	4491	1	0.00272483638919232
38	5372	1	0.00221301979238047
39	7901	2	0.00186682583520425
40	8778	2	0.00178329528081118
41	2661	1	0.00485246559679819
42	5432	2	0.00193265584036207
43	8276	2	0.00180713930697701
44	4279	2	0.00313811247393757
45	2021	1	0.00432034233747345
46	11373	2	0.00204845955510231
47	12157	2	0.00200336568431949
48	10560	1	0.00211567933625873
49	10126	2	0.00260783666198434
50	0	0	N/A

Table 1: Experimental results

A External Code Used

For plane extraction, the project used the MATLAB functions provided at:

`https://www.inf.ed.ac.uk/teaching/courses/av/MATLAB/TASK3/`

Namely, the `getAllPoints` function were used, the `getPlane` function are adapted from the patch growing algorithm.

For transformation linear algebra functions, the following functions are used:

- getPlaneIntersection.m
- projectPointOnLine.m
- getPlaneLineIntersection.m
- rotx.m
- roty.m
- rotz.m

The `rot` functions are from later versions of MATLAB, and the other functions are obtained from:

`https://uk.mathworks.com/matlabcentral/fileexchange/17618-plane-intersection`

`https://uk.mathworks.com/matlabcentral/fileexchange/17751-straight-line-and-plane-intersection`

`https://uk.mathworks.com/matlabcentral/fileexchange/7844-geom2d?focused=8114527&tab=function`

There are also some more efficient distance metric functions (e.g. Euclidean Distance between point/-plane) are adapted from various StackOverflow answers.

B References

Velasquez, H.C. and Fisher, B. 2018. Lab 4.[Online]. School of Informatics, University of Edinburgh. [Accessed 28 March 2018]. Available from: `https://www.inf.ed.ac.uk/teaching/courses/av/DEMOS/lab_4.pdf`

Fisher, B. 2018. Advanced Vision Practical. [Online]. School of Informatics. [Accessed 28 March 2018]. Available from: `https://www.inf.ed.ac.uk/teaching/courses/av/PRACTICALS/PRACT1/pracl1718.pdf`.

C Extract the relevant data from each point cloud

```
%%
function [IMG] = getImage(frameNum, plot)
% Input: frameNum: (int)      The number of frame.
% plot: (boolean) Use of Imshow or not.
% return the image as RGB array.
global pcl_train;

if nargin < 2
plot = false;
end
```

```
img = pcl_train{frameNum}.Color;
IMG = flip(rot90(reshape(img, [512, 424, 3])));

if plot
imshow(IMG);
end

end
%%
function [XYZ] = getDepth(frameNum)
% Input: frameNum: (int)      The number of frame.
% return the xyz locs.
global pcl_train;

img = pcl_train{frameNum}.Location;
XYZ = flip(rot90(reshape(img, [512, 424, 3])));

end
%%
function [mask] = getMask(frameNum)
% Input: frameNum: (int) The number of frame.
% return the mask.
offset = [ 0.15, 0.20, 0.25];
center = [-0.71,-0.30, 0.81];

IMG = getImage(frameNum);
XYZ = getDepth(frameNum);

% BG removal: clears boundary
% Not very useful, table and box have similar color
% BG = double(rgb2gray(getImage(pcl_train,1)));
% mask = abs(double(rgb2gray(IMG))-BG) > 20;

% XYZ boundary
mask = (XYZ(:,:,1) > center(1)-offset(1));
mask(XYZ(:,:,1) > center(1)+offset(1)) = 0;
mask(XYZ(:,:,2) < center(2)-offset(2)) = 0;
mask(XYZ(:,:,2) > center(2)+offset(2)) = 0;
mask(XYZ(:,:,3) < center(3)-offset(3)) = 0;
mask(XYZ(:,:,3) > center(3)+offset(3)) = 0;

%Erode to remove noise
SE1 = strel('square',3);
SE2 = strel('square',4);
mask1 = imerode(imdilate(mask,SE1),SE2);

%Hand HSV
mask2 = removeSkin(IMG);
%Disk because finger?
SE3 = strel('disk',3);
SE4 = strel('disk',7);
mask2 = imdilate(imerode(mask2,SE3),SE4);

%Background Blue
mask3 = removeBlue(IMG);
SE5 = strel('square',10);
mask3 = imerode(imdilate(mask3,SE5),SE5);
```

```
%Combine
maskf = bitand(mask1,~mask2);
maskf = bitand(maskf,~mask3);
%Remove points with wrong XYZ
mask = bitand(maskf,mask);
%mask = mask2;
%mask = mask3;
end

%%
function [pc] = getPC(frameNum, filter)
% GETPC Summary of this function goes here
% Detailed explanation goes here

if nargin < 2
filter = true;
end

IMG = getImage(frameNum);
XYZ = getDepth(frameNum);
mask = getMask(frameNum);

points={};
rgbs={};

count = 1;
for ii = 1:424
for jj = 1:512
if (~filter) || (mask(ii,jj)~=0)
point = IMG(ii,jj,:);
if ~(point(1)<=0 && point(2)<=0 && point(3)<=0) %?
points{count} = XYZ(ii,jj,:);
rgbs{count} = point;
count = count + 1;
end
end
end
end

pc = pointCloud(cell2mat(points), 'Color', cell2mat(rgbs));
showPointCloud(pc)

end
```

D Extract the planes from each point cloud

```
%%
function [planes, pts] = getPlanes(xyz, plot)
% GETPLANES return a list of candidate planes
% Input: xyz: a list of coordinate with N x 3 shape.
%        plot: Optional, show the colored planes.
%------ Set parameter ------------------------------------------------
PLANENUM = 5;       %Max number of planes needed.

PATDISTOL = 0.01;   %Patch-Distance Tolerance.
PATMINPNT = 10;     %Minimum points for a patch to form.
PATMAXRES = 0.0005; %Patch Plane Fitting Error Tolerance.
PATTRIES = 500;     %Limit attempts to find patches.
```

```
PLANEGROWLIM = 5;    %Limit of times to grow a plane.
PLANEREFITLIM = 50;  %Number of newpoints needed to refit.
PLANEDISTOL = 0.006; %Plane-Distance Tolerance.
PLANEPDISTOL = 2;  %Tolerance between point on same plane.
PLANEMAXRES = 0.005;%Max Average Fitting Error Tolerance

global SEED; %Random seed for RandPerm
s = RandStream('mt19937ar', 'Seed', SEED);
%---End setting-------------------------------------------------------

if nargin < 2
plot = false;
end

[NPts, W] = size(xyz);
assert(W==3, 'Nx3␣shape␣pls');

planes = zeros(PLANENUM, 4);
remaining = xyz;

if plot
figure(1)
clf
hold on
%plot3(remaining(:,1),remaining(:,2),remaining(:,3),'k.');
end

indexs = 1:NPts;
pts = zeros(NPts,1);

for ii = 1:PLANENUM
%----Init----
fitlist = [];
newlist = [];
fitindex = [];
breaknextfor = false;
%---Patch selection----------------------------------------
L = size(remaining,1); % 8884
idxs = randperm(s,L);
for jj = 1:L
pnt = remaining(idxs(jj),:);
dists = sqrt(sum((remaining-ones(size(remaining))*diag(pnt)).^2,2));
mask = dists < PATDISTOL;
if sum(mask) > PATMINPNT %Minimum points for a patch to form.
fitlist = remaining(mask,:);
% Too lazy to optimize this.
[plane,resid] = fitplane(fitlist);
if resid < PATMAXRES
remaining = remaining(~mask,:); % update remaining mask
planes(ii,:) = plane;
fitindex = indexs(mask);
indexs = indexs(~mask);
break
end
end
%Skipped when patch is found.
%When no more patch could be found, skip the next for
```

```
%loop to printout result.
if jj == L || jj == PATTRIES
breaknextfor = true;
%disp('no more patches could be found!')
end
end
%----Grow Patch to Plane----------------------------------------
for jj = 1:PLANEGROWLIM
if breaknextfor
breaks
end
distoplane = abs(remaining * planes(ii,1:3)' + planes(ii,4));
% |ax1+bx2+cx3+d|/sqrt(a^2+b^2+c^3+d^2)
mask = distoplane < PLANEDISTOL;
%Is this really necessary? It's very slow.s
%for k = 1:size(mask,1)
%    if ~mask(k)
%        continue
%    end
%    for l = 1:size(fitlist,1)
%        if norm(remaining(k,:)-fitlist(l,:)) < PLANEPDISTOL
%            break;
%        end
%        if l == size(fitlist,1)
%            mask(k) = false;
%        end
%    end
%end
newlist = cat(1,fitlist,remaining(mask,:));

if sum(mask) > PLANEREFITLIM %Number of newpoints needed to refit.
[newplane,fit] = fitplane(newlist);
if fit > PLANEMAXRES * size(newlist,1)
break
end
planes(ii,:) = newplane;
fitlist = newlist;
remaining = remaining(~mask,:);
fitindex = cat(2,fitindex,indexs(mask));
indexs = indexs(~mask);
continue
end
break %Weird control loop in bob's code.
end
%----End Grow Plane-----------------------------------------------
pts(fitindex) = ii;
try
if plot
if ii == 1
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'r.');
elseif ii==2
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'b.');
elseif ii == 3
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'g.');
elseif ii == 4
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'c.'); % Cyan
else
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'m.'); % Magenta
```

```
end
end
end
end
end

%%
function [planesout,pointsout] = getCleanPlane(frameNum, plot )
% Input: plot: boolean, optional, self explanatory.
% return a list of planes, and a list of plane allocation.
% set detailed parameters at getPlanes function, dont do it here.
%---Set parameter------------------------------------------------
MINPNT = 400; %Minimum point needed in plane for it to count.
MAXANG = 15;    %Maximum angle in merge plane phase.
REJANG = 65;   %Minimum angel needed between axis.
ROUND = 5;     %Number of times to run getPlane.m
OUT = 10;       %Maximum number of planes needed.
PLANEDISTOL = 0.01; %Plane-Distance Tolerance.

%---End setting --------------------------------------------------
try
pc = getPC(frameNum);
catch Error
planesout = [];
pointsout = [];
return
end

if nargin < 2
plot = false;
end

XYZ = squeeze(pc.Location);

planes = [];
points = [];

planesout = [];

%--Get the planes
for i = 1:ROUND
[tempplane, temppoint] = getPlanes(XYZ);
planes = cat(1,planes,tempplane);
points = cat(1,points,histc(temppoint,1:5));
end

%Get most possible plane.
for i = 1:OUT
if numel(points) == 0
break
end
%Sort the remaining
[points, indexs] = sort(points,'descend');
if points(1) < MINPNT %Stop if no more good planes
break
end
%Get a list of similar planes
planes = planes(indexs,:);
```

```
mask = false(size(points));
mask2 = false(size(points));
u = planes(1,1:3);
for jj = 1:size(points)
if points(jj) < MINPNT
continue
end
v = planes(jj,1:3);
angle = atan2d(norm(cross(u,v)),dot(u,v)); %minimal angle
if angle < MAXANG
mask(jj) = true;
end
if angle < REJANG
mask2(jj) = true;
end
end
%'Average' the candidate planes.
if sum(mask)==1
planesout = cat(1,planesout,planes(mask,:));
else
planesout = cat(1,planesout,mean(planes(mask,:)));
end

%Remove planes have similar angles
points = points(~mask2);
planes = planes(~mask2,:);
end

%prepare output
%Mark all the points
NPts = size(XYZ,1);
planenum = size(planesout,1);
pointsout = zeros(NPts,1);
index = 1:NPts;
count = 1;
newplaneouts = [];
for i = 1:planenum
%Get distant of everypoint to the plane checking.
distoplane = abs(XYZ * planesout(i,1:3)' + planesout(i,4));
mask = distoplane < PLANEDISTOL; %points in range
%Check size of plane.
mask2 = bitand(mask, pointsout == 0);
if sum(mask2) < MINPNT
continue
end

newplaneouts = cat(1,newplaneouts,planesout(i,:));

mask2 = pointsout > 0; %Currently Marked points
mask2 = bitand(mask, mask2); %Intersect Point
mask = bitand(mask, ~mask2); %Remove the intersection points.

pointsout(index(mask)) = count;
pointsout(index(mask2)) = -1;

%index = index(~mask);
%XYZ = XYZ(~mask,:);
count = count + 1;
```

```
end

%Re-mark cross points
for i = 1:size(pointsout,1)
if pointsout(i) == -1
distoplane = abs(XYZ(i,:)*newplaneouts(:,1:3)'+newplaneouts(:,4)');
[~, pointsout(i)] = min(distoplane);
end
end

%plot stuff
if plot
figure(1)
clf
hold on
for i = 1:planenum
mask = pointsout == i;
newlist = XYZ(mask,:);
if i == 1
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'r.');
elseif i==2
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'b.');
elseif i == 3
plot3(newlist(:,1),newlist(:,2),newlist(:,3),'g.');
end
end

mask3 = pointsout == -1;
mask4 = pointsout == 0;
intersect = XYZ(mask3,:);
XYZ = XYZ(mask4,:);
plot3(intersect(:,1),intersect(:,2),intersect(:,3),'y.');
plot3(XYZ(:,1),XYZ(:,2),XYZ(:,3),'k.');
end
planesout = newplaneouts;
size(planesout,1)
end
```

E Estimate the 3D positions of the corners where planes meet

```
function [C] = getCorner(arg1, planes, points)
%GETCORNER Summary of this function goes here
%   INPUT: arg1 (frameNum / pc): if frameNum: getPC and other arguments
%if pc: need other argyments
%            planes, points: from getCleanPlane.
%   OUTPUT: C: if one plane: []
%if two plane: 2x3 matrix of 2 corner points
%if  3  plane: 4x3 matrix of 4 corner points

%---Setting Parameter---
cutoff      = 4;%  For winsor the connect line
cutoffshift = 2;%  Shift % for 3 plane case.
cutoff2     = 3;%  Cutoff % for 3/4th point
plot = true;
%---Initialize---
%Problematic frames: 7 34 42
C = zeros(3,3);
if nargin < 2
```

```
[planes points] = getCleanPlane(arg1);
try
arg1 = getPC(arg1);
catch Error
return
end
end
XYZ = squeeze(arg1.Location);

%--- Ignore if only 1 plane---
NPS = size(planes,1);
if NPS <= 1
C = [];
plot3(XYZ(points==1,1),XYZ(points==1,2),XYZ(points==1,3),'b.');
return
end

% if two lines
if plot
clf;
hold on;
plot3(XYZ(points==1,1),XYZ(points==1,2),XYZ(points==1,3),'b.');
plot3(XYZ(points==2,1),XYZ(points==2,2),XYZ(points==2,3),'g.');
plot3(XYZ(points==3,1),XYZ(points==3,2),XYZ(points==3,3),'r.');
end

mask = bitor(points == 1,points==2);
XYZ2 = XYZ(mask,:);

%Project all points on the connect line.
[P,V,~] = getPlaneIntersection(planes(1,:),planes(2,:));
tps = zeros(size(XYZ2,1),1);
for i = 1:size(XYZ2,1)
[point, tp] = projectPointOnLine(XYZ2(i,:),P,V);
XYZ2(i,:) = point;
tps(i) = tp;
end

%Winsor the line
lower = cutoff; % 4
upper = 100 - lower; % 96
lim = prctile(tps,[lower,upper]);

%If third plane, shift the winsor direction.
if NPS == 3
[NP,~] = getPlaneLineIntersection(planes(3,:),P,V);
[~, tp3] = projectPointOnLine(NP,P,V);
if abs(lim(1)-tp3) < abs(lim(2)-tp3)
lower = cutoff - cutoffshift;
upper = 100 - lower;
lim = prctile(tps,[lower,upper]);
else
lower = cutoff + cutoffshift;
upper = 100 - lower;
lim = prctile(tps,[lower,upper]);
end
end
```

```
%Remove outlier in the line
mask = bitand(tps>lim(1),tps<lim(2));
XYZ2 = XYZ2(mask,:);
tps2 = tps(mask);
[~, MAX] = max(tps2);
[~, MIN] = min(tps2);
C(1,:) = XYZ2(MAX,:);
C(2,:) = XYZ2(MIN,:);

% In 3-plane's case, points could be shifted.
if NPS == 3
[NP,~] = getPlaneLineIntersection(planes(3,:),P,V);
dists = sqrt(sum((C(1:2,:)-ones(size(C(1:2,:)))*diag(NP)).^2,2));
[~, MAX] = max(dists);
C2 = zeros(4,3);
C2(1,:) = NP;
C2(2,:) = C(MAX,:);
% Also get a third point in plane.
mask = points > 1;
XYZ3 = XYZ(mask,:);
tps = zeros(size(XYZ3,1),1);
for i = 1:size(XYZ3,1)
[point, tp] = projectPointOnLine(XYZ3(i,:),NP,planes(1,1:3));
XYZ3(i,:) = point;
tps(i) = tp;
end
lim = prctile(tps,[cutoff2,100-cutoff2]);
mask = bitand(tps>lim(1),tps<lim(2));
XYZ3=XYZ3(mask,:);
tps=tps(mask);
if mean(tps) < 0
[~, MIN] = min(tps);
C2(3,:) = XYZ3(MIN,:);
XYZ3 = XYZ3(tps < 0,:);
else
[~, MAX] = max(tps);
C2(3,:) = XYZ3(MAX,:);
XYZ3 = XYZ3(tps > 0,:);
end

%And fourth point
mask = bitor(points==1,points==3);
XYZ = XYZ(mask,:);
tps = zeros(size(XYZ,1),1);
crossV = cross(V,planes(1,1:3));
for i = 1:size(XYZ,1)
[point, tp] = projectPointOnLine(XYZ(i,:),NP,crossV);
XYZ(i,:) = point;
tps(i) = tp;
end
lim = prctile(tps,[cutoff2,100-cutoff2]);
mask = bitand(tps>lim(1),tps<lim(2));
XYZ=XYZ(mask,:);
tps=tps(mask);
if mean(tps) < 0
[~, MIN] = min(tps);
C2(4,:) = XYZ(MIN,:);
XYZ = XYZ(tps < 0,:);
```

```
else
[~, MAX] = max(tps);
C2(4,:) = XYZ(MAX,:);
XYZ = XYZ(tps > 0,:);
end

%Put C back
C = C2;
if plot
plot3(XYZ3(:,1),XYZ3(:,2),XYZ3(:,3),'b.');
plot3(XYZ(:,1),XYZ(:,2),XYZ(:,3),'g.');
end
end

%Add optional line on 2 plane
if NPS == 2
NP = C(1,:);
mask = points == 2;
XYZ3 = XYZ(mask,:);
tps = zeros(size(XYZ3,1),1);
v = planes(1,1:3);
for i = 1:size(XYZ3,1)
[point, tp] = projectPointOnLine(XYZ3(i,:),NP,v);
XYZ3(i,:) = point;
tps(i) = tp;
end
lim = prctile(tps,[cutoff2,100-cutoff2]);
if mean(tps) < 0
mask = bitand(tps>lim(1),tps<0);
XYZ3=XYZ3(mask,:);
tps=tps(mask);
[~, MIN] = min(tps);
C(3,:) = XYZ3(MIN,:);
else
mask = bitand(tps<lim(2),tps>0);
XYZ3=XYZ3(mask,:);
tps=tps(mask);
[~, MAX] = max(tps);
C(3,:) = XYZ3(MAX,:);
end

mask = points == 1;
XYZ = XYZ(mask,:);
tps = zeros(size(XYZ,1),1);
crossV = cross(V,planes(1,1:3));
for i = 1:size(XYZ,1)
[point, tp] = projectPointOnLine(XYZ(i,:),NP,crossV);
XYZ(i,:) = point;
tps(i) = tp;
end
lim = prctile(tps,[cutoff2,100-cutoff2]);
if mean(tps) < 0
mask = bitand(tps>lim(1),tps<0);
XYZ=XYZ(mask,:);
tps=tps(mask);
[~, MIN] = min(tps);
C(4,:) = XYZ(MIN,:);
else
```

```
mask = bitand(tps<lim(2),tps >0);
XYZ=XYZ(mask,:);
tps=tps(mask);
[~, MAX] = max(tps);
C(4,:) = XYZ(MAX,:);
end

if plot
plot3(XYZ3(:,1),XYZ3(:,2),XYZ3(:,3),'b.');
plot3(XYZ(:,1),XYZ(:,2),XYZ(:,3),'g.');
end

end


if plot
plot3(XYZ2(:,1),XYZ2(:,2),XYZ2(:,3),'r.');
scatter3(C(:,1),C(:,2),C(:,3),100,'filled');
hold off;
end

end
```

F Corners and point clouds normalization

```
function [XYZ, C, points] = getRotatedPoints(arg1, C)
%GETROTATEDPOINTS Project the coordinates onto xy,yz,xz planes.
%   arg1 : frameNum/ pc: if not frameNum, use the pc to extract corner.
%                     C: the extracted corner for rotation.
%   output: XYZ: points mapped to plane
%             C: swapped corners
%        points: new marked points (removing points outside the box)


%-----Setting Parameter-----
removeNegative = true;
plot = true;
%-----Initialize -----

[planes, points] = getCleanPlane(arg1);
if nargin < 2
try
arg1 = getPC(arg1);
catch Error
return
end
C = getCorner( arg1, planes, points );
end
XYZ = squeeze(arg1.Location);

if size(C,1) == 0
return
end

X = [1 0 0];
Y = [0 1 0];
Z = [0 0 1];
```

```
Norm = C(1,:);
C = bsxfun(@minus, C, Norm);
XYZ = bsxfun(@minus, XYZ, Norm);

%Calculate rotation matrix of first axis -- eliminate y;
D = C(2,:);
D(3) = 0;
angle = atan2d(norm(cross(X,D)),dot(X,D));
r = rotz(angle);
D = r * C(2,:)';
if abs(D(2)-0)>0.001
r = rotz(360 - angle);
D = r * C(2,:)';
end

%Calculate rotation matrix of first axis -- eliminate z;
angle = atan2d(norm(cross(X,D)),dot(X,D));
r2 = roty(angle);
D = r2 * (r * C(2,:)');
if abs(D(3)-0)>0.001
r2 = roty(360 - angle);
D = r2 * (r * C(2,:)');
end

%Calculate rotation matrix of second axis
D = r2 * r * C(4,:)';
angle = atan2d(norm(cross(Y,D)),dot(Y,D));
r3 = rotx(angle);
D = r3 * r2 * r * C(4,:)';
if abs(D(3)-0)>0.001
r3 = rotx(360 - angle);
D = r3 * r2 * r * C(4,:)';
end

%Get final rotation matrix.
r = r3 * r2 * r;

%if mismatch, rotate more and swap axis.
%Rotate to negative, (another rotation below rotate it back)
D = r * C(3,:)';
if D(3) > 0
if size(planes,1) == 2
r = roty(180) * r;
D = C(2,:) - C(1,:);
C(2,:) = C(2,:) - 2 * D;
XYZ = bsxfun(@minus, XYZ, D);
end
end

%Put red plane on xy plane, second plane on yz plane.
%For better merging of planes.
%i hate linear algebra.
r = rotz(90) * rotx(180) * r;

for i=1:numel(points)

%Project all points on their plane.
PN = points(i);
```

```
if PN == 1
PN = 3;
elseif PN == 2
PN = 4;
elseif PN == 3
PN = 2;
else
%Throw away the urchins
XYZ(i,:) = [0 0 0];
continue
end
point = projectPointOnLine(XYZ(i,:),[0 0 0],C(PN,:));
XYZ(i,:) = XYZ(i,:) - point;
XYZ(i,:) = r * XYZ(i,:)';

end

C = (r * C')';

% Optionally remove all points with negative dim:
if removeNegative
%some error here
mask = sum(XYZ < -0.0002,2) > 0;
points(mask) = 0;
end

% Optionally remove outliers
% NO, leave the work to C

if plot
clf;
hold on;
xlabel('X')
ylabel('Y')
zlabel('Z')
plot3(XYZ(points==1,1),XYZ(points==1,2),XYZ(points==1,3),'r.');
plot3(XYZ(points==2,1),XYZ(points==2,2),XYZ(points==2,3),'b.');
plot3(XYZ(points==3,1),XYZ(points==3,2),XYZ(points==3,3),'g.');
%plot3(XYZ(points==-1,1),XYZ(points==-1,2),XYZ(points==0,3),'k.');
scatter3(C(:,1),C(:,2),C(:,3),300,'filled');
hold off;
end

end
```

G Corners and point clouds normalization

```
function [pc] = fuseModel()
%COM Summary of this function goes here
%   Detailed explanation goes here

global estimated_size model;

%Optionally remove points outside ze box
removeOutside = true;
%plot = true; %you know you want to plot.

XYZs = [];
```

```
RGBs = [];

%Rotation and shifting needed to apply.
%r =  rotx(90);
r = rotx(270) * rotz(180);
norm = - (estimated_size / 2);

for i = 1:50
try
pc = getPC(i);
catch Error
continue
end
[planes,~] = getCleanPlane(i);

[XYZ, ~, points] = getRotatedPoints(i);
IMG = squeeze(pc.Color);

%Compute Normalization and Rotate for Next main plane.
rotate = model(i,1);
if rotate ~= 0

if rotate > 10
rot1 = fix(rotate/10);
rotate = fix(mod(rotate,10));
if rot1 == 1
r = r * rotx(90);
elseif rotate == 3
r = r * rotx(270);
end
end

if rotate == 1
r = r * rotx(90);
elseif rotate == 3
r = r * rotx(270);
elseif rotate == 2
r = r * roty(90);
elseif rotate == 4
r = r * roty(270);
end

end

local_rotate = model(i,2);

if local_rotate >= 1
if local_rotate == 1
r2 = r * rotz(270);
elseif local_rotate == 2
r2 = r * rotz(0);
elseif local_rotate == 3
r2 = r * rotz(90);
else
r2 = r * rotz(180);
end
else
r2 = r;
```

```
end

%Normalize the points so the model center at (0,0).
if size(planes,1) <= 1


continue
end

for j=1:numel(points)
XYZ(j,:) = r2 * XYZ(j,:)';

thisnorm = norm .* sign((r2*[1;1;1])');
XYZ(j,:) = XYZ(j,:) + thisnorm;
end

mask = points > 0;
if removeOutside
mask = bitand(mask, XYZ(:,1) < estimated_size(1)/2+0.001);
mask = bitand(mask, XYZ(:,1) >-estimated_size(1)/2 -0.001);
mask = bitand(mask, XYZ(:,2) < estimated_size(2)/2+0.001);
mask = bitand(mask, XYZ(:,2) >-estimated_size(2)/2 -0.001);
mask = bitand(mask, XYZ(:,3) < estimated_size(3)/2+0.001);
mask = bitand(mask, XYZ(:,3) >-estimated_size(3)/2 -0.001);
end

XYZs = cat(1,XYZs,XYZ(mask,:));
RGBs = cat(1,RGBs,IMG(mask,:));
%pc = pointCloud(XYZ(mask,:), 'Color', IMG(mask,:));
pc = pointCloud(XYZs, 'Color', RGBs);
clf;
showPointCloud(pc)
xlabel('X')
ylabel('Y')
zlabel('Z')

pause();
end

end
```

H Model.m Location Marking File

```
%Records the details of change to surface, for better merge.

%Random Patches is used to find planes, fix the seed for reproducible
%results. Random seed for RandPerm:

    global SEED estimated_size model;
    SEED = 420;

%Estimated size of the model
%   Use getRotatedPoints function with plot to find this.
%       %Dimension of the X Z Y edge of the box

    estimated_size = [0.17 0.09 0.17];
    %Use the below one for testing
    %estimated_size = [0.9 0.7 0.5];
```

```
%Red is main plane, blue is the second (clockwise second if third
%plane exists) (It may show up as green).

%Use playPlane2D to estimate this.

%below is a matrix showing:
%   first column: Change to red plane
%       0:  not changed
%       1:  changed to top,
%       2:  changed to  right,
%       3:  changed to  bottom,
%       4:  changed to  left,
%       12/14/32/34: Multiple Changes
%   second colum: direction of the "blue" plane.
%       0:  one or less plane is shown,
%       1:  on top,
%       2:  on right,
%       3:  on bottom,
%       4:  on left,

    model = [0 0; ... % 1
              0 0; ... %
              0 0; ... %
              0 1; ... %
              0 1; ... % 5
              0 1; ... %
              1 0; ... %
              0 0; ... %
              1 3; ... %
              0 0; ... % 10
              0 1; ... %
              0 1; ... %
             12 4; ... %
              4 0; ... %
              1 3; ... % 15
              0 2; ... %
              0 1; ... %
              0 2; ... %
              2 4; ... %
              0 2; ... % 20
              2 4; ... %
              0 4; ... %
              0 0; ... %
              0 2; ... %
              2 0; ... % 25
              0 2; ... %
              2 4; ... %
              0 4; ... %
              0 0; ... %
              0 2; ... % 30
              2 4; ... %
              0 0; ... %
              1 3; ... %
              0 3; ... %
              0 2; ... % 35
              0 1; ... %
              1 0; ... %
```

```
          0 0; ... %
          0 1; ... %
          1 3; ... % 40
          0 0; ... %
          0 1; ... %
          0 1; ... %
          1 2; ... %
          0 0; ... % 45
          2 4; ... %
          0 4; ... %
          0 0; ... %
          0 1; ... %
          0 0];    % 50
```

I PlayPlane2D: For tracking the location of the planes

```
previmage = [];
for i=1:50
    a = getImage(i);
    mask = getMask(i);
    [A,B] = getCleanPlane(i);
    if numel(A) == 0
        %imshow(a);
        pause(0.1);
        continue
    end
    curi = i;
    count = 1;
    for i=1:424
        for j=1:512
            if mask(i,j)
                point = a(i,j,:);
                if ~(point(1)<=0 && point(2)<=0 && point(3)<=0);
                if B(count)==1
                    a(i,j,:) = [255 0 0];
                elseif B(count)==2
                    a(i,j,:) = [0 0 255];
                elseif B(count)==3
                    a(i,j,:) = [0 255 0];
                elseif B(count)==-1
                    a(i,j,:) = [255 255 0];
                end
                count = count + 1;
                end
            end
        end
    end
    if numel(previmage) == 0
        previmage = a;
        continue
    end
    [curi-1 curi]
    a2 = cat(2, previmage, a);
    previmage = a;
    imshow(a2);
    pause();
end
reload;
```