Project: 3D Perception

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

1. Filtering

First of all, we need to filter the point cloud received by the subscriber in order to eliminate the points that we don't need and also reduce the noise as much as we can.

The filters used were defined as functions outside the callback function (*pcl_callback*)

After converting the ROS message to PCL data, I applied a Statistical Outlier Filter, to get rid of the noise. There the threshold scale factor was set to **0.4**, after some tries, and the number of neighboring points to analyze was set to **50.** The resulting PC is shown in figure 2.

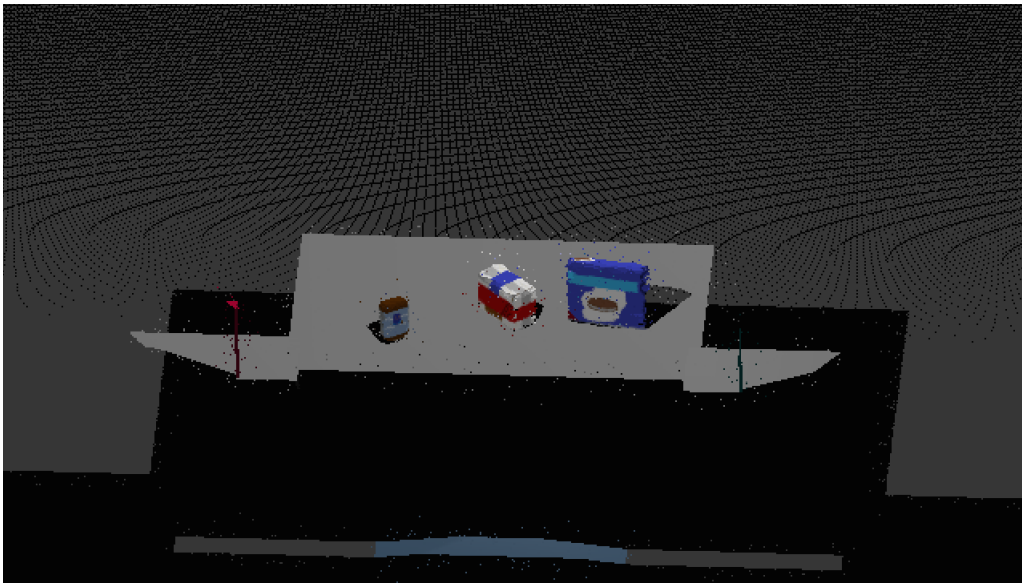

Figure 1. Original point cloud

For the downsampling filter, the voxel size was set to **0.01.** Result shown in figure 3.

The pass through filter was applied in the three dimensions in order to get only the table and the objects. The parameters are:

Axis	Minimum	Maximum
X	0.34	0.90
Y	-0.46	0.46
Z	0.5	0.9

The parameters were calculated analyzing the point cloud and then testing. The resulting PC is shown in figure 4.

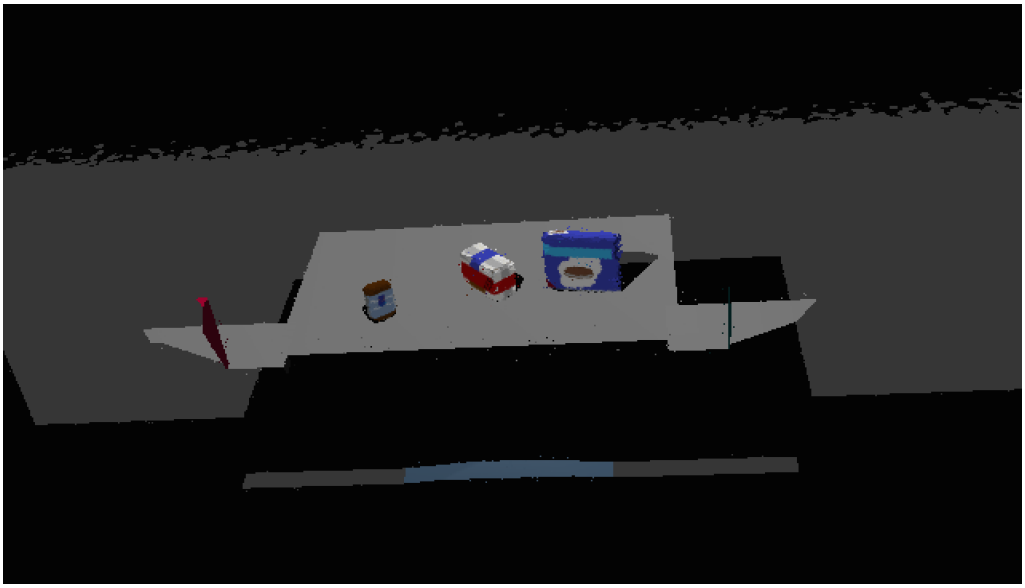

Figure 2. Statistical filter applied.

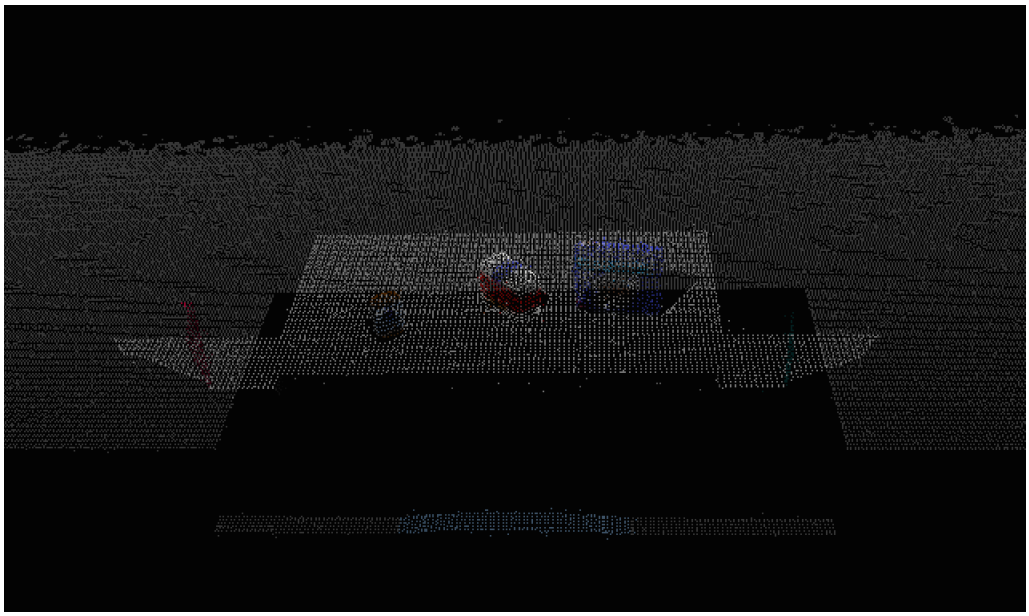

Figure 3. PC down sampled.

Having the table and objects points cloud, I applied RANSAC segmentation to extract the table. The maximum distance from the plane model was set to **0.01,** i. e., 1 [cm]. This parameter needs to be small in order to not cut the bottom of the objects. In figure 5 we have the table cloud and in figure 6 the objects cloud.

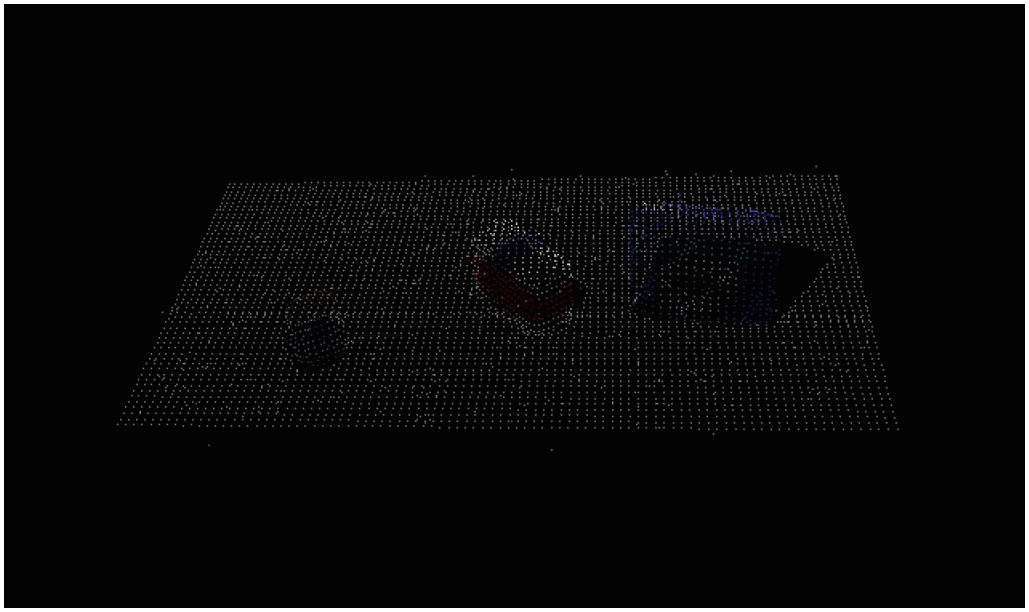

Figure 4. Pass Through Filter applied.

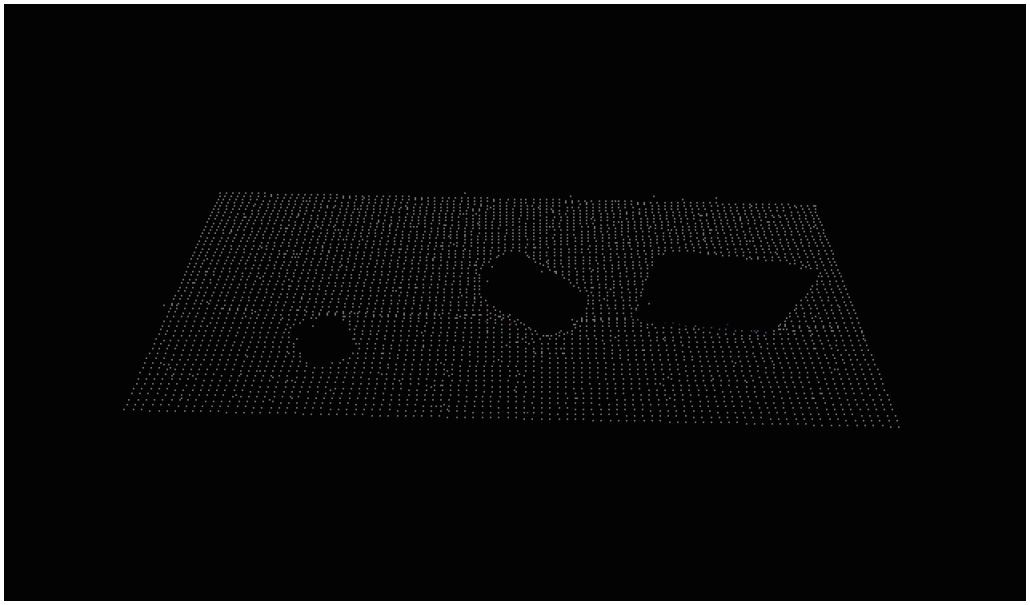

Figure 5. Table cloud.

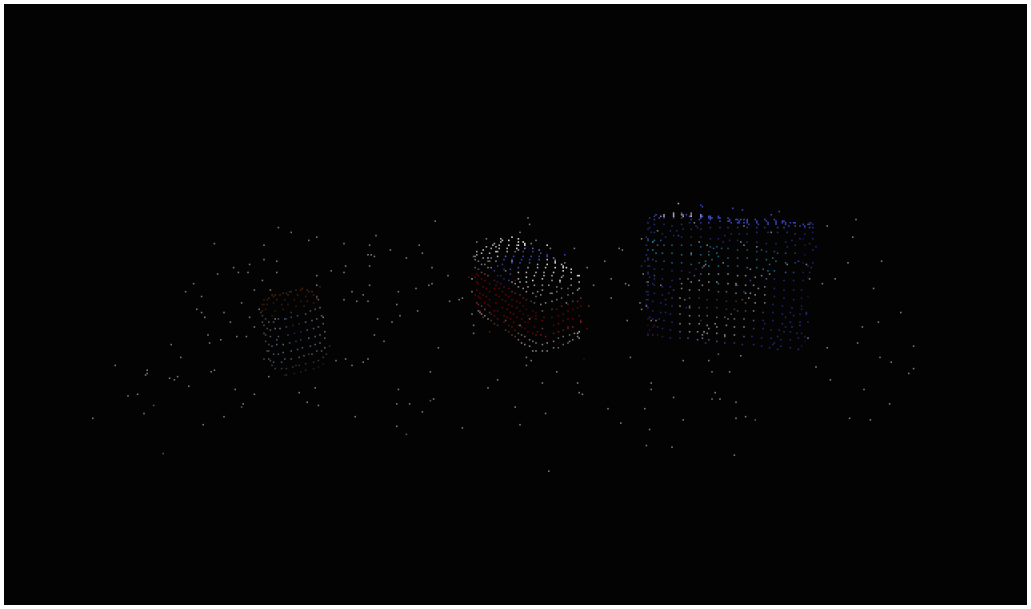

Figure 6. Objects cloud.

2. Segmentation

After filtering the original point cloud, I got the objects of interest with minimum noise. Now I can separate them using Euclidian Clustering. Here the parameters are: cluster tolerance = **0.01**, minimum cluster size = **100**, maximum cluster size = **5000**. These parameters were taken from the previous exercise but could be set in a more accurate way analyzing the individual point cloud for each object.

The point cloud of all the objects is shown in figure 7.

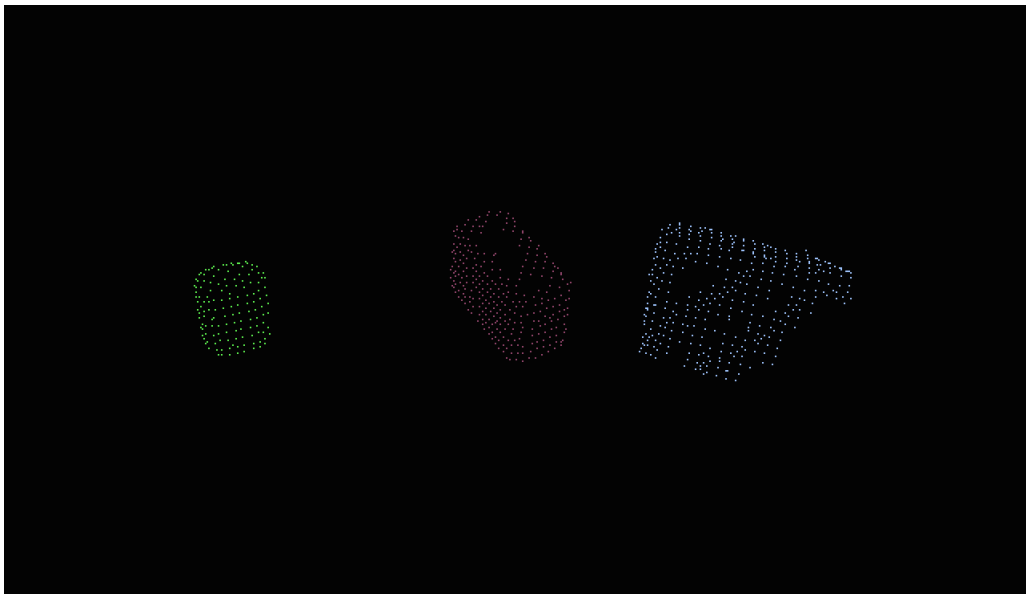

Figure 7. The three segmented objects.

Later in the code, the table, objects and cluster point clouds were converted to ROS messages and published to see them in RViz.

3. Cluster classification

The classification was done using SVM, but before to make a prediction, it is necessary to get the features of the objects and train the SVM.

In *capture_features.py* I had to modify the models list to include all the objects:

```
models = ['biscuits', 'soap', 'soap2', 'book', 'glue', 'sticky_notes', 'snacks', 'eraser']
```

I took 50 images for each objects and then the color and normal histograms were calculated using 32 bins for each channel. I used HSV color space. I think I should use more images and less bins and test if there is an improvement.

After getting the training set, the SVM was trained using a linear kernel and default parameters. The resulting normalized confusion matrix is shown in figure 8. I tried with rbf kernel which had worse performance.

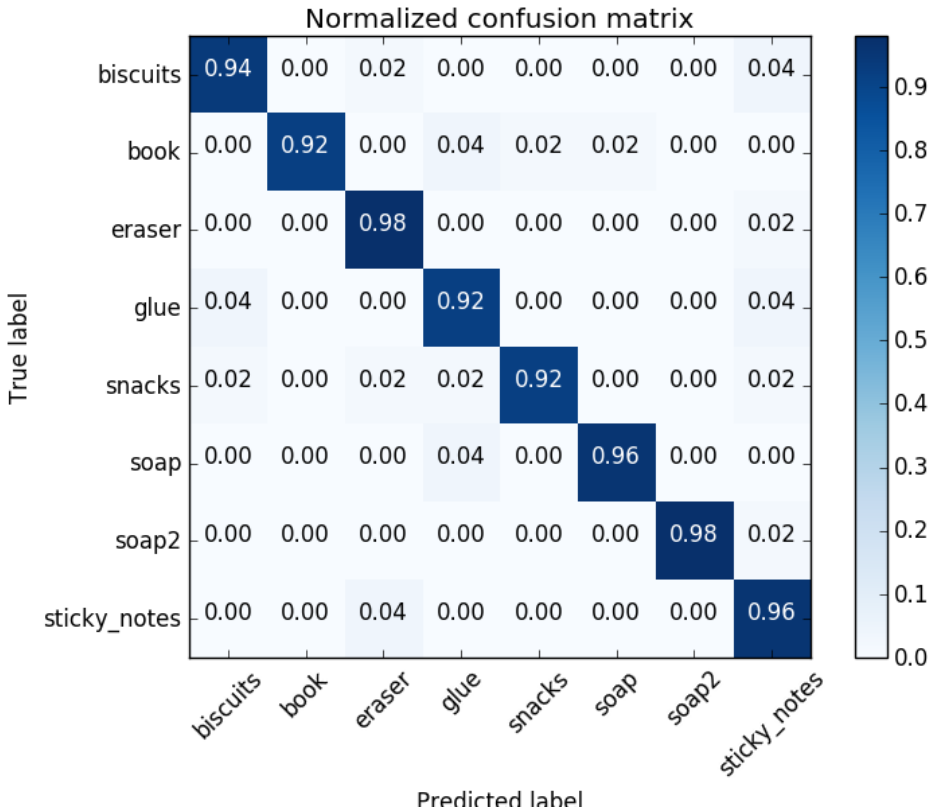

Figure 8. Normalized confusion matrix.

Being trained the SVM, I can use the model, which was saved in *model.sav*. But before it is necessary to get the feature vector of the object as:

```
# Compute the associated feature vector
chists = compute_color_histograms(ros_cluster, using_hsv=True)
normals = get_normals(ros_cluster)
nhists = compute_normal_histograms(normals)
feature = np.concatenate((chists, nhists))
```

and then make the prediction and put the name of the objects (labels) in a list:

```
# Make the prediction
prediction = clf.predict(scaler.transform(feature.reshape(1, -1)))
label = encoder.inverse_transform(prediction)[0]
detected_objects_labels.append(label)
```

Afterwards the labels are published and we can see them in RViz:

Figure 9. Object recognition in RViz for world 1.

Figure 10. Object recognition in RViz for world 2.

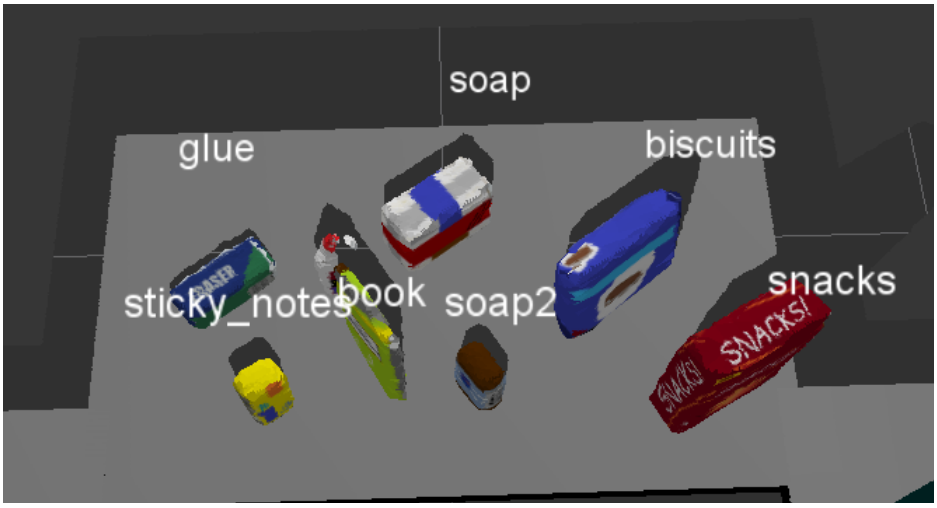

Figure 11. Object recognition in RViz for world 3.

In the world 1, I was able to detect 3 of 3 objects.

In the world 2, I was able to detect 4 of 5 objects. Although the book's color is different to the snacks color, the shape is similar, so that is affecting the recognition.

In the world 3, I was able to detect 6 of 8 objects. In this test, the book was recognized successfully because it has another orientation. The eraser's shape is similar to the glue's shape, so the system can't recognize it well. The glue is behind the book and it isn't recognized as a different object.

4. Parameters for PickPlace service

At the end of *pcl_callback()* we call the *pr2_mover()* function, in which we set all the parameters for the PickPlace service and write them to a .yaml file. Some of the parameters like the list of objects to pick and the dropbox positions were read from the ros parameter server. Then I loop through the pick list and check if the object we want to pick is in the list of detected objects. If that is true, we calculate the centroid and fill the pick pose parameters. In this project, the orientation parameters were set to 0. Regarding the object selection, if we had detected and object multiple times, the arm will pick the first in the list, not having into account which of them is more likely the object we want to pick, so this can be improved.

The arm name was set reading the group parameter for each object in the pick list, the left arm for the red group and the right arm for the green group. Also the group defines the dropbox parameters for each object.

After that I created the list of dictionaries and then the out.yaml file with all the parameters.