3D Perception Writeup

Writeup/ README

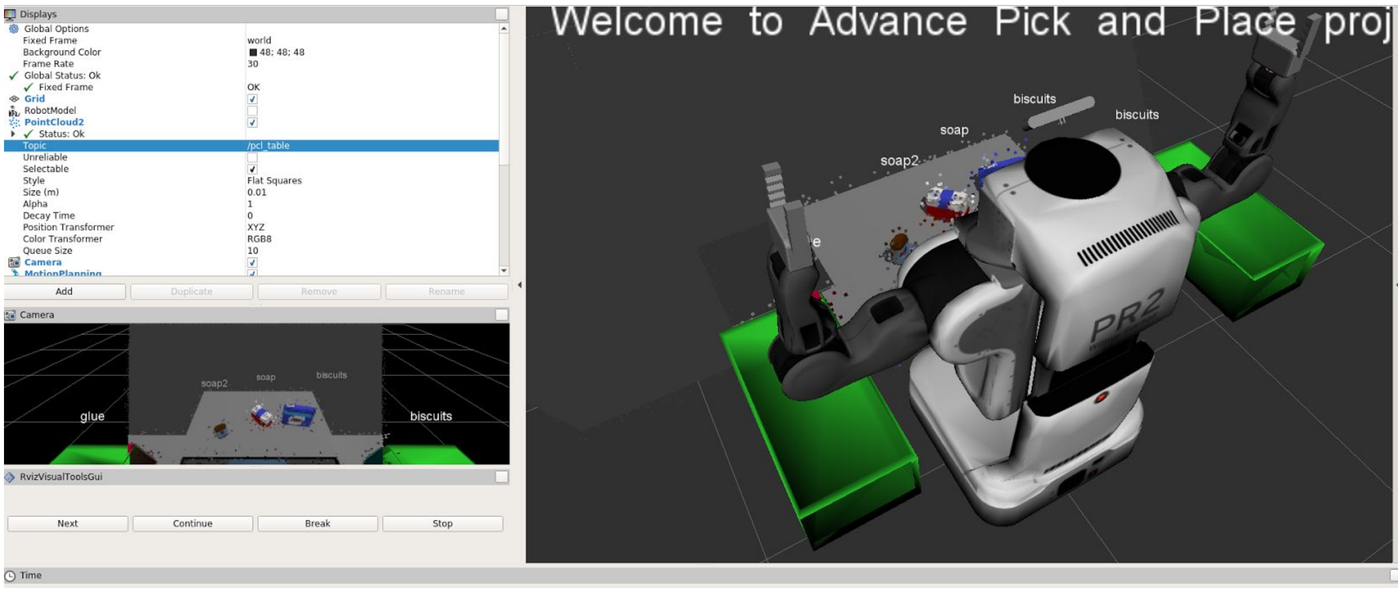
1. ***Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.***

This document is the writeup for the 3D Perception project.

Exercise 1, 2 and 3 Pipeline Implemented

1. ***Complete Exercise 1 steps. Pipeline for filtering and RANSAC plane fitting implemented.***

In this project, it was required to implement the whole perception pipeline to identify correctly the objects that need pick-up from a pick-up list. Taking a look at the point cloud detected directly from the camera, it can be seen that it needs some work on the point cloud itself:

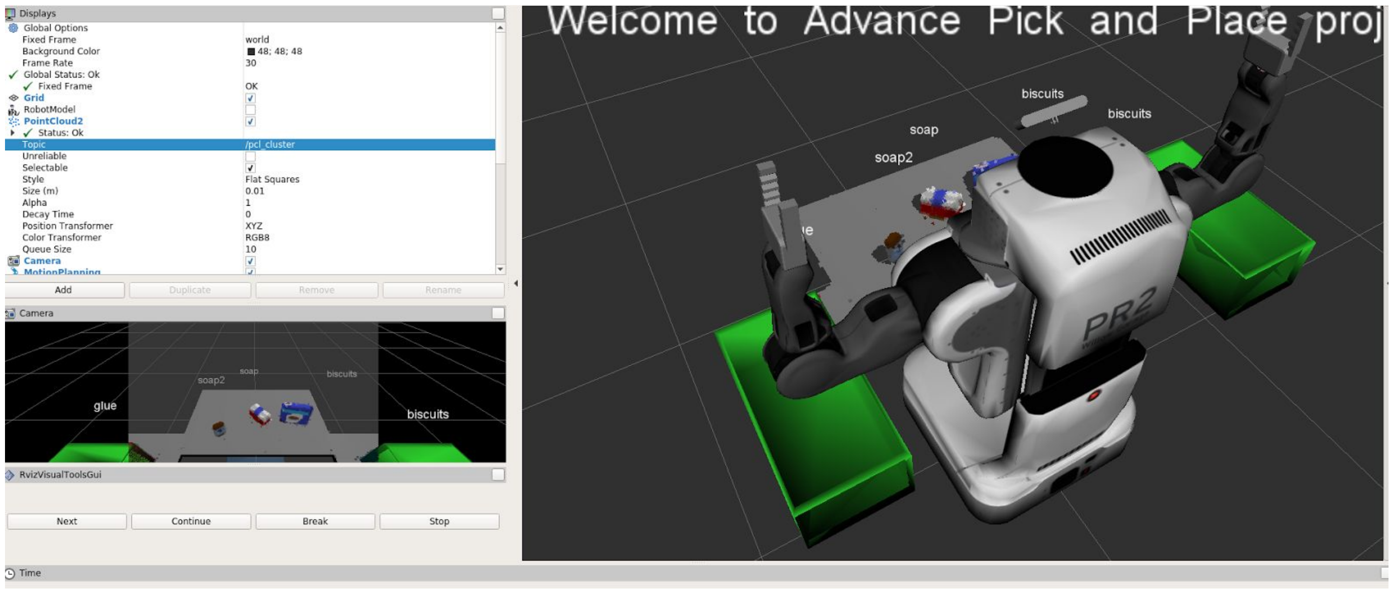


Displays	
Global Options	
Fixed Frame	world
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	
PointCloud2	✓
✓ Status: Ok	
Topic	/pcl_table
Unreliable	
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
Queue Size	10
Camera	✓
MotionPlanning	✓

The first step, before doing any kind of object recognition, is to filter this points cloud. By this, it means that it is necessary to remove any part of the scene that is not important for the object recognition, as well remove outliers. The following code down-samples, crops and removes the outliers of the pointcloud that it is being worked upon:

```
# TODO: Convert ROS msg to PCL data
cloud = ros_to_pcl(pcl_msg)

# TODO: Voxel Grid Downsampling
vox = cloud.make_voxel_grid_filter()
LEAF_SIZE = 0.01

# Set the voxel (or leaf) size
vox.set_leaf_size(LEAF_SIZE, LEAF_SIZE, LEAF_SIZE)
cloud_filtered = vox.filter()
# TODO: PassThrough Filter
passthrough = cloud_filtered.make_passthrough_filter()

# Assign axis and range to the passthrough filter object.
filter_axis = 'z'
passthrough.set_filter_field_name(filter_axis)
axis_min = 0.6
axis_max = 1.1
passthrough.set_filter_limits(axis_min, axis_max)

# Finally use the filter function to obtain the resultant point cloud.
cloud_filtered = passthrough.filter()

# Much like the previous filters, we start by creating a filter object:
outlier_filter = cloud_filtered.make_statistical_outlier_filter()

# Set the number of neighboring points to analyze for any given point
outlier_filter.set_mean_k(50)

# Set threshold scale factor
x = 0.5

# Any point with a mean distance larger than global (mean distance+x*std_dev) will be considered outlier
outlier_filter.set_std_dev_mul_thresh(x)

# Finally call the filter function for magic
cloud_filtered = outlier_filter.filter()
```

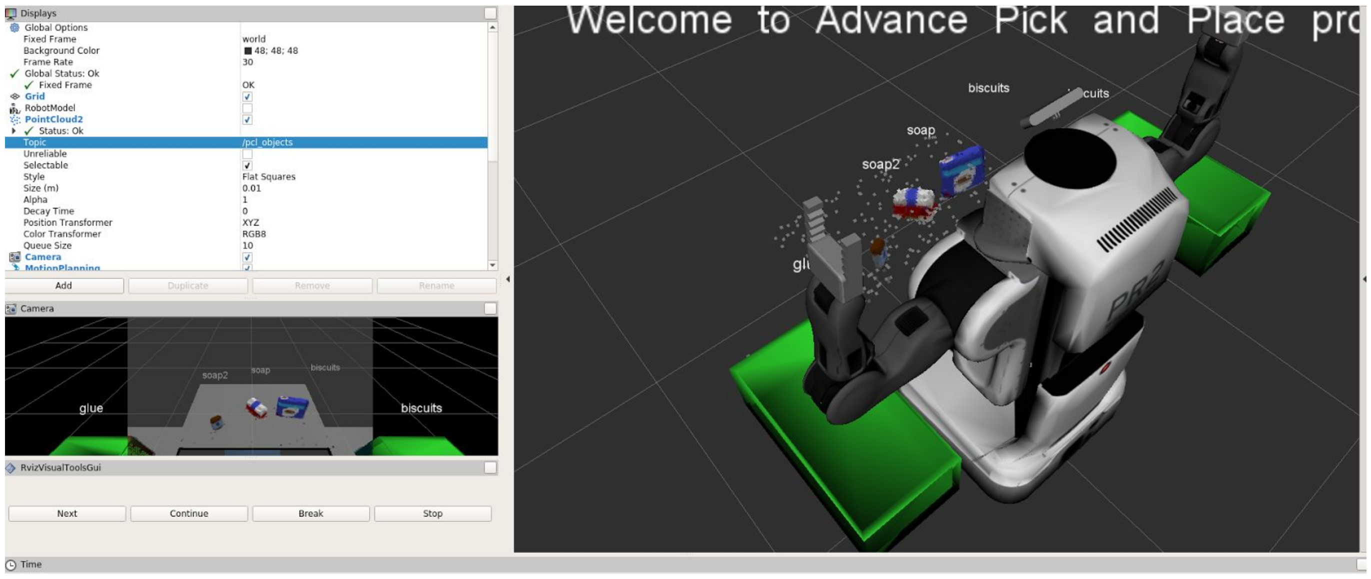
Resulting in the following point cloud:


```
Displays
Global Options
  Fixed Frame            world
  Background Color       48; 48; 48
  Frame Rate             30
✓ Global Status: Ok
  ✓ Fixed Frame          OK
Grid                     ✓
RobotModel
PointCloud2              ✓
  ✓ Status: Ok
  Topic                  /pcl_cluster
  Unreliable
  Selectable             ✓
  Style                  Flat Squares
  Size (m)               0.01
  Alpha                  1
  Decay Time             0
  Position Transformer   XYZ
  Color Transformer      RGB8
  Queue Size             10
Camera                   ✓
MotionPlanning           ✓
```

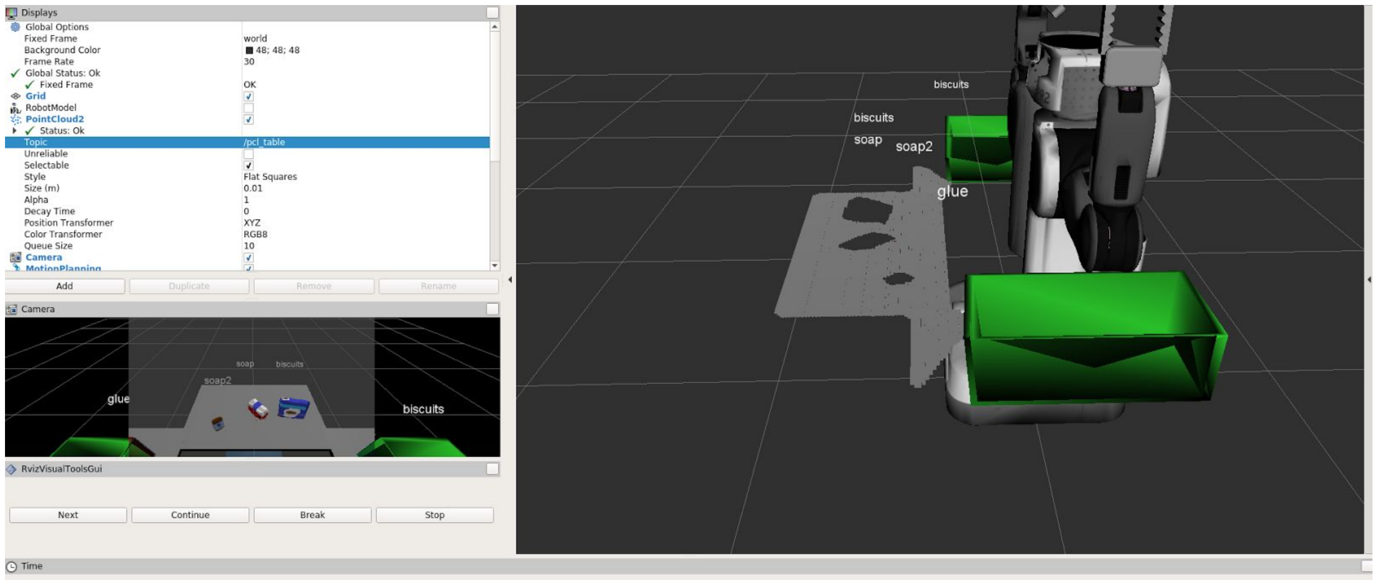
After taking care of the pointcloud, it is possible to proceed to the steps necessary for object recognition. The next step is the implementation of segmentation for the filtered pointcloud. The following code implements RANSAC Plane Segmentation where it extracts the inliers and outliers of the pointcloud:

```
# TODO: RANSAC Plane Segmentation
seg = cloud_filtered.make_segmenter()

# Set the model you wish to fit
seg.set_model_type(pcl.SACMODEL_PLANE)
seg.set_method_type(pcl.SAC_RANSAC)

# Max distance for a point to be considered fitting the model
# Experiment with different values for max_distance
# for segmenting the table
max_distance = 0.001
seg.set_distance_threshold(max_distance)

# Call the segment function to obtain set of inlier indices and model coefficients
inliers, coefficients = seg.segment()
# TODO: Extract inliers and outliers
cloud_table = cloud_filtered.extract(inliers, negative=False)
cloud_objects = cloud_filtered.extract(inliers, negative=True)
```

This, results in having only the objects of interest and outliers:



Global Options
Fixed Frame world
Background Color ■ 48; 48; 48
Frame Rate 30
✓ Global Status: Ok
✓ Fixed Frame OK
Grid ✓
RobotModel ☐
PointCloud2 ✓
✓ Status: Ok
Topic /pcl_objects
Unreliable ☐
Selectable ✓
Style Flat Squares
Size (m) 0.01
Alpha 1
Decay Time 0
Position Transformer XYZ
Color Transformer RGB8
Queue Size 10
Camera ✓
MotionPlanning ✓



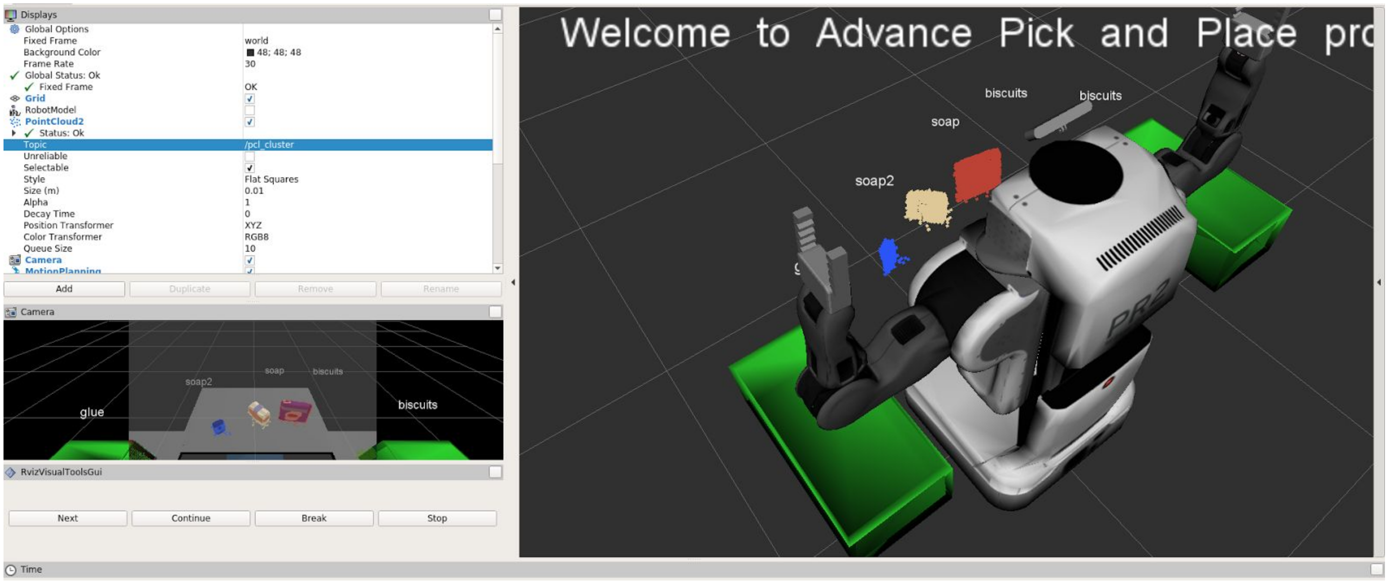
Global Options
Fixed Frame world
Background Color ■ 48; 48; 48
Frame Rate 30
✓ Global Status: Ok
✓ Fixed Frame OK
Grid ✓
RobotModel ✓
PointCloud2 ✓
✓ Status: Ok
Topic /pcl_table
Unreliable ☐
Selectable ✓
Style Flat Squares
Size (m) 0.01
Alpha 1
Decay Time 0
Position Transformer XYZ
Color Transformer RGB8
Queue Size 10
Camera ✓
MotionPlanning ✓

2. ***Complete Exercise 2 steps: Pipeline including clustering for segmentation implemented.***

From the results obtained in step 1), now it is necessary to cluster the objects of interest, to be, later on, classified. The clustering is done with Euclidean Clustering algorithm and the following code implements it:

```python
# TODO: Euclidean Clustering
white_cloud = XYZRGB_to_XYZ(cloud_objects)
tree = white_cloud.make_kdtree()
# Create a cluster extraction object
ec = white_cloud.make_EuclideanClusterExtraction()
# Set tolerances for distance threshold
# as well as minimum and maximum cluster size (in points)
ec.set_ClusterTolerance(0.025)
ec.set_MinClusterSize(30)
ec.set_MaxClusterSize(1200)
# Search the k-d tree for clusters
ec.set_SearchMethod(tree)
# Extract indices for each of the discovered clusters
cluster_indices = ec.Extract()
# TODO: Create Cluster-Mask Point Cloud to visualize each cluster separately
#Assign a color corresponding to each segmented object in scene
cluster_color = get_color_list(len(cluster_indices))

color_cluster_point_list = []

for j, indices in enumerate(cluster_indices):
    for i, indice in enumerate(indices):
        color_cluster_point_list.append([white_cloud[indice][0],
                                         white_cloud[indice][1],
                                         white_cloud[indice][2],
                                          rgb_to_float(cluster_color[j])])

#Create new cloud containing all clusters, each with unique color
cluster_cloud = pcl.PointCloud_PointXYZRGB()
cluster_cloud.from_list(color_cluster_point_list)
```

Resulting in the following clusters:

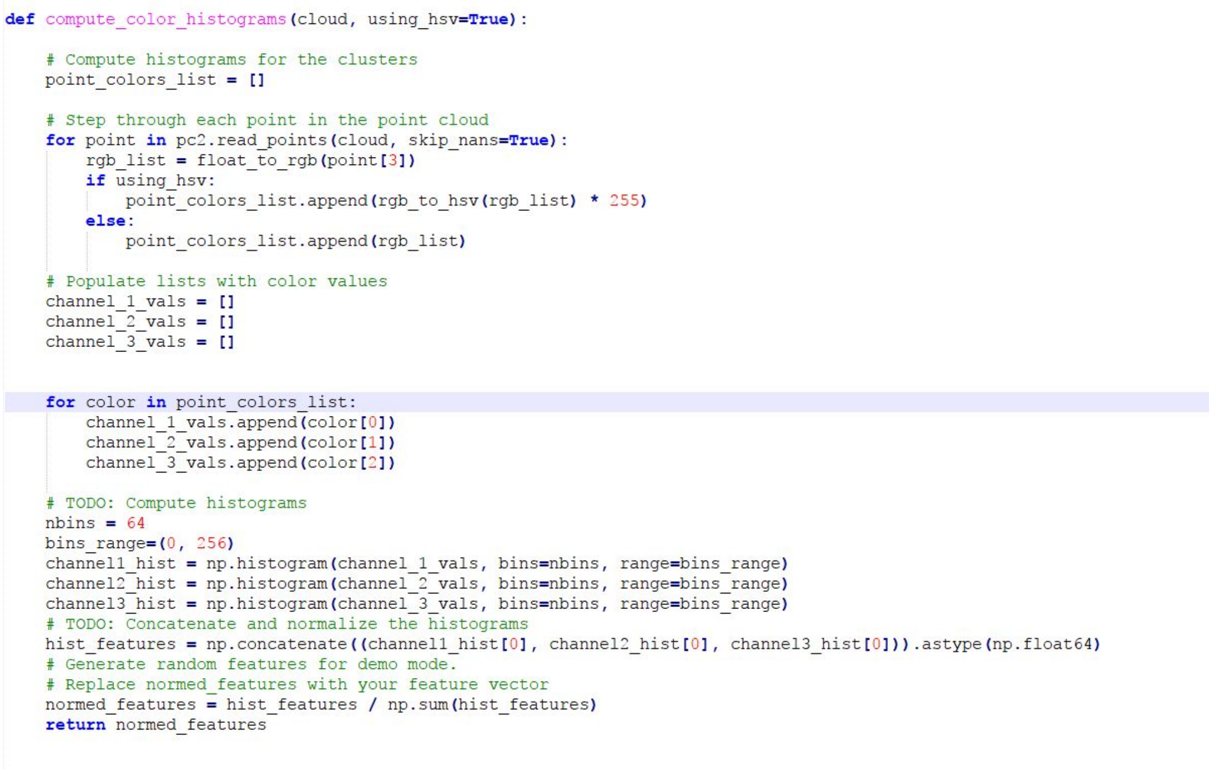


```
Global Options
  Fixed Frame              world
  Background Color         ■ 48; 48; 48
  Frame Rate               30
✓ Global Status: Ok
  ✓ Fixed Frame            OK
Grid                       ✓
RobotModel                 ☐
PointCloud2                ✓
  ✓ Status: Ok
  Topic                    /pcl_cluster
  Unreliable               ☐
  Selectable               ✓
  Style                    Flat Squares
  Size (m)                 0.01
  Alpha                    1
  Decay Time               0
  Position Transformer     XYZ
  Color Transformer        RGB8
  Queue Size               10
Camera                     ✓
MotionPlanning             ✓
```

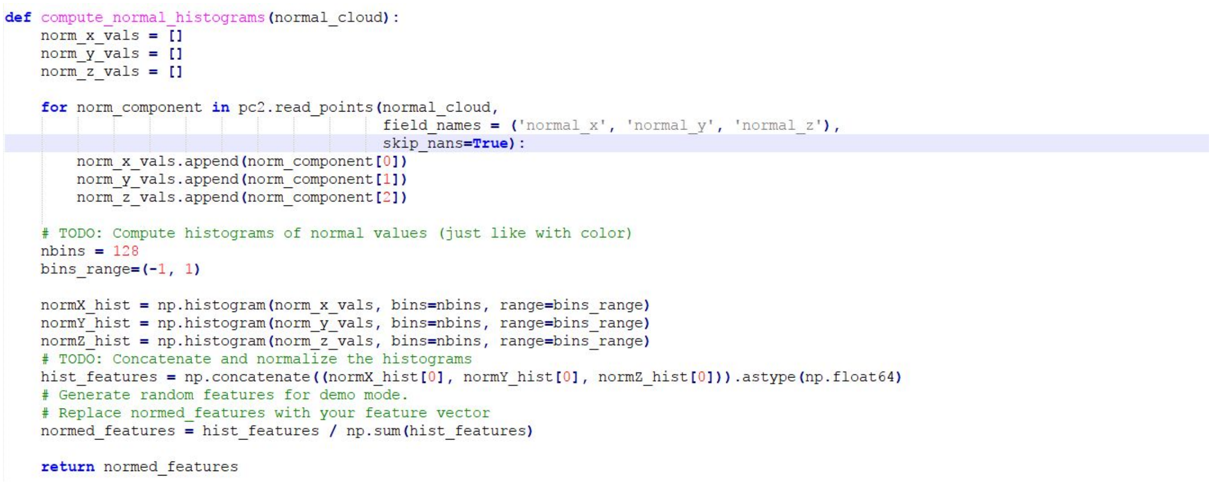
The 3 different coloured clusters mean that 3 different objects were found in the pointcloud. With this, the following step is to classify these clusters.

3. ***Complete Exercise 3 Steps. Features extracted and SVM trained. Object recognition implemented.***

Before moving on to the implementation of the classification, it is necessary to train the classifier with a dataset of pickup items generated at random orientations and many times. The following functions computed the color histograms and normal histograms, respectively:

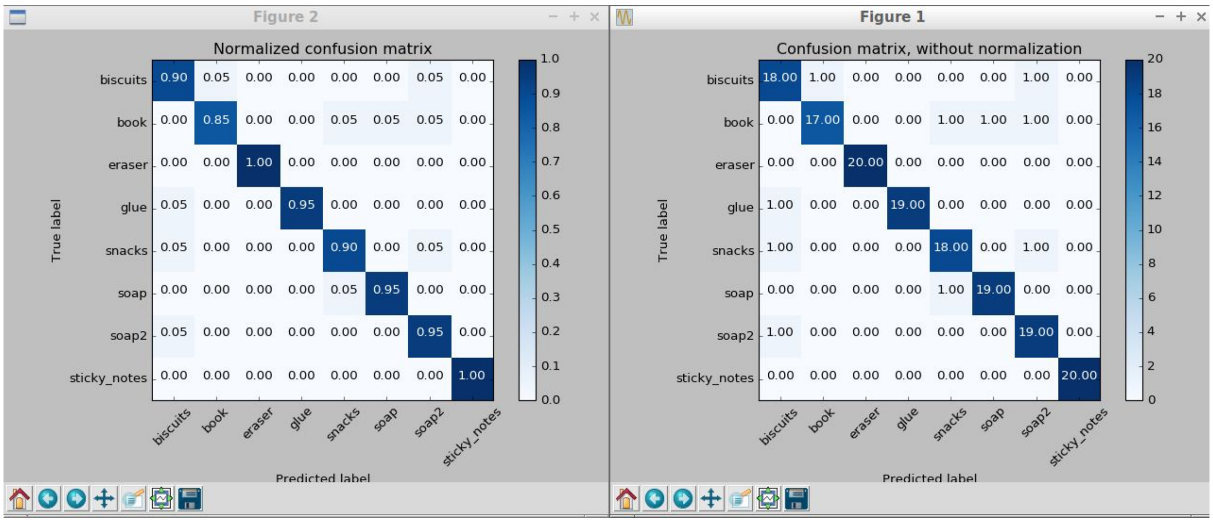


```
def compute_color_histograms(cloud, using_hsv=True):

    # Compute histograms for the clusters
    point_colors_list = []

    # Step through each point in the point cloud
    for point in pc2.read_points(cloud, skip_nans=True):
        rgb_list = float_to_rgb(point[3])
        if using_hsv:
            point_colors_list.append(rgb_to_hsv(rgb_list) * 255)
        else:
            point_colors_list.append(rgb_list)

    # Populate lists with color values
    channel_1_vals = []
    channel_2_vals = []
    channel_3_vals = []


    for color in point_colors_list:
        channel_1_vals.append(color[0])
        channel_2_vals.append(color[1])
        channel_3_vals.append(color[2])

    # TODO: Compute histograms
    nbins = 64
    bins_range=(0, 256)
    channel1_hist = np.histogram(channel_1_vals, bins=nbins, range=bins_range)
    channel2_hist = np.histogram(channel_2_vals, bins=nbins, range=bins_range)
    channel3_hist = np.histogram(channel_3_vals, bins=nbins, range=bins_range)
    # TODO: Concatenate and normalize the histograms
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0])).astype(np.float64)
    # Generate random features for demo mode.
    # Replace normed_features with your feature vector
    normed_features = hist_features / np.sum(hist_features)
    return normed_features
```



```
def compute_normal_histograms(normal_cloud):
    norm_x_vals = []
    norm_y_vals = []
    norm_z_vals = []

    for norm_component in pc2.read_points(normal_cloud,
                                          field_names = ('normal_x', 'normal_y', 'normal_z'),
                                          skip_nans=True):
        norm_x_vals.append(norm_component[0])
        norm_y_vals.append(norm_component[1])
        norm_z_vals.append(norm_component[2])

    # TODO: Compute histograms of normal values (just like with color)
    nbins = 128
    bins_range=(-1, 1)

    normX_hist = np.histogram(norm_x_vals, bins=nbins, range=bins_range)
    normY_hist = np.histogram(norm_y_vals, bins=nbins, range=bins_range)
    normZ_hist = np.histogram(norm_z_vals, bins=nbins, range=bins_range)
    # TODO: Concatenate and normalize the histograms
    hist_features = np.concatenate((normX_hist[0], normY_hist[0], normZ_hist[0])).astype(np.float64)
    # Generate random features for demo mode.
    # Replace normed_features with your feature vector
    normed_features = hist_features / np.sum(hist_features)

    return normed_features
```

After computing the information, it is sent to the SVM, creating the following confusion matrix and accuracy results:

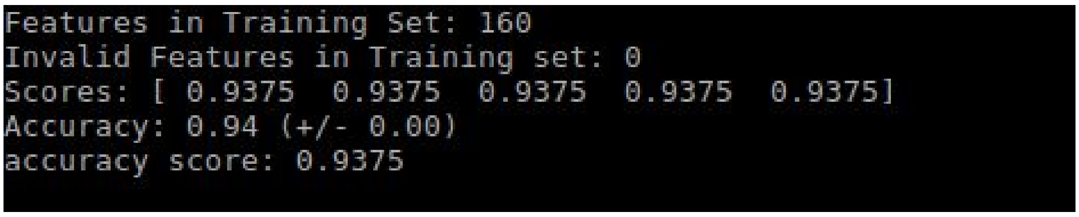


Normalized confusion matrix

	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	0.90	0.05	0.00	0.00	0.00	0.00	0.05	0.00
book	0.00	0.85	0.00	0.00	0.05	0.05	0.05	0.00
eraser	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
glue	0.05	0.00	0.00	0.95	0.00	0.00	0.00	0.00
snacks	0.05	0.00	0.00	0.00	0.90	0.00	0.05	0.00
soap	0.00	0.00	0.00	0.00	0.05	0.95	0.00	0.00
soap2	0.05	0.00	0.00	0.00	0.00	0.00	0.95	0.00
sticky_notes	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

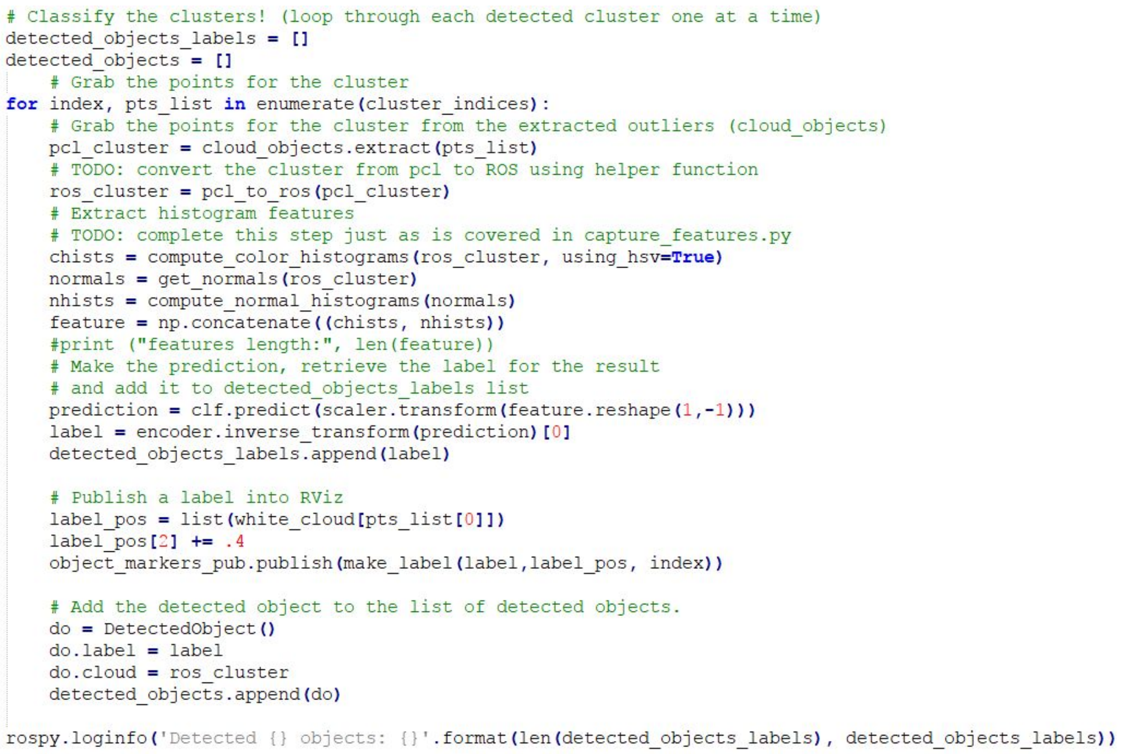
Confusion matrix, without normalization

	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	18.00	1.00	0.00	0.00	0.00	0.00	1.00	0.00
book	0.00	17.00	0.00	0.00	1.00	1.00	1.00	0.00
eraser	0.00	0.00	20.00	0.00	0.00	0.00	0.00	0.00
glue	1.00	0.00	0.00	19.00	0.00	0.00	0.00	0.00
snacks	1.00	0.00	0.00	0.00	18.00	0.00	1.00	0.00
soap	0.00	0.00	0.00	0.00	1.00	19.00	0.00	0.00
soap2	1.00	0.00	0.00	0.00	0.00	0.00	19.00	0.00
sticky_notes	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00



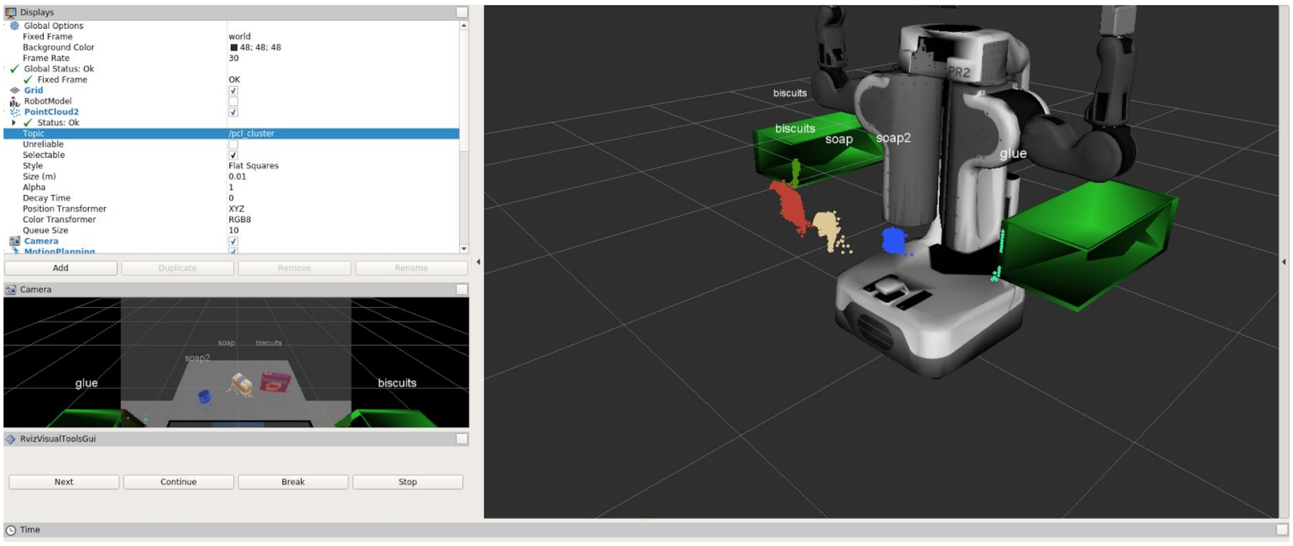
```
Features in Training Set: 160
Invalid Features in Training set: 0
Scores: [ 0.9375  0.9375  0.9375  0.9375  0.9375]
Accuracy: 0.94 (+/- 0.00)
accuracy score: 0.9375
```

The results presented in the previous images are good results, therefore, the object recognition can be implemented on the pcl_callback function. The code was implemented the following way:



```
# Classify the clusters! (loop through each detected cluster one at a time)
detected_objects_labels = []
detected_objects = []
    # Grab the points for the cluster
for index, pts_list in enumerate(cluster_indices):
    # Grab the points for the cluster from the extracted outliers (cloud_objects)
    pcl_cluster = cloud_objects.extract(pts_list)
    # TODO: convert the cluster from pcl to ROS using helper function
    ros_cluster = pcl_to_ros(pcl_cluster)
    # Extract histogram features
    # TODO: complete this step just as is covered in capture_features.py
    chists = compute_color_histograms(ros_cluster, using_hsv=True)
    normals = get_normals(ros_cluster)
    nhists = compute_normal_histograms(normals)
    feature = np.concatenate((chists, nhists))
    #print ("features length:", len(feature))
    # Make the prediction, retrieve the label for the result
    # and add it to detected_objects_labels list
    prediction = clf.predict(scaler.transform(feature.reshape(1,-1)))
    label = encoder.inverse_transform(prediction)[0]
    detected_objects_labels.append(label)

    # Publish a label into RViz
    label_pos = list(white_cloud[pts_list[0]])
    label_pos[2] += .4
    object_markers_pub.publish(make_label(label,label_pos, index))

    # Add the detected object to the list of detected objects.
    do = DetectedObject()
    do.label = label
    do.cloud = ros_cluster
    detected_objects.append(do)

rospy.loginfo('Detected {} objects: {}'.format(len(detected_objects_labels), detected_objects_labels))
```

Leading to the following representation of the point cloud:



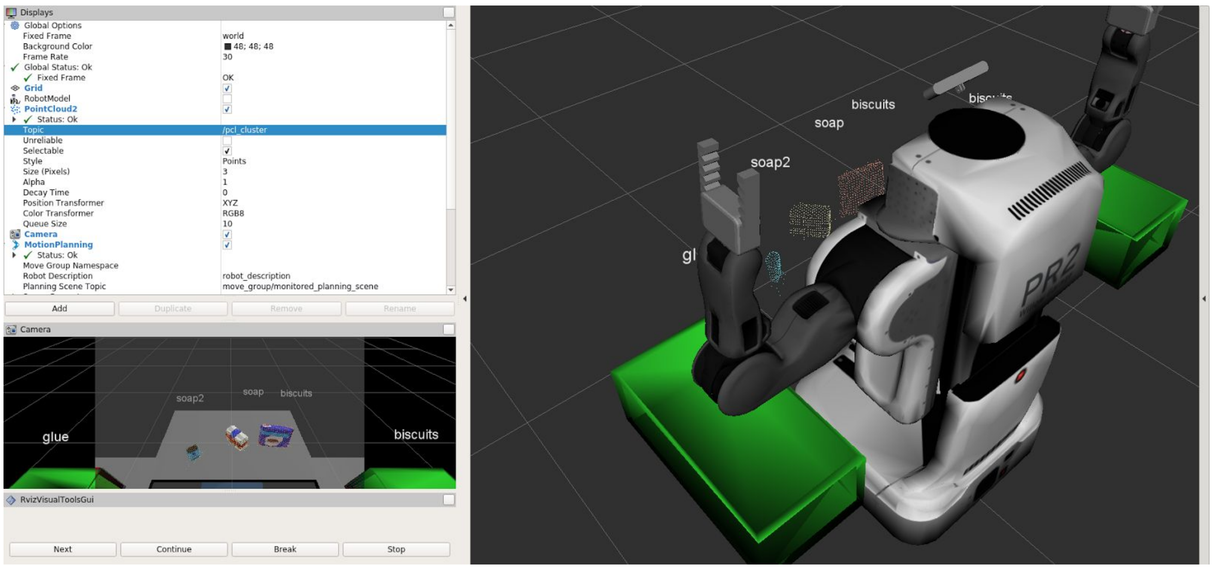
```
Displays
Global Options
  Fixed Frame                 world
  Background Color            ■ 48; 48; 48
  Frame Rate                  30
✓ Global Status: Ok
  ✓ Fixed Frame               OK
Grid                          ✓
RobotModel                    ☐
PointCloud2                   ✓
  ✓ Status: Ok
  Topic                       /pcl_cluster
  Unreliable                  ☐
  Selectable                  ✓
  Style                       Flat Squares
  Size (m)                    0.01
  Alpha                       1
  Decay Time                  0
  Position Transformer        XYZ
  Color Transformer           RGB8
  Queue Size                  10
Camera                        ✓
MotionPlanning                ✓
```

Where the clusters have been classified and labeled in the pointcloud.

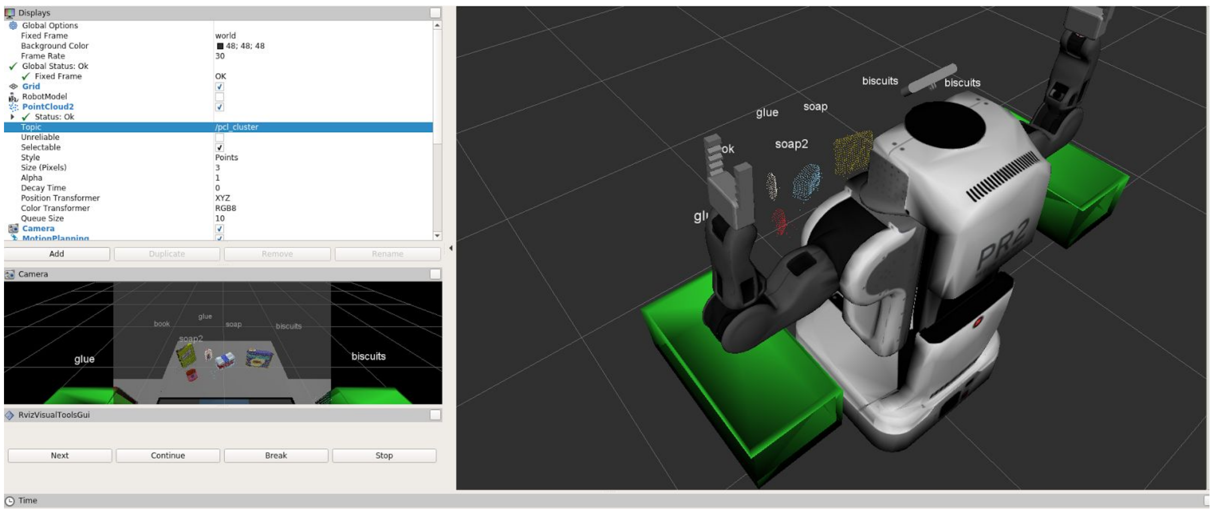
Pick and Place Setup

1. ***For all three tabletop setups (test*.world), perform object recognition, then read in respective pick list (pick_list_*.yaml). Next construct the messages that would comprise a valid PickPlace request output them to .yaml format.***

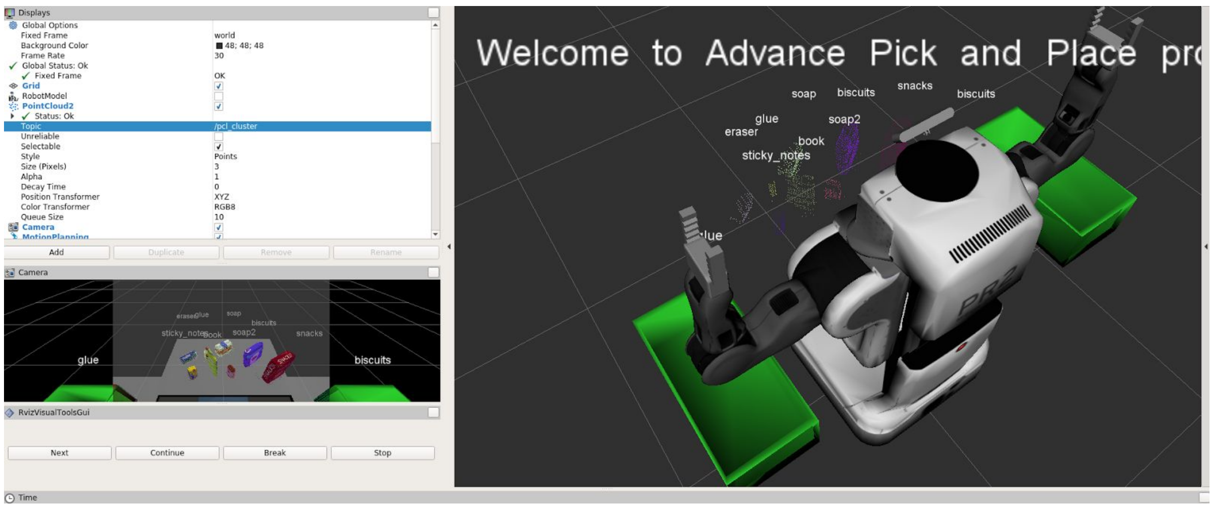
Now, by finishing the perception pipeline implementation, the ROS node was tested into the 3 different scenarios offered in this project. The resulting .yaml files are annexed to this report. The resulting images show the correct classification for world1, world2 and world3, respectively:



```
Displays
Global Options
  Fixed Frame                 world
  Background Color            ■ 48; 48; 48
  Frame Rate                  30
✓ Global Status: Ok
  ✓ Fixed Frame               OK
Grid                          ✓
RobotModel                    ☐
PointCloud2                   ✓
  ✓ Status: Ok
  Topic                       /pcl_cluster
  Unreliable                  ☐
  Selectable                  ✓
  Style                       Points
  Size (Pixels)               3
  Alpha                       1
  Decay Time                  0
  Position Transformer        XYZ
  Color Transformer           RGB8
  Queue Size                  10
Camera                        ✓
MotionPlanning                ✓
  ✓ Status: Ok
  Move Group Namespace
  Robot Description           robot_description
  Planning Scene Topic        move_group/monitored_planning_scene
```



```
Displays
Global Options
  Fixed Frame              world
  Background Color         48; 48; 48
  Frame Rate               30
Global Status: Ok
  Fixed Frame              OK
Grid                       ✓
RobotModel
PointCloud2                ✓
  Status: Ok
  Topic                    /pcl_cluster
  Unreliable
  Selectable               ✓
  Style                    Points
  Size (Pixels)            3
  Alpha                    1
  Decay Time               0
  Position Transformer     XYZ
  Color Transformer        RGB8
  Queue Size               10
Camera                     ✓
MotionPlanning
```



```
Displays
Global Options
  Fixed Frame              world
  Background Color         48; 48; 48
  Frame Rate               30
Global Status: Ok
  Fixed Frame              OK
Grid                       ✓
RobotModel
PointCloud2                ✓
  Status: Ok
  Topic                    /pcl_cluster
  Unreliable
  Selectable               ✓
  Style                    Points
  Size (Pixels)            3
  Alpha                    1
  Decay Time               0
  Position Transformer     XYZ
  Color Transformer        RGB8
  Queue Size               10
Camera                     ✓
MotionPlanning
```

As shown in these images, the classifier identified correctly all the objects presented in all scenes. The code for this perception pipeline is also annexed to this report.

Although the results are good, it can still be improved(it can be seen by the confusion matrix). To improve the results:

- Increase the number of random generating poses.
- Tweak the color and normal histograms bins.

There is no need to change the kernel, since Linear presented the best results so far.

Also, to truly have a finished code, the pick and place operation should be implemented(It will be in the future.). One last thing to add, is that detection objects list needs a bit of improvement, since it always adds 2 objects in the edges that should not appear(filtering the pointcloud scene can be better.).