3D Perception

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

1. Statistical Outlier Filtering
   First I implemented statistical outliner filter in order to get rid of the noise present in the environment.

```
# TODO: Convert ROS msg to PCL data

cloud=ros_to_pcl(pcl_msg)

# TODO: Statistical Outlier Filtering
# Much like the previous filters, we start by creating a filter object:
outlier_filter =(cloud).make_statistical_outlier_filter()

# Set the number of neighboring points to analyze for any given point
outlier_filter.set_mean_k(10)

# Set threshold scale factor
x = 0.001

# 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()
```

2. Voxel Grid Downsampling
   Next, I applied voxel grid downsampling in order to speed up the process as Voxel Grid Downsampling Filter help to derive a point cloud that has fewer points by taking a spatial average of the points in the cloud confined by each voxel. Here I set leaf_size to be 0.01.

```
# TODO: Voxel Grid Downsampling

vox = cloud_filtered.make_voxel_grid_filter()

# Choose a voxel (also known as leaf) size
LEAF_SIZE = 0.01

# Set the voxel (or leaf) size
vox.set_leaf_size(LEAF_SIZE, LEAF_SIZE, LEAF_SIZE)

# Call the filter function to obtain the resultant downsampled point cloud
vox_cloud = vox.filter()
```

3. Pass Through Filter

Next, I applied pass through the filter to remove useless data from the point cloud. In my project I applied 2 pass through filters, first one is along the z-axis with axis_min as 0.5 and axis_max as 0.85 and the second one along x-axis with axis_min as 0.4 and axis_max as 0.9.

```
# Create a PassThrough filter object.
passthrough = vox_cloud.make_passthrough_filter()

# Assign axis and range to the passthrough filter object.
filter_axis = 'z'
passthrough.set_filter_field_name(filter_axis)
axis_min = 0.5
axis_max = 0.85
passthrough.set_filter_limits(axis_min, axis_max)

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

# Assign axis and range to the passthrough filter object.
filter_axis = 'x'
passthrough.set_filter_field_name(filter_axis)
axis_min = 0.4
axis_max = 0.9
passthrough.set_filter_limits(axis_min, axis_max)

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

4. RANSAC Plane Segmentation

Applying the RANSAC plane fitting code gives us the separate point cloud of table and object. Now we extract indices of object and table separately, so we can focus on the objects and apply object recognition algorithm on it.

```
# TODO: RANSAC Plane Segmentation

seg = passthrough_cloud.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
# for segmenting the table
max_distance = 0.01
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

# Extract inliers
cloud_table = passthrough_cloud.extract(inliers, negative=False)

# Extract outliners
cloud_objects = passthrough_cloud.extract(inliers, negative=True)
```

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

Euclidean Clustering

After segmentation, we will perform Euclidean clustering using k-d tree to decrease the computational burden of searching for neighboring points. This will give us the cluster of points corresponding to each object. Then we perform cluster extraction and create a new point cloud to visualize the clusters by assigning a color to each of them. Then convert PCL message to ros message and publish ros message.

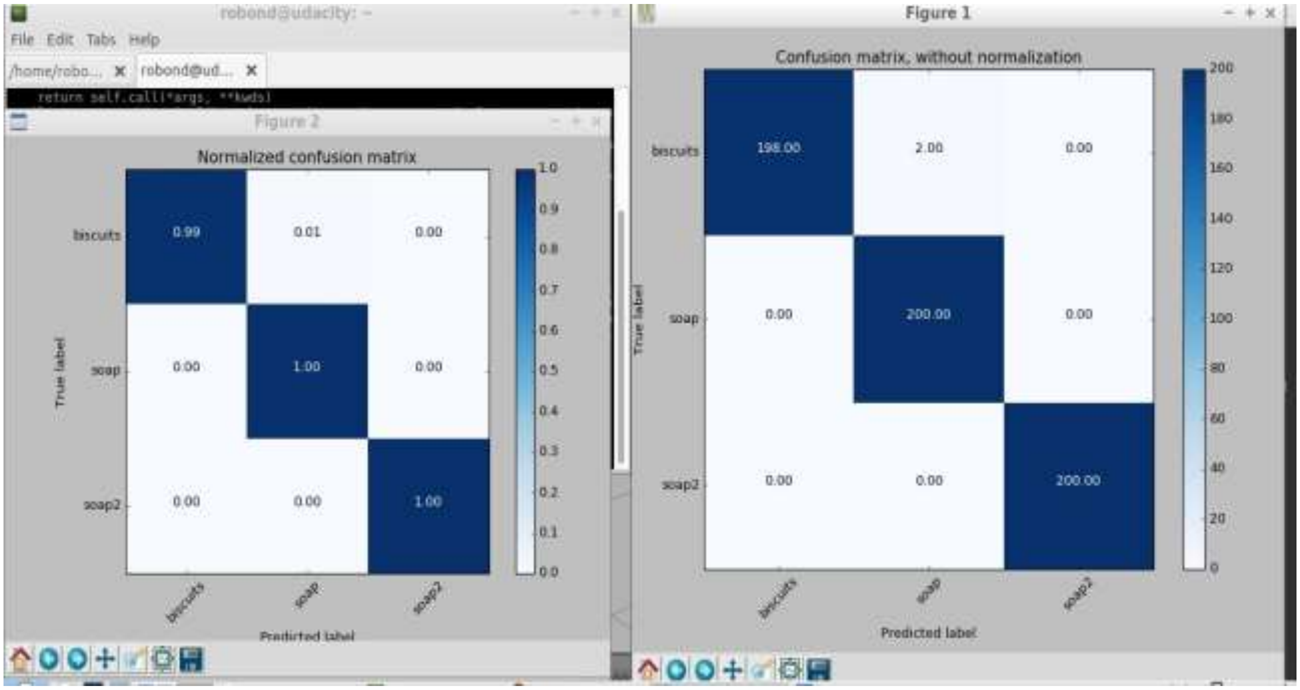
```
# TODO: Euclidean Clustering

white_cloud = XYZRGB_to_XYZ(cloud_objects) # Apply function to convert XYZRGB to XYZ
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(0.05)
ec.set_MinClusterSize(50)
ec.set_MaxClusterSize(200000)
# 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

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)
```

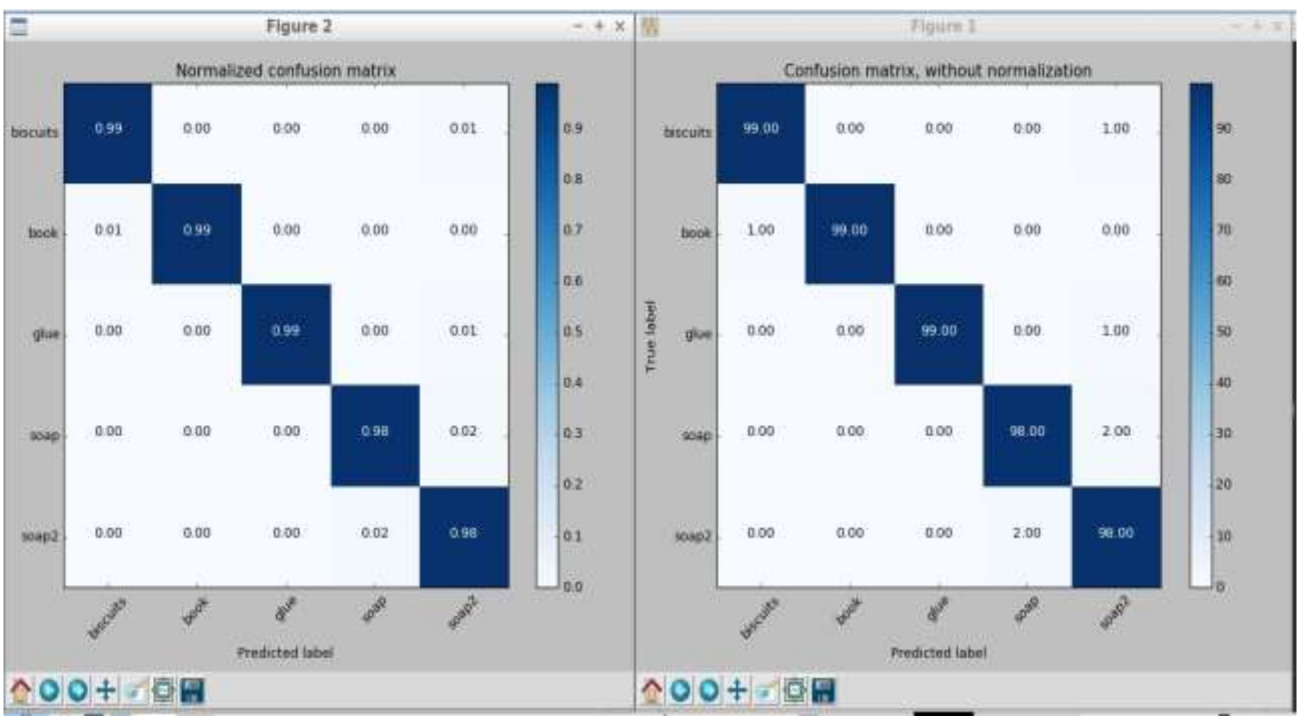
```
# TODO: Convert PCL data to ROS messages

ros_cloud_objects = pcl_to_ros(cloud_objects)
ros_cloud_table = pcl_to_ros(cloud_table)
ros_cluster_cloud = pcl_to_ros(cluster_cloud)

# TODO: Publish ROS messages

pcl_objects_pub.publish(ros_cloud_objects)
pcl_table_pub.publish(ros_cloud_table)
pcl_cluster_pub.publish(ros_cluster_cloud)
```

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

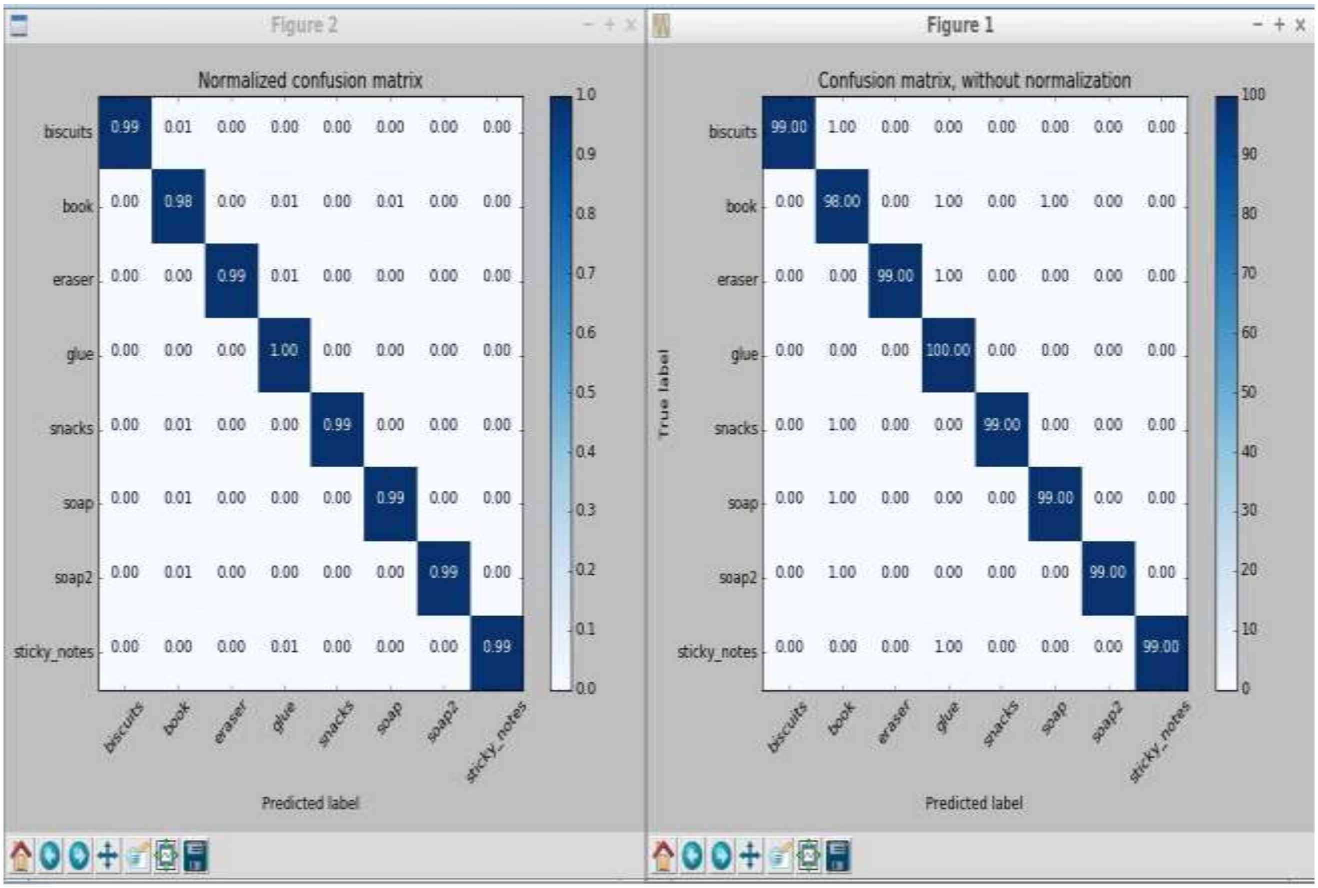
Now to perform object recognition, I first filled out the compute_color_histograms() and compute_normal_histograms() functions in features.py . 100 orientations were used to train the model. The models are trained using SVM using a Linear Kernel. The confusion matrix for all three world are shown below-



World 1



World 2



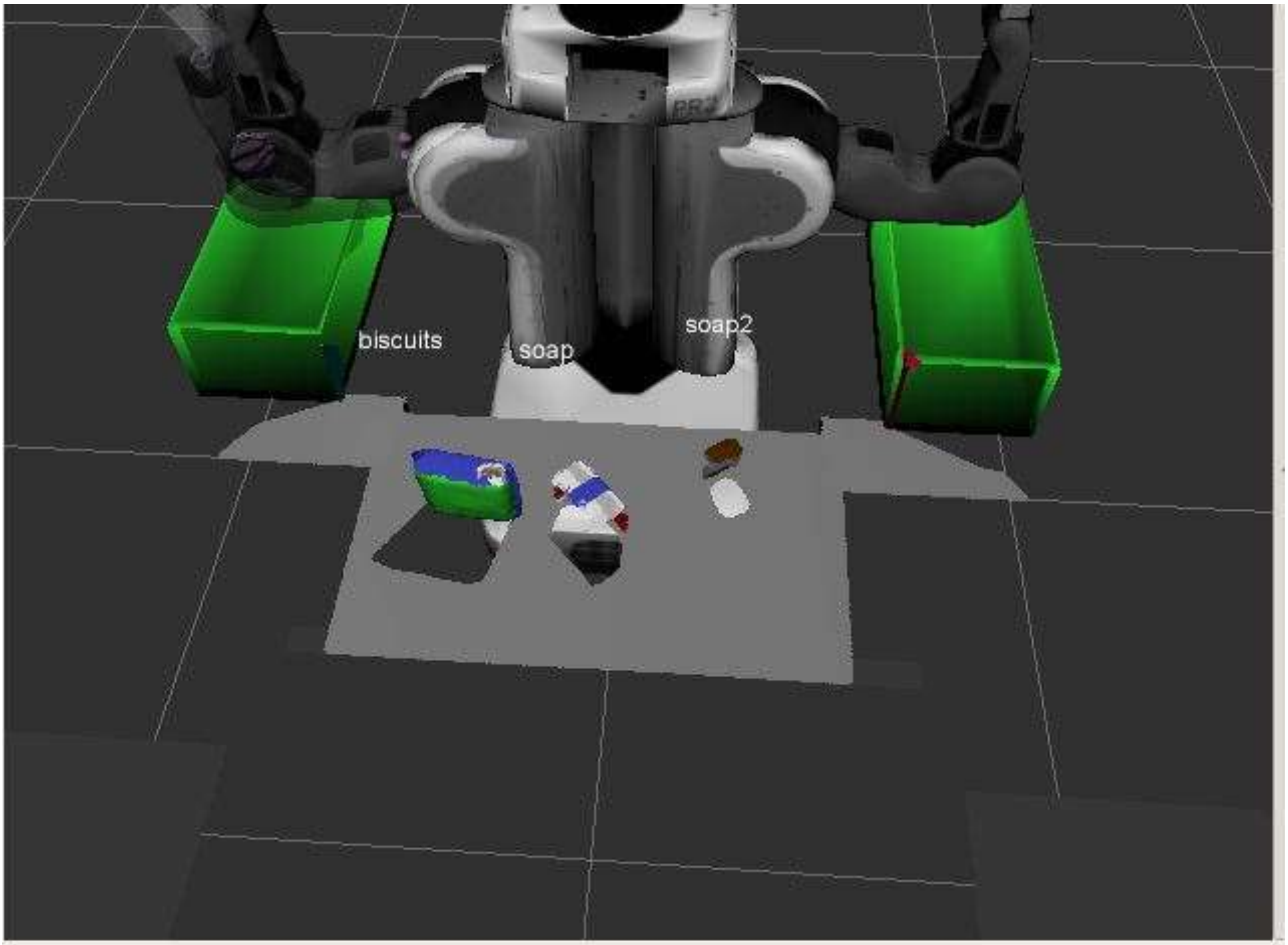
World 3

In the perception pipeline, the histogram features are computed for each object, and prediction is done using the trained SVM model and then add it to detected_objects_labels list.Then we add the detected object to the list of detected objects and publish it.

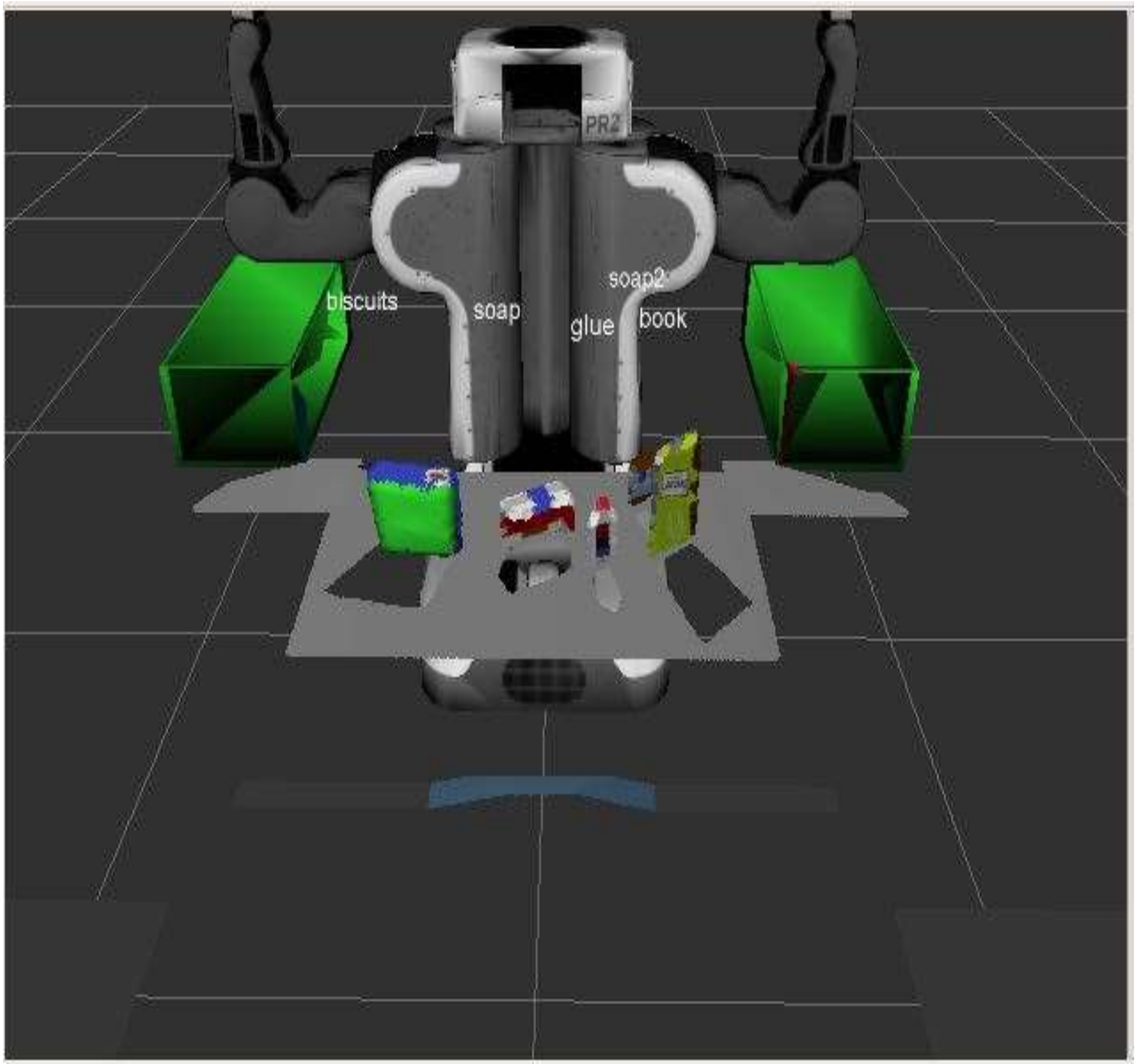
```python
# Classify the clusters! (loop through each detected cluster one at a time)
detected_objects_labels = []
detected_objects = []

for index, pts_list in enumerate(cluster_indices):
    # Grab the points for the cluster
    pcl_cluster = cloud_objects.extract(pts_list)
    # convert the cluster from pcl to ROS using helper function
    ros_cluster = pcl_to_ros(pcl_cluster)
    # Compute the associated feature vector

    # Extract histogram features
    # 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))

    # Make the prediction
    # 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))

# Publish the list of detected objects
detected_objects_pub.publish(detected_objects)
```

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.

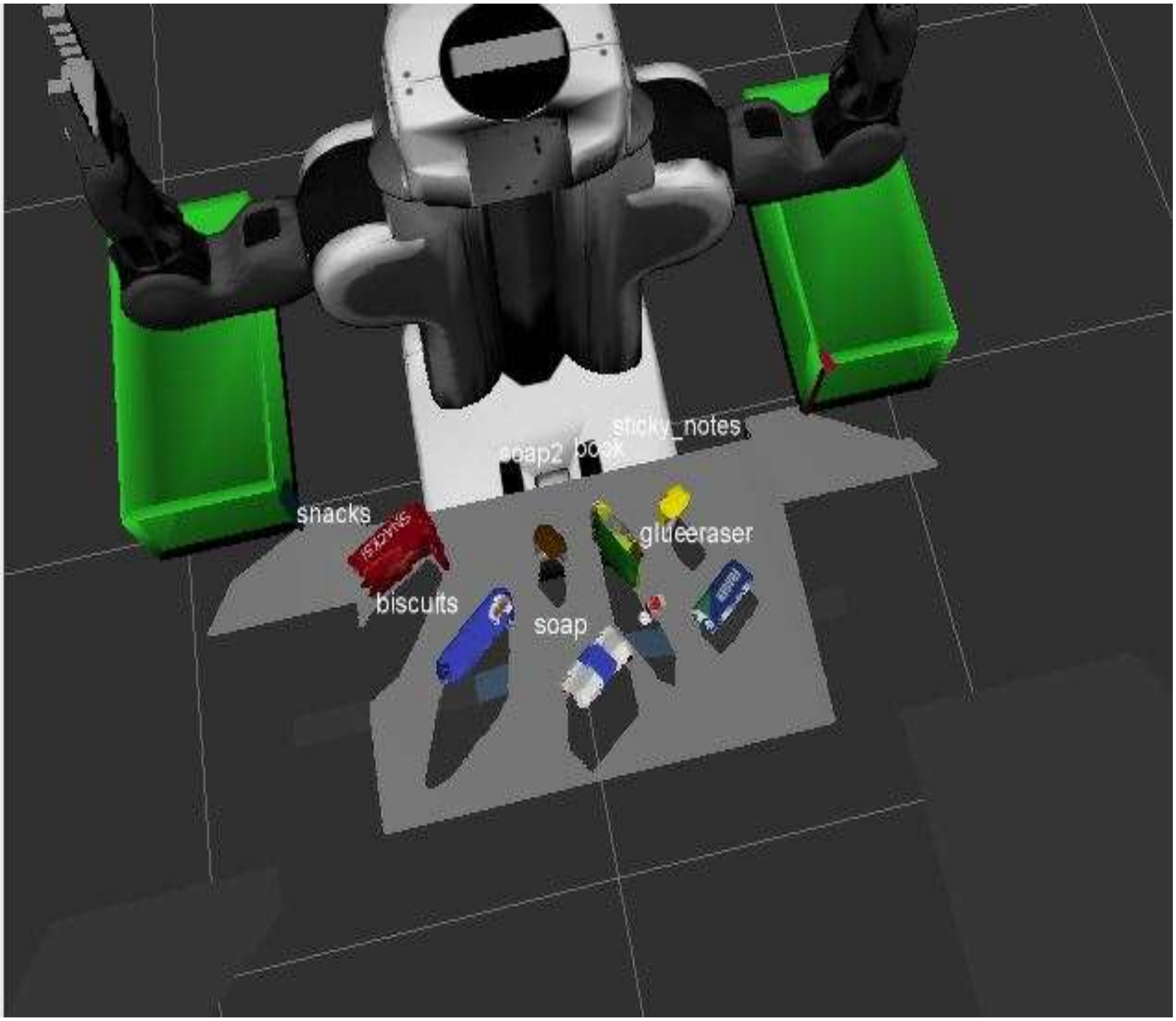
Performing all the above perception pipeline exercises and reading the pick_list_. Yaml, I obtained the following results in the three worlds.After Pick and Place is performed, messages are constructed that would comprise a valid pick n place request and output were recorded in .yaml format(output_1.yaml, output_2.yaml, output_3.yaml) which can be found in the output .yaml folder.



World 1



World 2



World 3

Conclusion

The perception project is successfully completed. But it was observed that as the number of objects to be recognized increases the used parameters doesn't give a 100% results. So I need to further improve the learning model by collecting more data. Further, the robot's pick and place operation can be improved by adding code dealing with the collision scenarios.