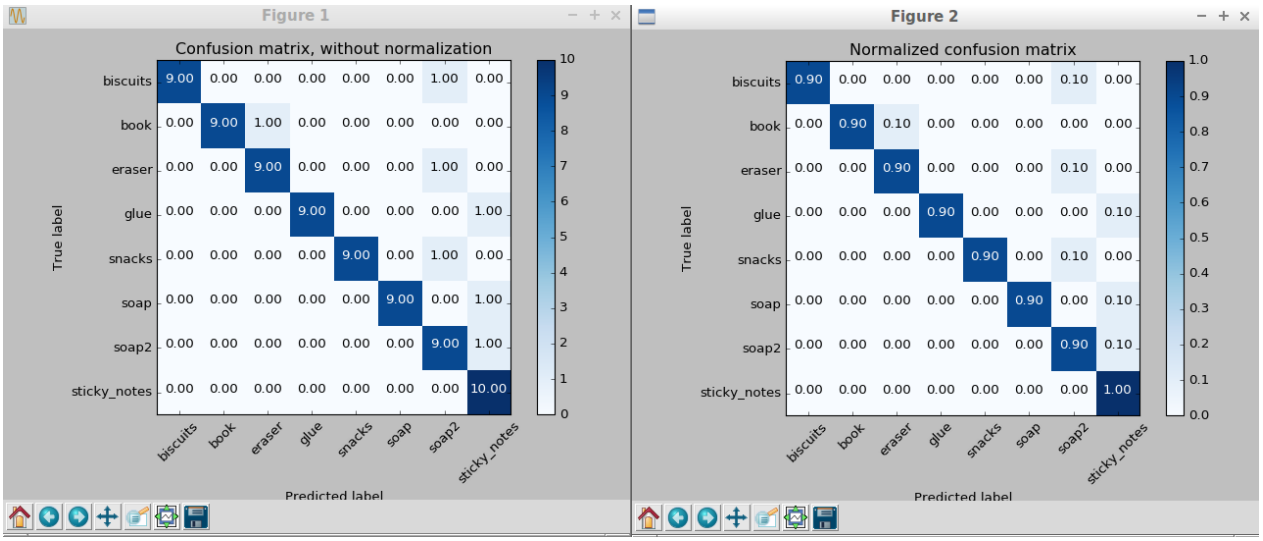
Project 3: 3D Perception Project, Udacity Robotics Nanodegree

Bahadır Özkan, December 2017

Introduction

In this 3D perception project, PR2 robot needs to identify the items on the table in front. There are total of 8 items which in first scenario biscuits, soap and soap2, in the second scenario, biscuits, soap, book, soap2, glue and in the last scenario, sticky notes, book, snacks, biscuits, eraser, soap2, soap and glue are available. The challenge was to identify all 3 items in first scenario, 4/5 of the items in second and 6/8 of the items in the last scenario.

In order to identify the items a LinearSVC classifier was used. Random orientations of the items were captured for object recognition. I have used 10 random orientations due to the time it takes to perform this task however 50 or 100 orientations would give better accuracy. A hand engineered SVC algorithm with 10 loops gave an accuracy of 0.91 and was sufficient to recognize all items correct. Confusion matrices which show the performance of the classifier are as follows:



256 bins were used for histograms. Histograms are normalized because we are seeking for relations. Otherwise we would not know how big the influence of one case over total number of cases. Using HSV format increases the performance. Different random orientations means different lighting conditions and HSV is better at handling these situations than RGB.

Perception Pipeline

Function pcl_callback was used to clean the cloud, do segmentation and detect objects using various filters, classifier etc. and to call pr2_mover function which we will discuss below.

Pass-through filters were used for both x and z dimensions. The x dimension filter had resolved the issue on mislabeling additional “snacks”. Corners of the boxes were identified as snacks in the previous state.

A portion of the pcl_callback function can be found below:

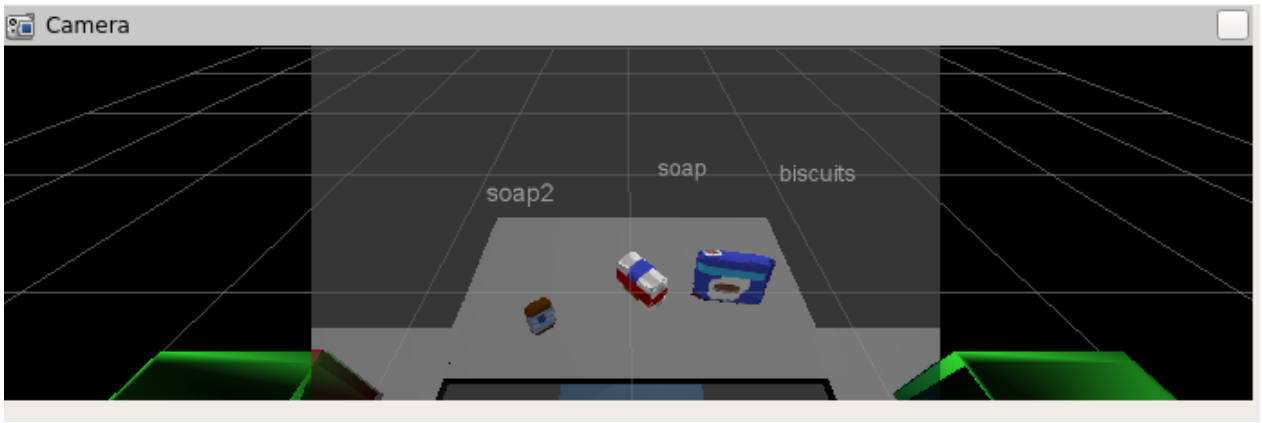
```
# Finally call the filter function for magic
cloud_filtered = outlier_filter.filter()

# TODO: Voxel Grid Downsampling
# Create a VoxelGrid filter object for our input point cloud
vox = cloud.make_voxel_grid_filter()

# Choose a voxel (also known as leaf) size
# Note: this (1) is a poor choice of leaf size
# Experiment and find the appropriate 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
cloud_filtered = vox.filter()
# TODO: PassThrough Filter
# PassThrough filter
# Create a PassThrough filter object.
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()

passthrough_x = cloud_filtered.make_passthrough_filter()

# Assign axis and range to the passthrough filter object.
filter_axis = 'x'
passthrough_x.set_filter_field_name(filter_axis)
axis_min = 0.33
axis_max = 1.0
passthrough_x.set_filter_limits(axis_min, axis_max)

cloud_filtered = passthrough_x.filter()
```

Function pr2_mover returns the necessary yaml documents. To do that first objects' positions are computed via taking the mean of the points in x,y and z dimensions. Then these points are appended in a list. Then, the pick positions, which arm to use and place positions are computed as shown below:

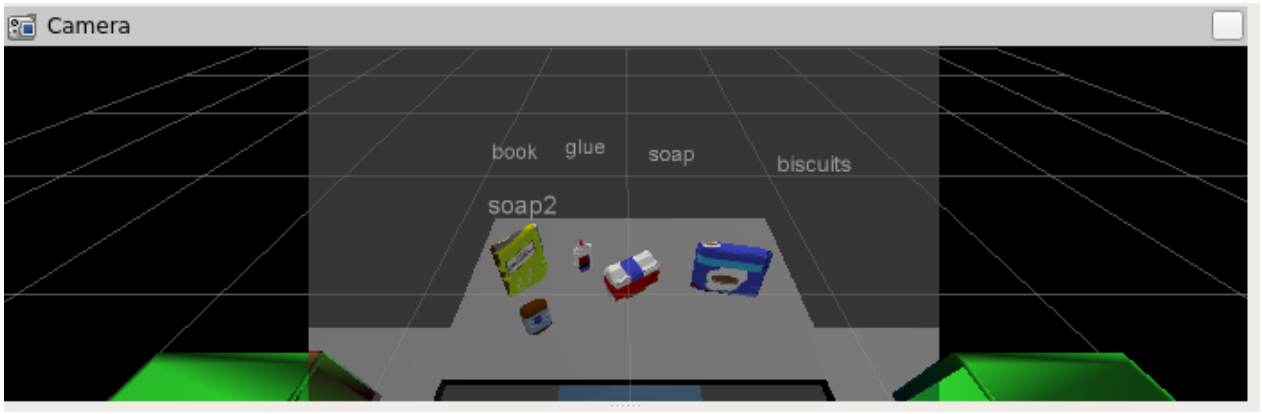
```
    for object in object_list:
        labels.append(object.label)
        print(labels)
        points_arr = ros_to_pcl(object.cloud).to_array()
        # TODO: Get the PointCloud for a given object and obtain it's centroid
        centroid = np.mean(points_arr, axis=0)[:3]
        centroids.append(centroid)
        obj_label[object.label] = centroid

    for i in range(0,len(object_list_param)):
        object_name.data = object_list_param[i]['name']
        object_group = object_list_param[i]['group']

        pick_pose.position.x = np.asscalar(obj_label[object_name.data][0])
        pick_pose.position.y = np.asscalar(obj_label[object_name.data][1])
        pick_pose.position.z = np.asscalar(obj_label[object_name.data][2])

    # TODO: Assign the arm to be used for pick_place

        if object_list_param[i]['group'] == 'red':
                arm_name.data = 'left'
         -      arm_name.data = 'right'


        if arm_name.data == 'left':
                box_target = box_left
        else:
                box_target = box_right

        place_pose.position.x = box_target[0]
        place_pose.position.y = box_target[1]
        place_pose.position.z = box_target[2]

        yaml_dict = make_yaml_dict(test_scene_num, arm_name, object_name, pick_pose, place_pose)
        dict_list.append(yaml_dict)
```

Finally, for all three scenarios screenshots are as follows:

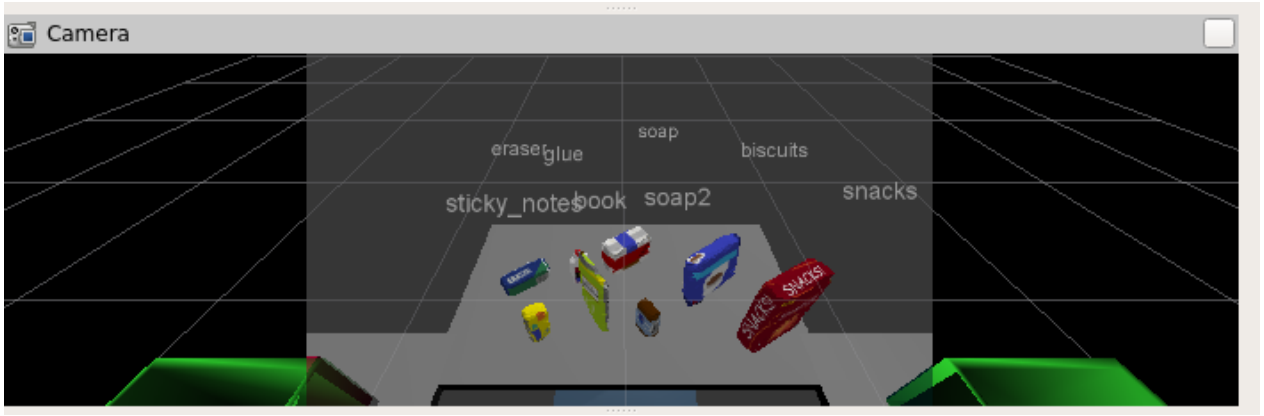
Scene 1:



Scene 2:



Scene 3:



As it can be seen, perception pipeline can recognize all items correctly and if commented out code is used, PR2 can perform Pick & Place operation.

Future Enhancements

However PR2 can perform Pick & Place, it occasionally collides with an item other than the target and this could be improved using a collision mapping. I did not work on the challenge part of the project. Classifier could be trained with more random orientations to get a better accuracy however as stated above the performance was satisfactory for these tabletop scenes. Currently PR2's grasp is not satisfactory. I have read on Slack that some student recommended applying friction on items in the simulation environment. This could help on the Pick & Place operation.