PERCEPTION PROJECT

Table des matières

The structure of the code **project.py**

The project consists of imports, the helper functions, pcl_callback(pcl_msg), and pr2_mover(object_list) and the main function.

In main, we initialize the node, we subscribe to the /pr2/world/points topic, and create publishers where we publish filtered pcl cluster, objects, table, detected objects and /pr2/world_joint_controller/command in order to rotate the robot from the base.

Afterwards, we load the model created by sklearn by pickle, create the necessary classifier encoder and scalers. Finally, we initialize the color list and make the rospy spin until we tell it to shutdown.

Helper functions:

```
def get_normals(cloud):
        get_normals_prox = rospy.ServiceProxy('/feature_extractor/get_normals', GetNormals)
        return get_normals_prox(cloud).cluster
        #search in list of dictionaries
def search_dict(key1,value1,key2,list_dicts):

        selected_dict=[element for element in list_dicts if element[key1]==value1][0]#accessing t
        return selected_dict[key2]

# Helper function to create a yaml friendly dictionary from ROS messages
def make_yaml_dict(test_scene_num, arm_name, object_name, pick_pose, place_pose):
        yaml_dict = {}
        yaml_dict["test_scene_num"] = test_scene_num.data
        yaml_dict["arm_name"]  = arm_name.data
        yaml_dict["object_name"] = object_name.data
        yaml_dict["pick_pose"] = message_converter.convert_ros_message_to_dictionary(pick_pose)
        yaml_dict["place_pose"] = message_converter.convert_ros_message_to_dictionary(place_pose)
        return yaml_dict

# Helper function to output to yaml file
def send_to_yaml(yaml_filename, dict_list):
        data_dict = {"object_list": dict_list}
        with open(yaml_filename, 'w') as outfile:
                yaml.dump(data_dict, outfile, default_flow_style=False)
```

1) Get normal from the parameter server
2) Searching the value of a key in the list of dictionaries with the known key and known value.
3) Creating the yaml dictionary from the std.msgs that we are going to create in the pr2_mover function.
4) Saving the dict. List in the yaml_filename which specifies the relative path to the script or absolut path.

pcl_callback(pcl_msg) or perception pipeline

Parts 1,2,3 are located here.

Why this? We receive the ros message, publish the point cloud after filtering, segmentation and clustering, then we detect the objects in rviz and pass it as the parameter to the pr2_mover function.

PART1: Filtering

First of all, we recieve the data in ros format, so we convert it to pcl point cloud with **ros_to_pcl.** Then we apply outlier filter to eliminate the noise with the values of mean_k as 3 and x=0.0001. Then, we apply the voxel grid downsampling, to make a decomposition of 3d shape in small cubes(3d pixels) point-cloud. The leaf size applied in the ex1 didn't work, the leaf_size chosen is 0.005. Then we apply passthrough filter in order to obtain the area of INTEREST. We apply in the z-axis since we want to remove table base, and in the y-axis(-0.45 0.45) to eliminate false positive for object-detection;

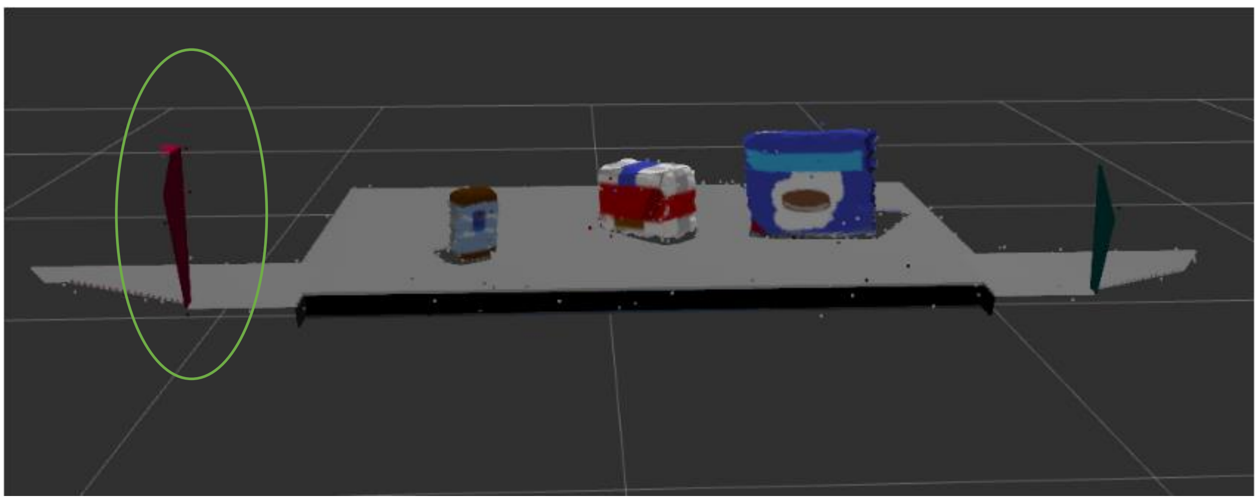

Figure 1(before path through in y-direction…detects objects as edges of bins)

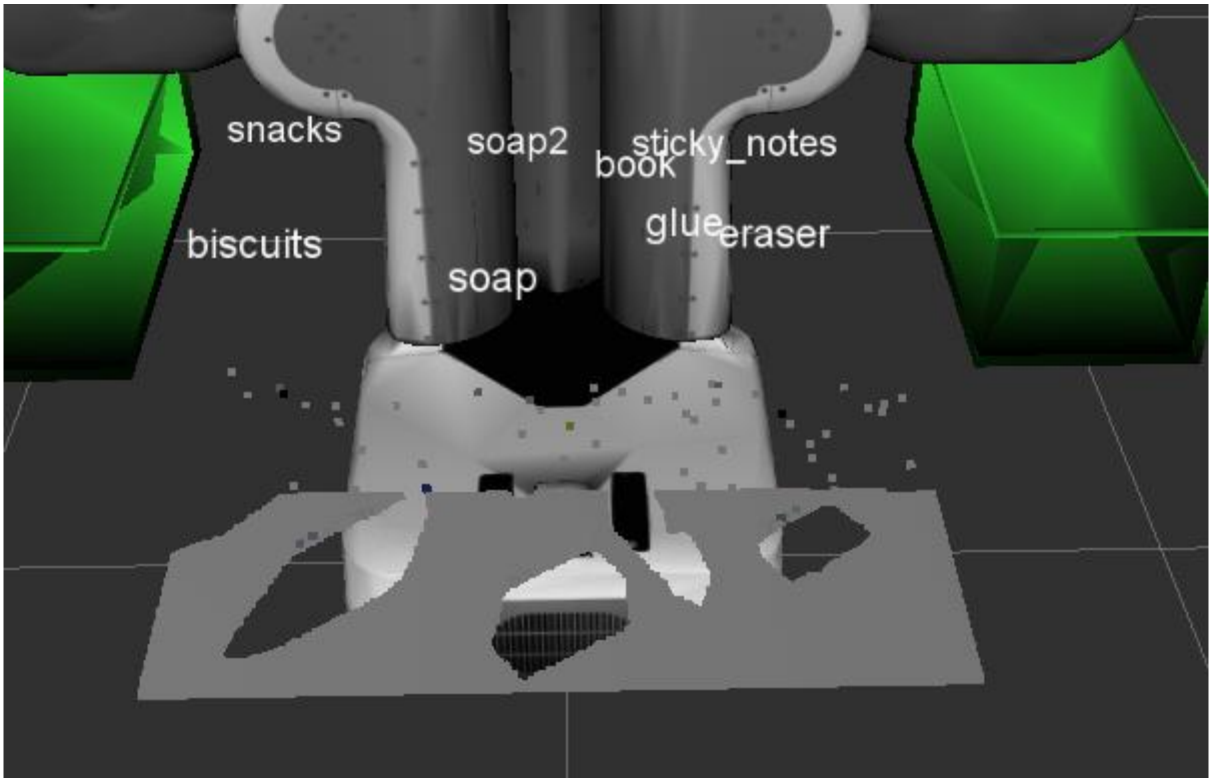

Figure 2(table) after ransac, outlier and paththrough filtering

otherwise the robot sees the edges of the tables as objects. Finally, we apply the ransac segmentation to identify the plane(table) and different objects:cloud_table and cloud_objects.

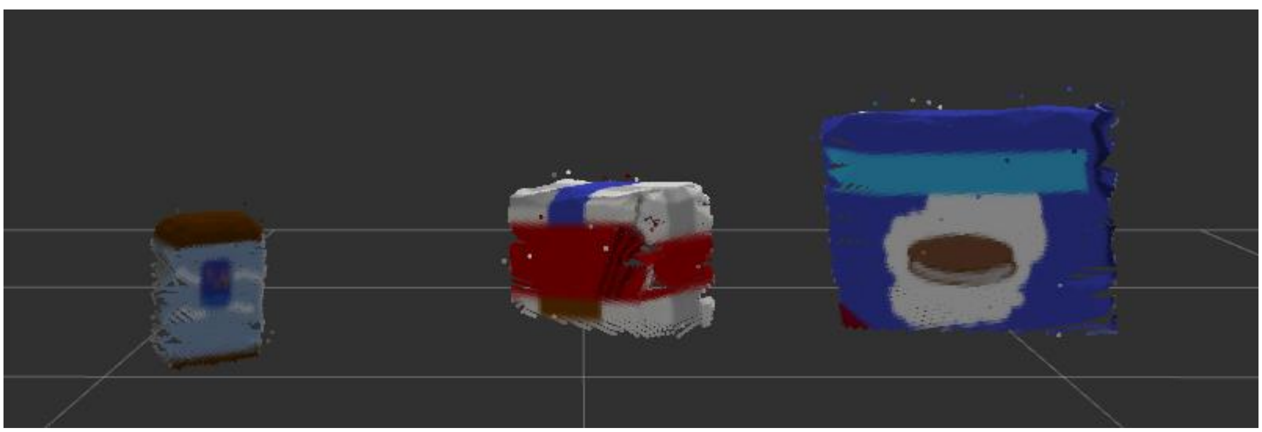

Figure 3(objects after RANSAC plane filtering)

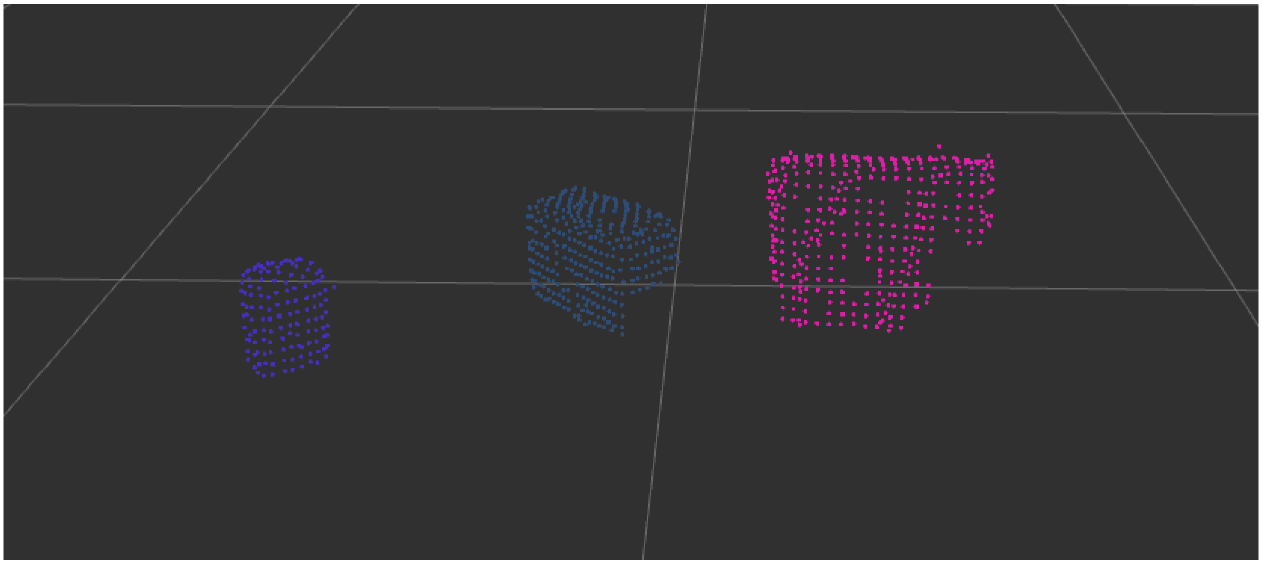

Figure 4(3 clusters with different colors)

PART2: clustering

We use the cloud_objects that we obtained after ransac segmentation. We construct a k-d tree from the cloud_objects point cloud and use the Euclidean clustering. The values used are

ec.set_ClusterTolerance(0.03) #the distance between points to clusters

ec.set_MinClusterSize(10)

ec.set_MaxClusterSize(9000)

set_ClusterTolerance shouldn't be very large (multiple objects as 1 cluster) nor very small(1 object is seen as multiple objects)

We extract cluster indices, and then we do 2 for loops to color the x,y,z points of a specific cloud to a certain color.

PART3: Object recognition:

The idea is that we modify the list of models in `capture_features.py` and then we generate the training set that we use to construct our model. The more iterations we have to generate the training set, the better is the prediction.

I implement in the same for loops we implement the object-recognition svm (supervised machine learning algorithm) from "sklearn" library based on normal histograms and color_histograms. The linear kernel is the most appropriate in this case.

So we use helper function like compute color_histograms:

We apply np.histogram for each channel, then we concatenate to the hist_features. Finally, we normalize the features. By concatenating features and color histograms, we obtain the "feature" by which we detect our object and then publish to rviz.

Pr2_mover

Why? We have a list of detected objects as a parameter and we have access to the parameter server where we can receive the following data:

```
object_list:
  - name: biscuits
    group: green
  - name: soap
    group: green
  - name: soap2
    group: red
```

```
object_list_param = rospy.get_param('/object_list')
```

group means to which bean we need to put a particular object.

Overall, we have the following parameters that are passed to the pick_place routine:

Name	Message Type	Description	Valid Values
test_scene_num	`std_msgs/Int32`	The test scene you are working with	1,2,3
object_name	`std_msgs/String`	Name of the object, obtained from the pick-list	-
arm_name	`std_msgs/String`	Name of the arm	right, left
pick_pose	`geometry_msgs/Pose`	Calculated Pose of recognized object's centroid	-
place_pose	`geometry_msgs/Pose`	Object placement Pose	-

The necessary steps for pr2_mover function:

1) Test_scene_name and object_name from the object_list.yaml
2) Arm_name is found by search_dict(helper) and the group(red or green bin)
3) Pick_pose is the centroid of the object in int32() format that is found using point cloud of detected_object(parameter of pr2_mover function) and a label of objects(biscuit, book…)

```
# TODO: Get the PointCloud for a given object and obtain it's centroid
for object in object_list:
    labels.append(object.label)
    points_arr=ros_to_pcl(object.cloud)

    center=np.mean(points_arr,axis=0)[:3]
    centroids.append(center)
```

4)

```yaml
dropbox:
  - name: left
    group: red
    position: [0,0.71,0.605]
  - name: right
    group: green
    position: [0,-0.71,0.605]
```

```
 The place pose is extracted from
the dropbox.yaml of the parameter
server
```

5) Finally, for each obect we construct a dictionary and save the list as .yaml file.

Additional Comments:

After publishing to rviz, the labels, we add all the detected objects to the list, and publish them for the usage by pr2_mover function.

test_scene_num = Int32()#put in the message

object_name = String()#put in the message

object_group = String()#put in the message

arm_name=String()#put in the message

pick_pose = Pose()#put in the message

place_pose = Pose()#put in the message

we initialize the variables of std.msgs type that we populate and pass in to the pick_place routine and the send_yaml function. We compute centroids and all the parameters (name, group, arm_name…) from the parameter_list of the parameter_server. Afterwards, we add to yaml_dict, and add it to the yaml_dict list in each loop.

Example of search:

arm_name.data=search_dict('group',object_group.data,'name',dropbox_param) using the following function:

Then we send_to_yaml and save it as output#.yaml file.

Also, we rotate the robot to capture the location of bins by publishing to pr2_robot=rospy.Publisher("/pr2/world_joint_controller/command",Float64,queue_size=1);

and sending:

pr2_robot.publish(-1.57)

rospy.sleep(15.0)# the time is seconds during which the base of the robot rotates

FUTURE IMPROVEMENTS:

It is possible to collision mapping and using the pick_place_server to execute the pick and place operation.I started to implement it, i.e I published 1.57 rad as joint angle to base joint of a robot. Afterwards, the we need to place the objects near each other in the bins, and not on top of each other. So, with each iterations we need to increment the x or y coordinate of the place pose. Note, that The left and right arms of PR2 are controlled using Moveit. Due to some issue, I still need to find where are the issues to execute pick_place routine.

Conclusion:

I have made changes to filtering, added the outlier filter since we didn't use it in the exercises, added paththrough filter in y direction, changed the voxel LEAF, the parameters of clustering completely. This project comprises panipulation of various kinds of filters of plc, svm of skLearn and ros elements. Udacity Slack and the mentor was a great help!

SCREENSHOTS

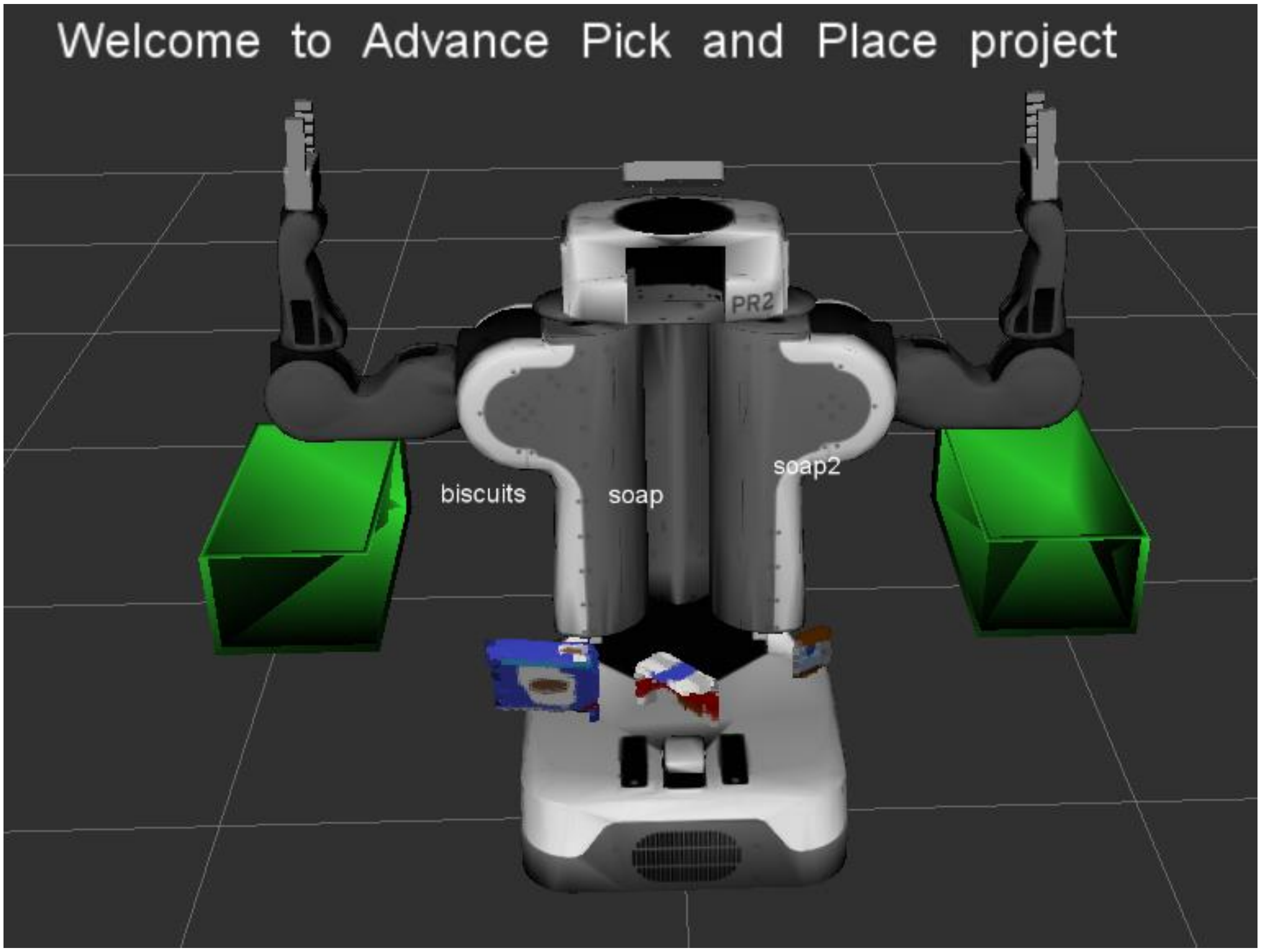

Figure 5(test world#1) detexted all 3 objects in rviz

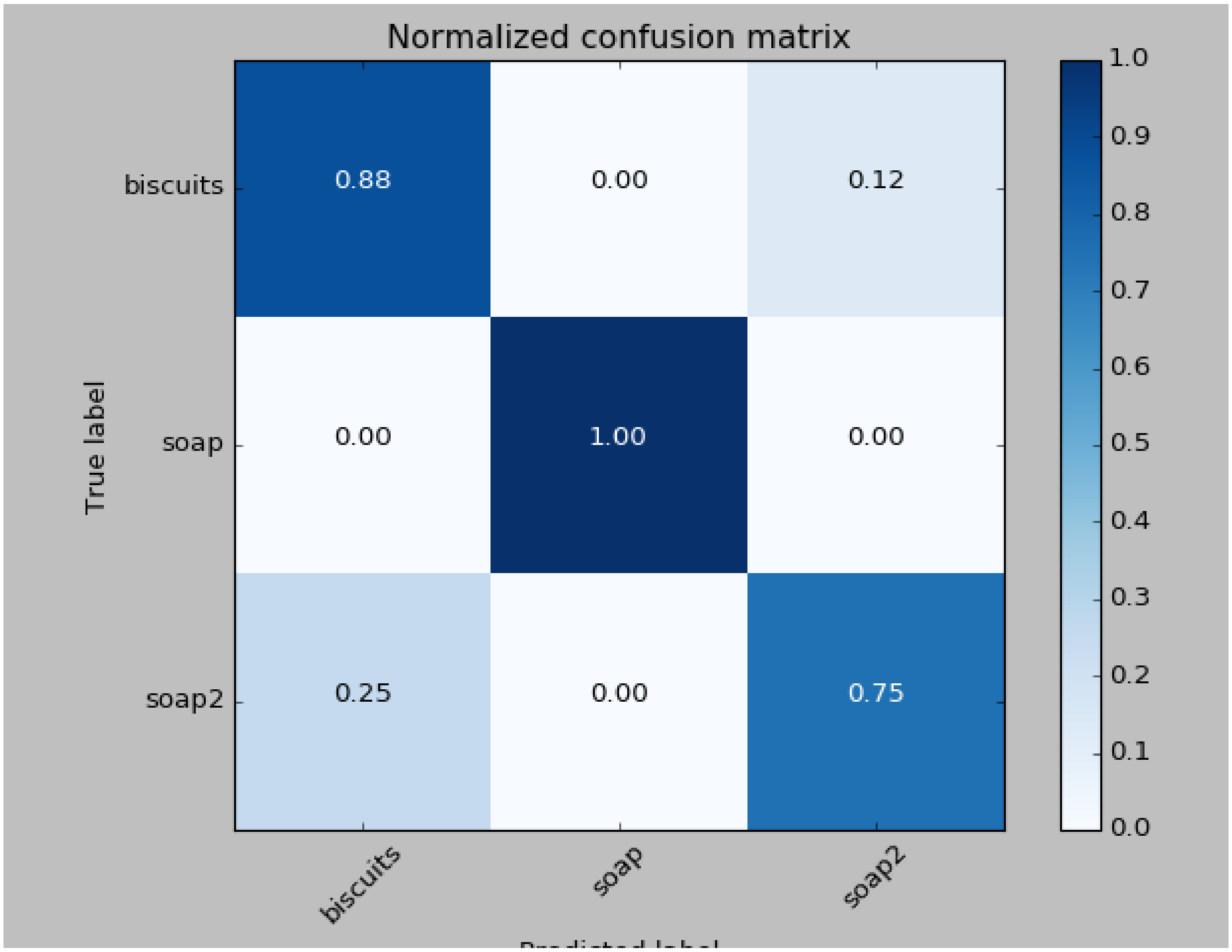

Figure 6(confusion matrix world#1)

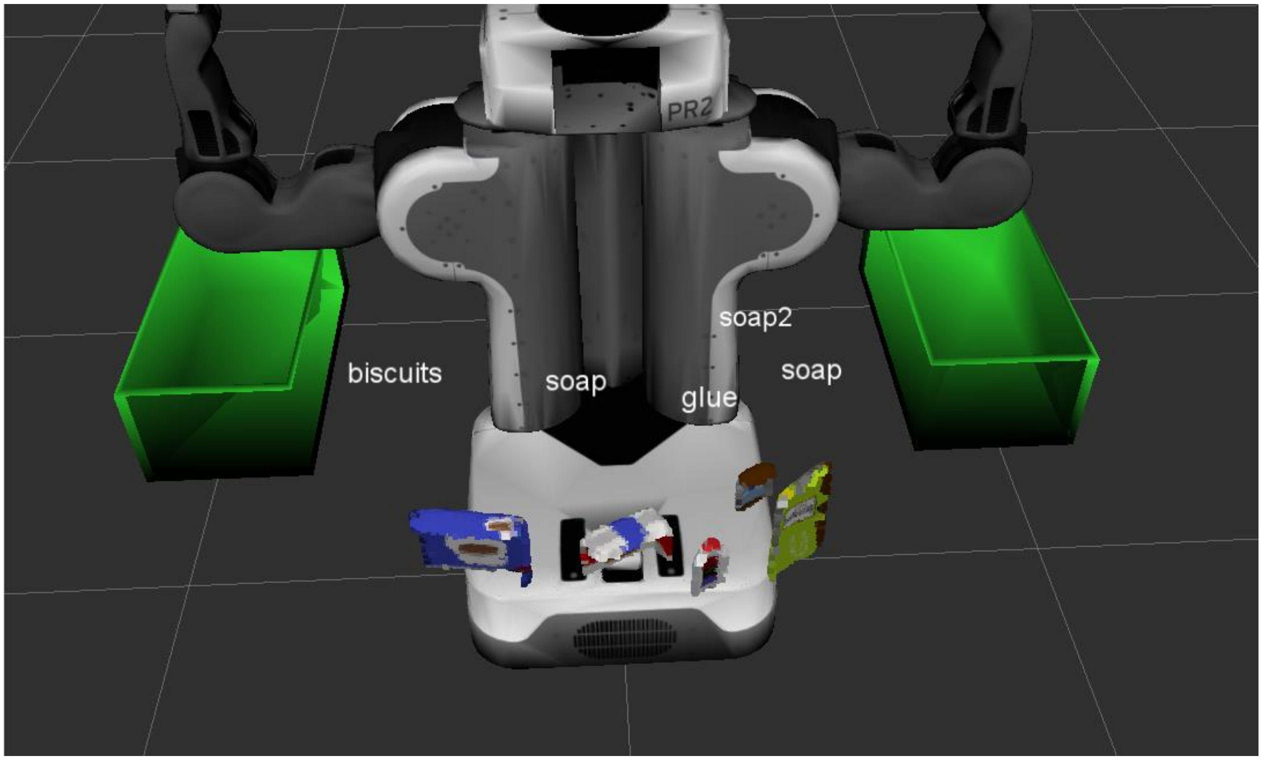

Figure 7(world#2)

We have 4 objects in the output_2.yaml since 1 object isn't detected well: book, so we won't have 2 soaps in the output.

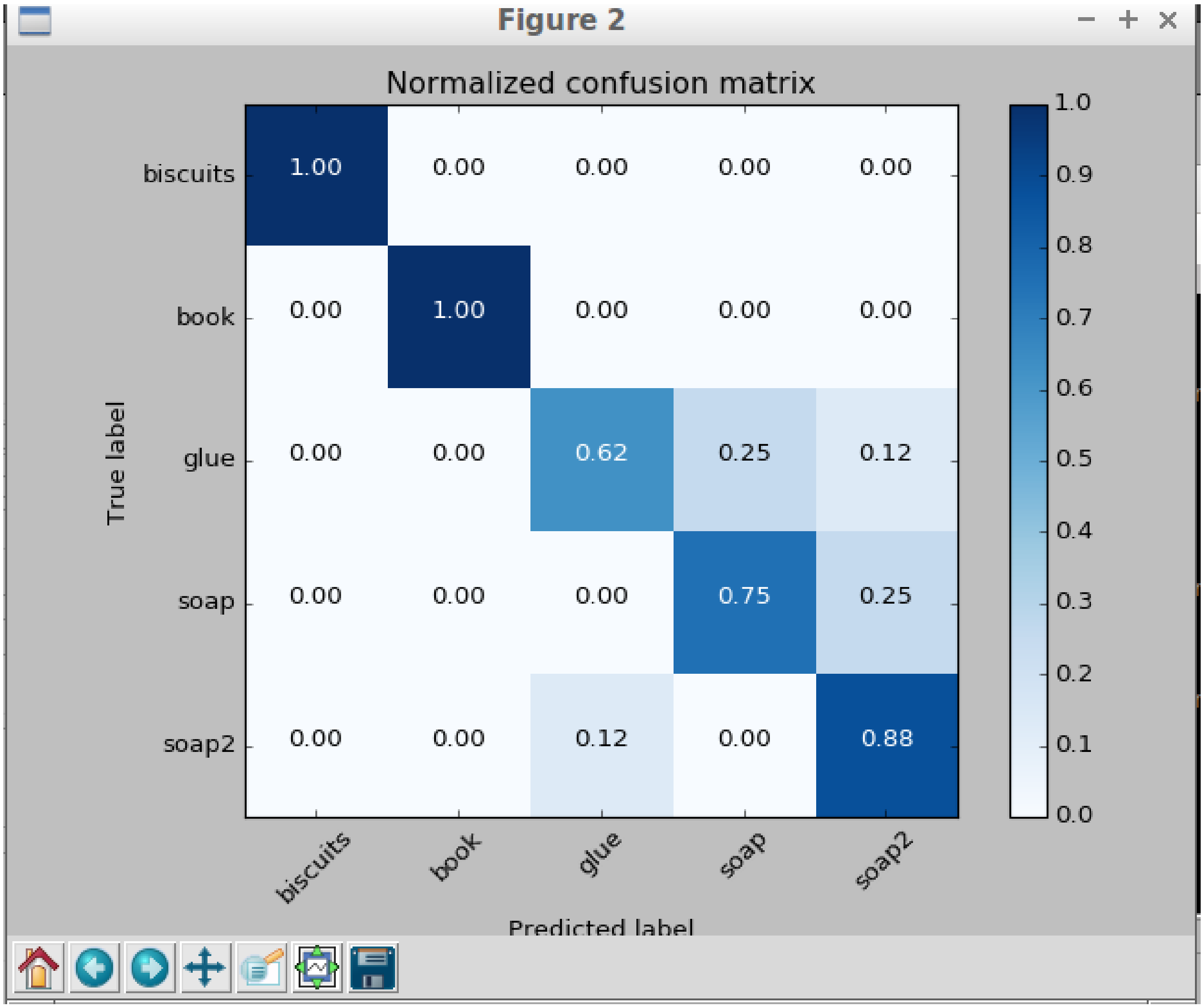

Figure 8(test_world#2 detected 4/5 objects)

World_3 :100%

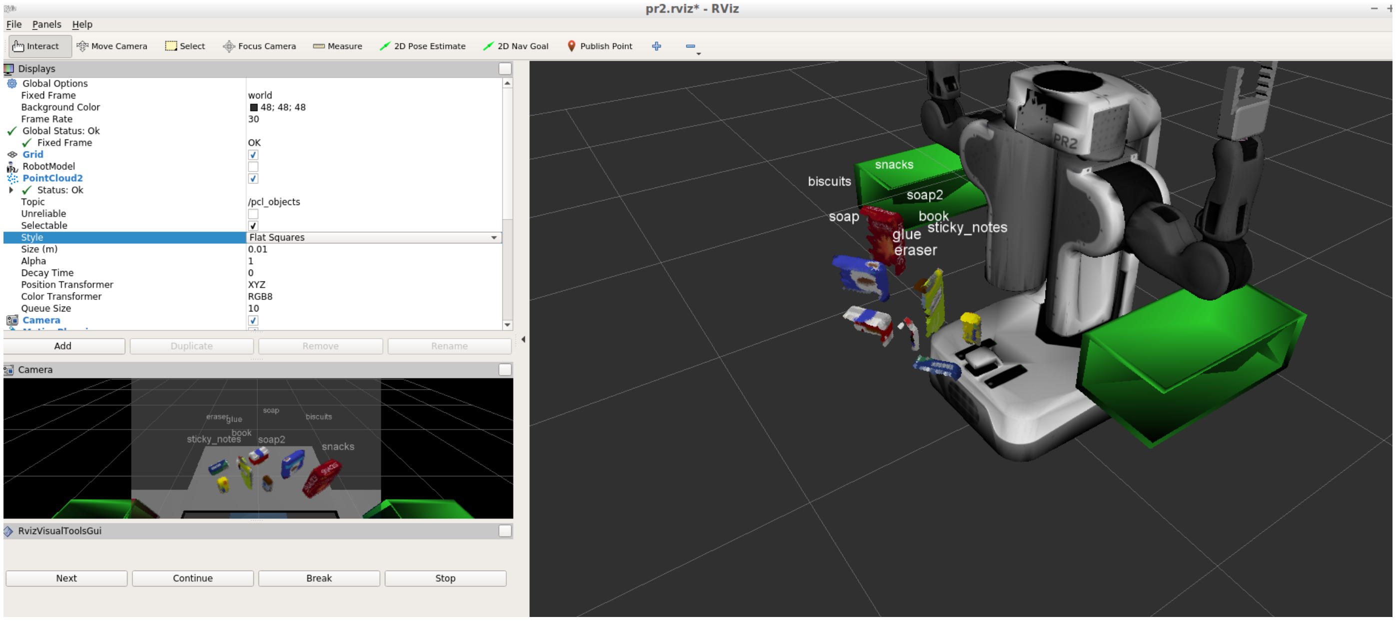

Figure 9(world#3)

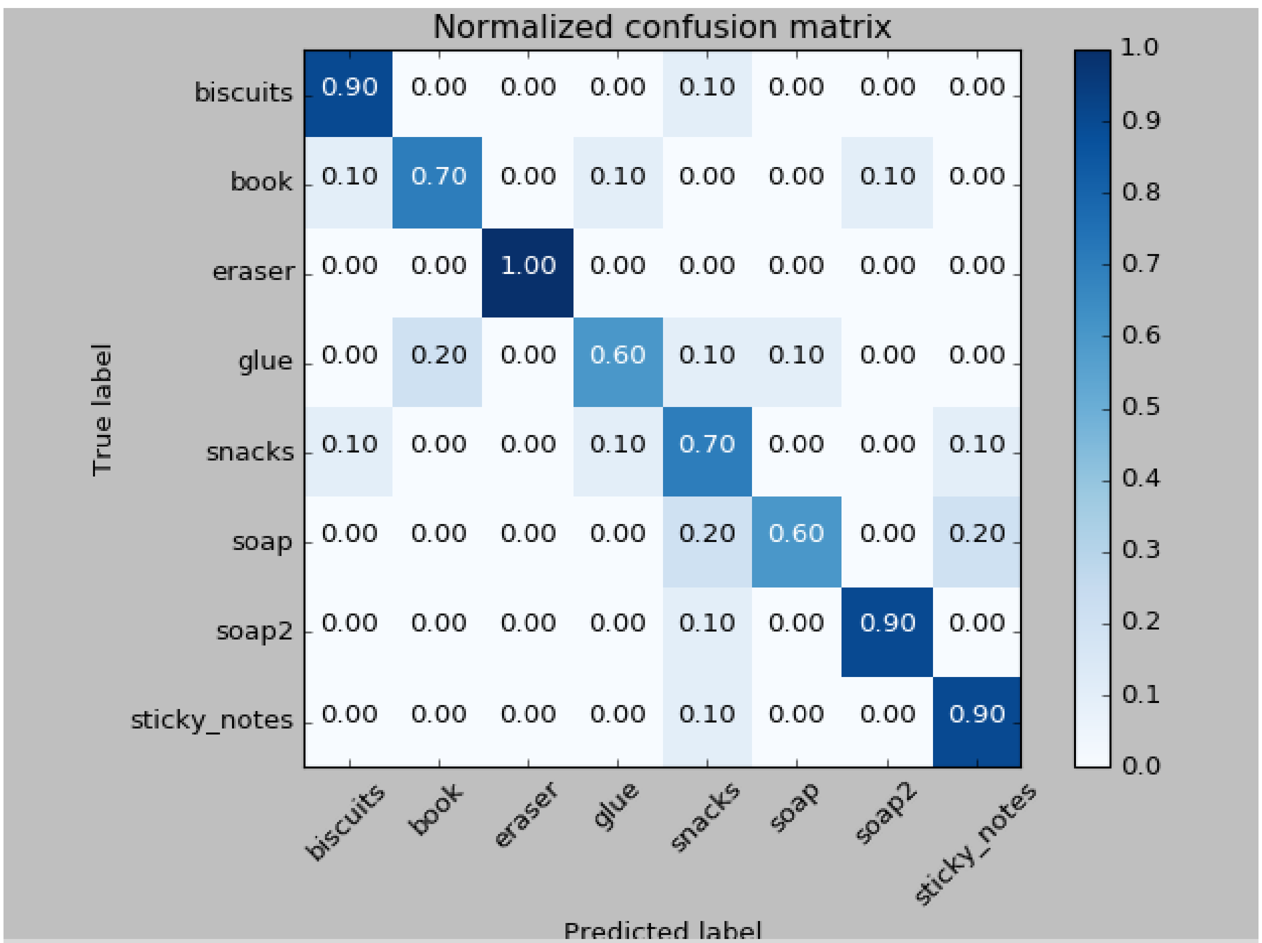

Figure 10(test_world#3 detected all objects)

Annexes:

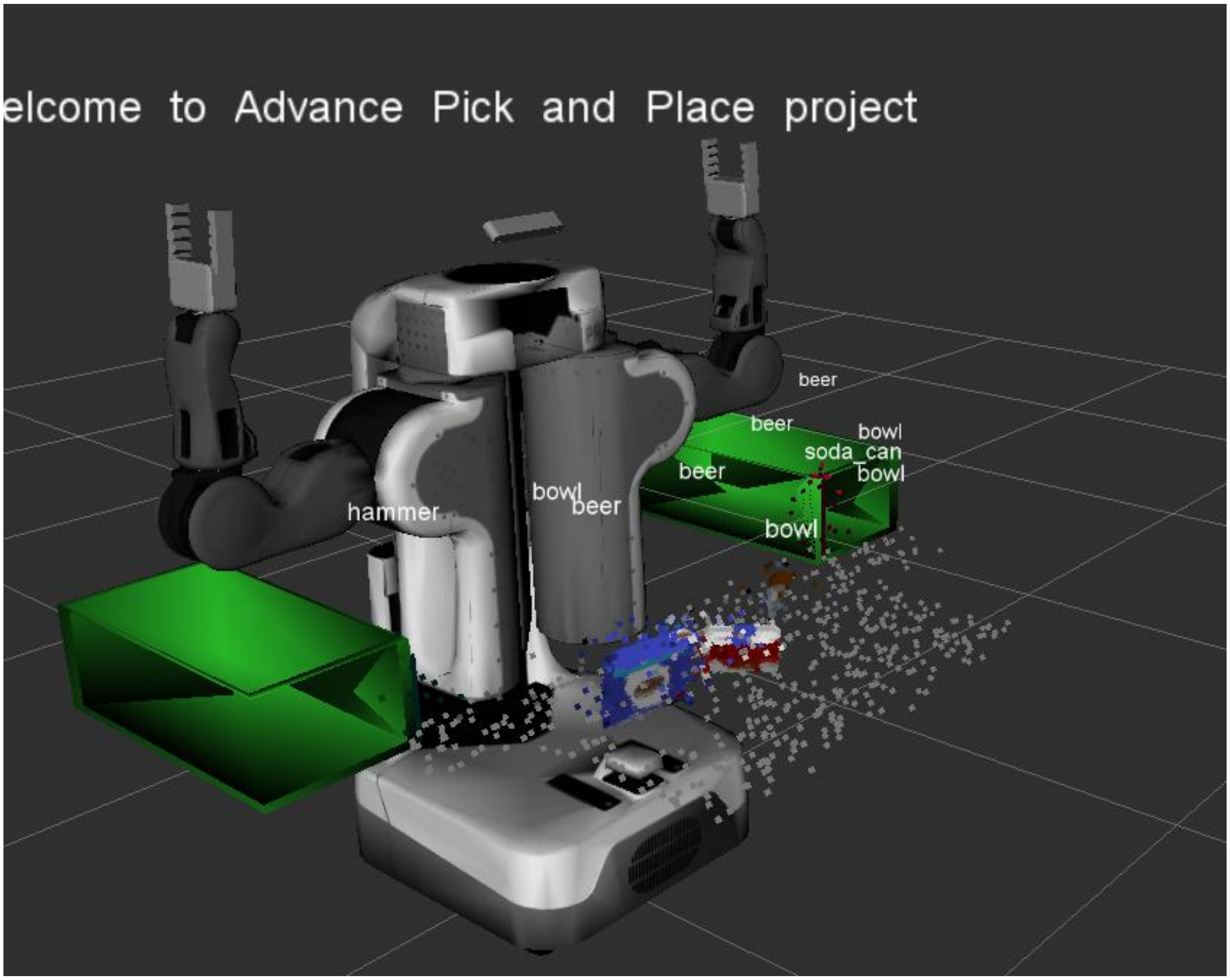

Figure 11(without filtering and some random data for of object recognition)

We see that we have a lot of noise and many more objects than there are in the reality.

Functions that are used to generate features for SVM:

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

    # Compute histograms for the clusters
    point_colors_list = []
    using_hsv=True
    # 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=32
    bins_range=(0, 256)
    h_hist=np.histogram(channel_1_vals, bins=nbins, range=bins_range)
    s_hist=np.histogram(channel_2_vals, bins=nbins, range=bins_range)
    v_hist=np.histogram(channel_3_vals, bins=nbins, range=bins_range)
    # TODO: Concatenate and normalize the histograms
    hist_features=np.concatenate((h_hist[0],s_hist[0],v_hist[0])).astype(np.float64)
    # Generate random features for demo mode.
    # Replace normed_features with your feature vector
    #normed_features = np.random.random(96)
    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])
    nbins=32
    bins_range=(0, 256)
    # TODO: Compute histograms of normal values (just like with color)
    x_hist=np.histogram(norm_x_vals, bins=nbins, range=bins_range)
    y_hist=np.histogram(norm_y_vals, bins=nbins, range=bins_range)
    z_hist=np.histogram(norm_z_vals, bins=nbins, range=bins_range)
    # TODO: Concatenate and normalize the histograms
    hist_features=np.concatenate((x_hist[0],y_hist[0],z_hist[0])).astype(np.float64)
    # Generate random features for demo mode.
    # Replace normed_features with your feature vector
    #normed_features = np.random.random(96)
    normed_features=hist_features/np.sum(hist_features)

    return normed_features
```

Commands:

cd ~/catkin_ws/src/sensor_stick/scripts

roslaunch sensor_stick training.launch

```
rosrun sensor_stick capture_features.py
```

to capture features.

rosrun sensor_stick train_svm.py

SVM:

Label: prediction of an svm for an object:is appended to the detected_objects list.

Commands to launch:

```
$ roslaunch pr2_robot pick_place_project.launch
```

and then,

#cd ~/catkin_ws/src/RoboND-Perception-Project/pr2_robot/scripts

```
$ rosrun pr2_robot project_template.py
```

roslaunch sensor_stick training.launch

```
rosrun sensor_stick capture_features.py
```

to capture features.

rosrun sensor_stick train_svm.py to observe the results and generate model.sav

NOTES :

```
# Euclidean Clustering
white_cloud = # Apply function to convert XYZRGB to XYZ
tree = white_cloud.make_kdtree()
```

Once your k-d tree has been constructed, you can perform the cluster extraction like this:

```
# 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)
# NOTE: These are poor choices of clustering parameters
# Your task is to experiment and find values that work for segmenting objects.
ec.set_ClusterTolerance(0.001)
ec.set_MinClusterSize(10)
ec.set_MaxClusterSize(250)
# Search the k-d tree for clusters
ec.set_SearchMethod(tree)
# Extract indices for each of the discovered clusters
cluster_indices = ec.Extract()
```

Color histogram:

Bin is a width of a small bar in histogram.

Voxel filter: decomposes the 3d objects on points

Pass through filter takes the part of the space min, max_distance.

Docs:

1) https://stackoverflow.com/questions/7900882/extract-item-from-list-of-dictionaries
2) http://pointclouds.org/documentation/tutorials/cluster_extraction.php#cluster-extraction
3) http://bit.ly/segmentation-intro-nn
4) http://strawlab.github.io/python-pcl/
5) http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FISHER/RANSAC/