PERCEPTION PROJECT

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; 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.

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)

ec.set_MinClusterSize(10)

ec.set_MaxClusterSize(9000)

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:

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.

In the "main", we import the model.sav that we generated using capture_features.py; in the test world#3 the robot detected all objects due to the fact that I increased the number of iterations for capturing features from 5 to 10.

model = pickle.load(open('model.sav', 'rb'))

clf = model['classifier']

encoder = LabelEncoder()

encoder.classes_ = model['classes']

scaler = model['scaler']

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:

def search_dict(key1,value1,key2,list_dicts):

selected_dict=[element for element in list_dicts if element[key1]==value1][0]#accessing the 1st element in the list of found dictionaries

return selected_dict[key2].

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(1.0)

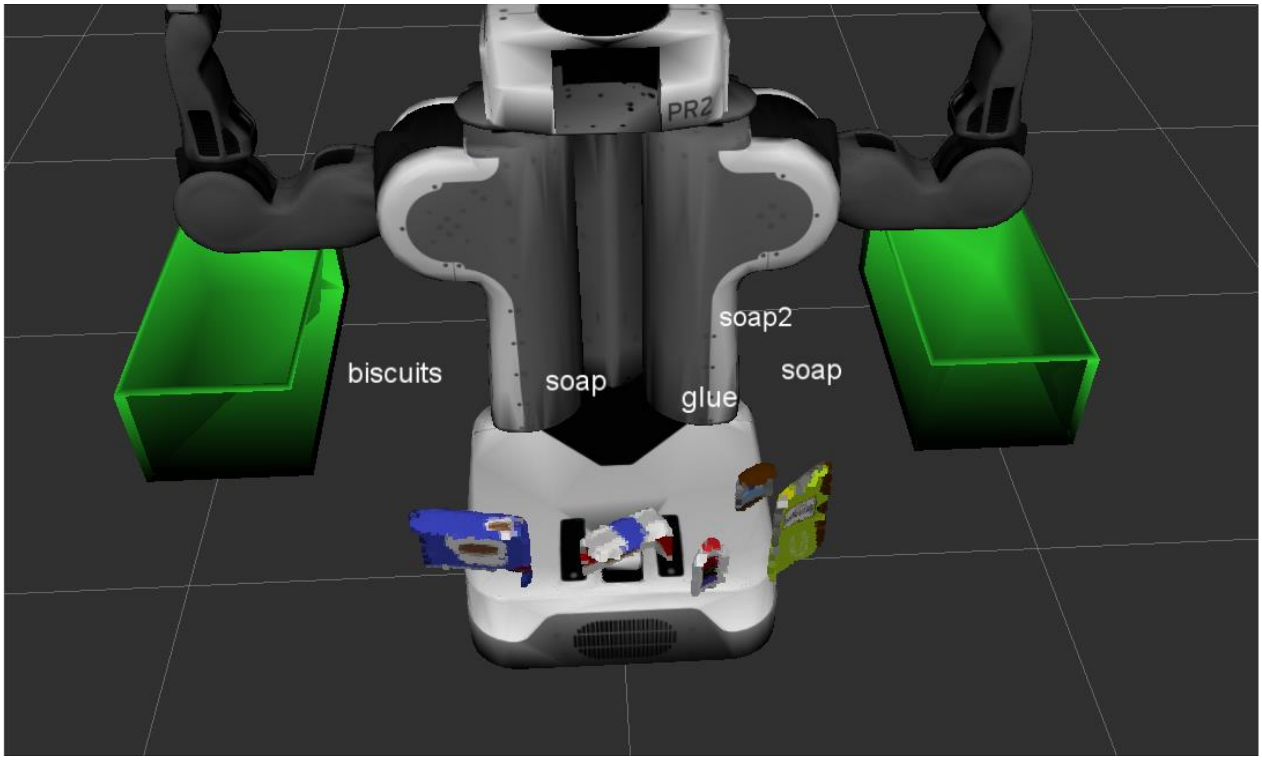

Figure 1(world#2)

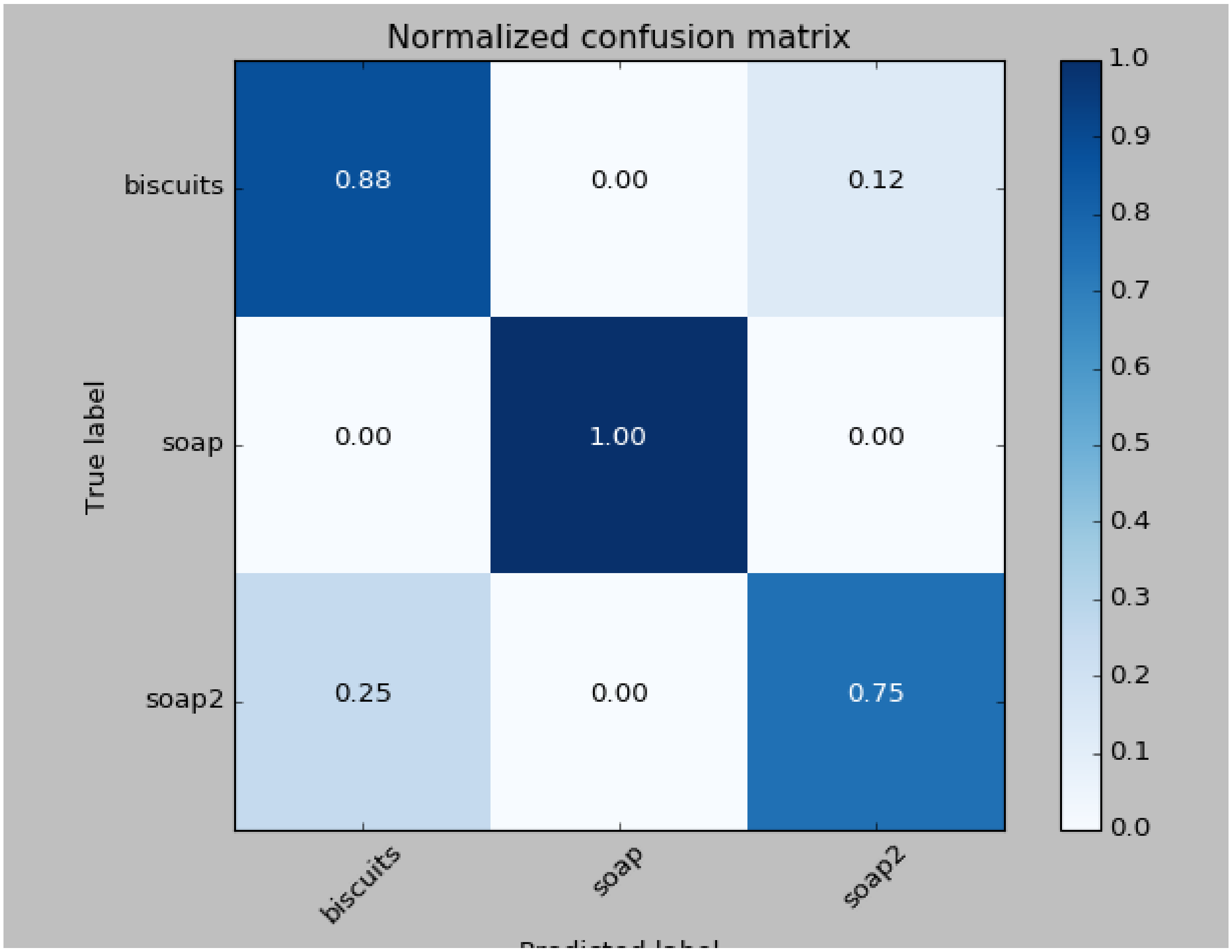

Figure 2(test_world#1) detexted all 3 objects in rviz

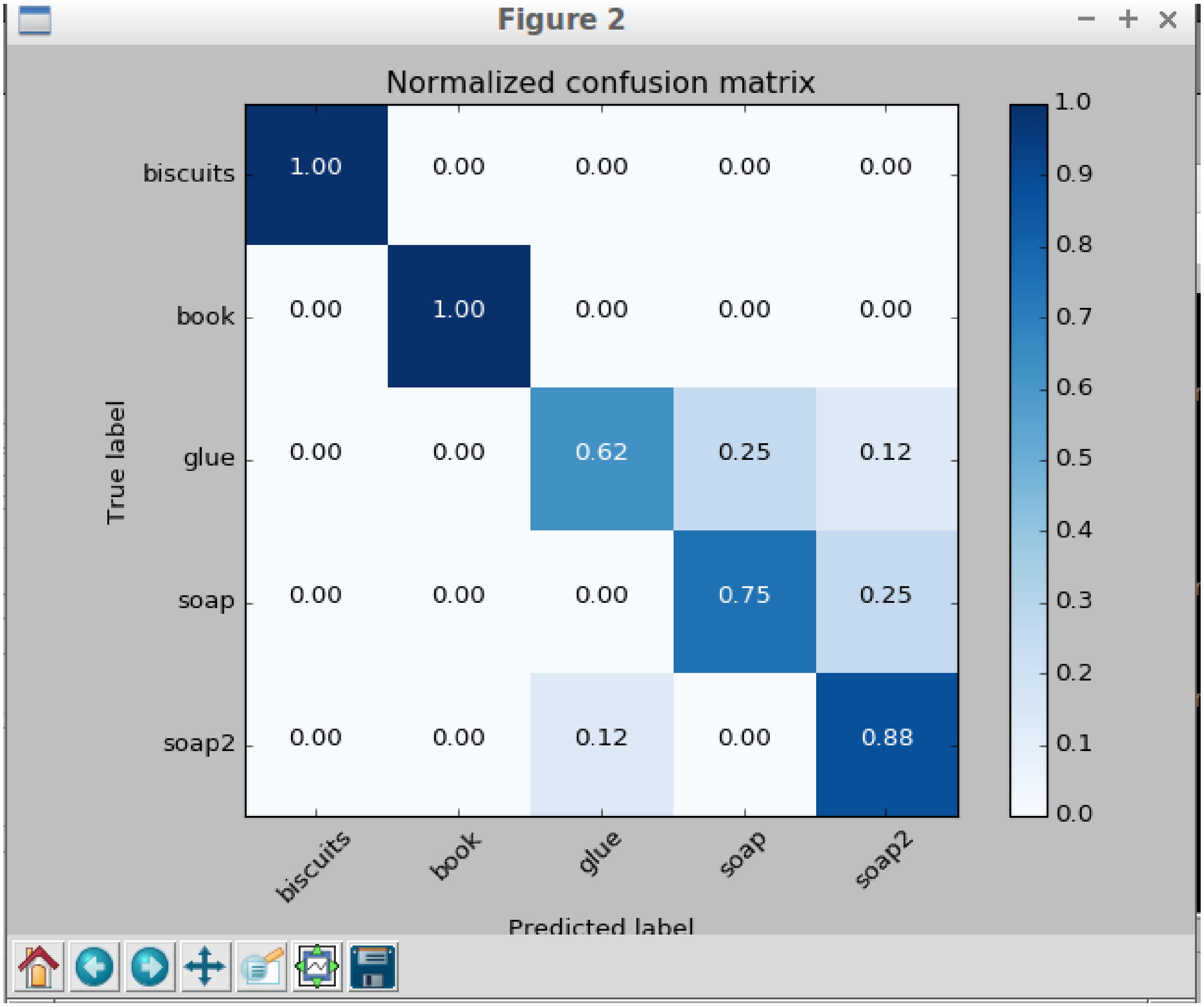

Figure 3(test_world#2 detected 4/5 objects)

World_3 :100%

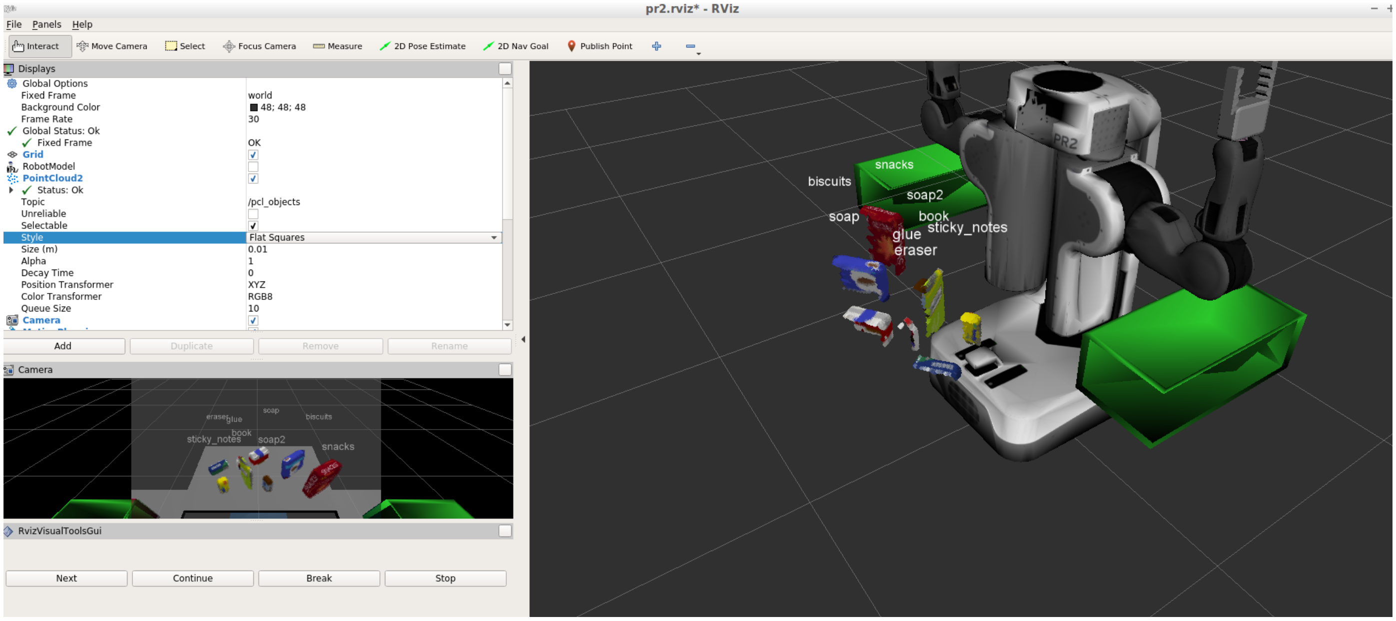

Figure 4(world#3)

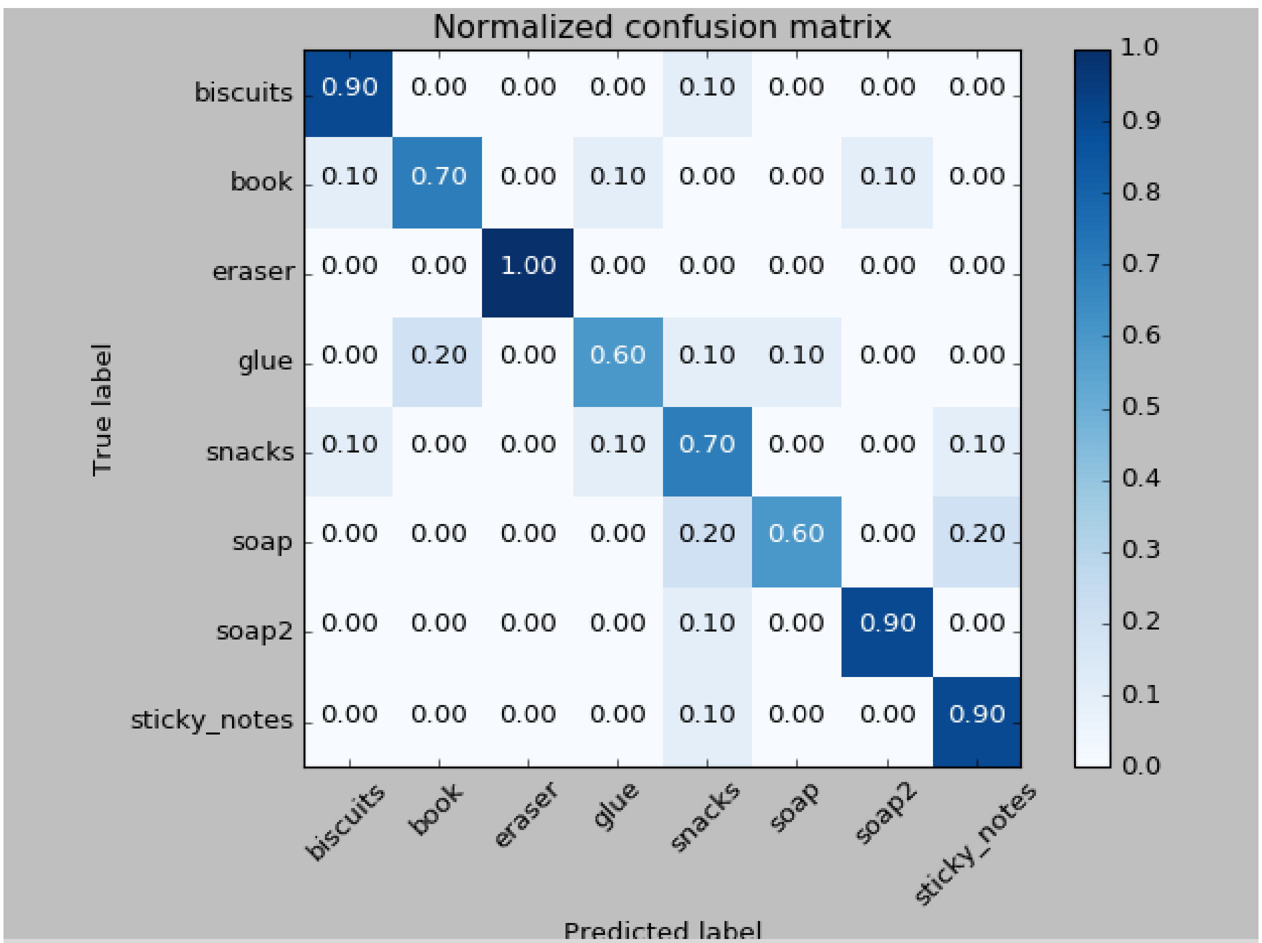

Figure 5(test_world#3 detected all objects)

FUTURE IMPROVEMENTS:

It is possible to generate the collision map for the robot, and place the objects near each other in the bins, and not on top of each other.

Annexe :

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:

https://stackoverflow.com/questions/7900882/extract-item-from-list-of-dictionaries