Project 3D Perception

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

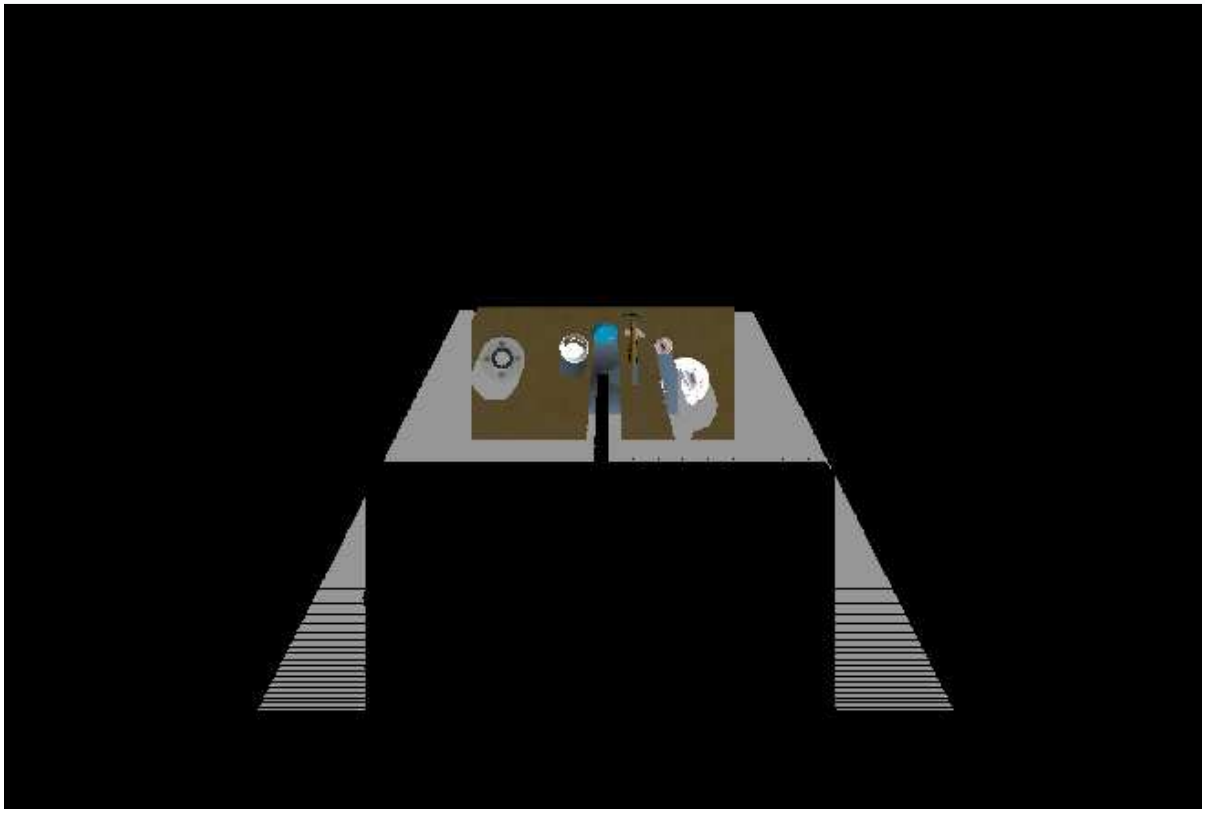
*The **pcl_callback()** function within the template Python script has been filled out to include filtering and RANSAC plane fitting.*

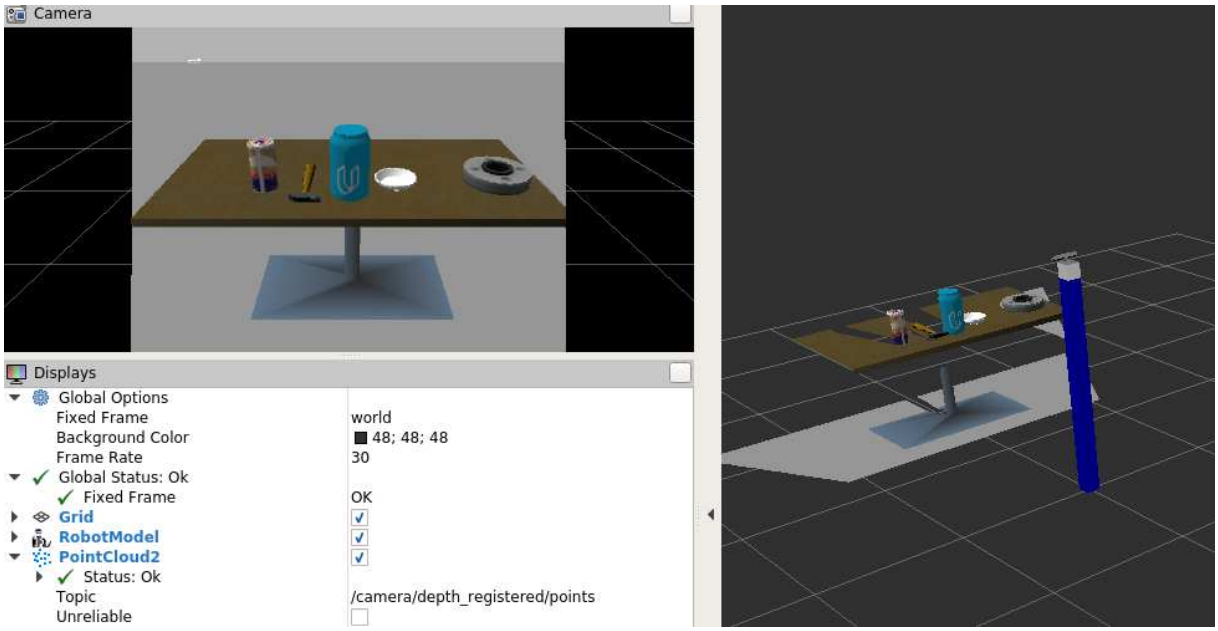
To accomplish these tasks, you will modify the code in the RANSAC.py script, which is provided in the repository in the Exercise-1 folder.

See attached the File: RANSAC.py with the code including Voxel Grid filter, Pass Through filter, RANSAC Plane fitting segmentation, Extract inliers and Outlier Remover Fitting.

Viewing:

$ python RANSAC.py
$ pcl_viewer voxel_downsampled.pcd





2. Segmentation and Clustering

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

Steps for cluster segmentation have been added to the **pcl_callback()** function in the template Python script.

The exercise includes filtering elements from **RANSAC.py** and the clustering for the segmentation using *Euclidean Clustering, Cluster-Mask Point Cloud to visualize each cluster separately with different colors.*

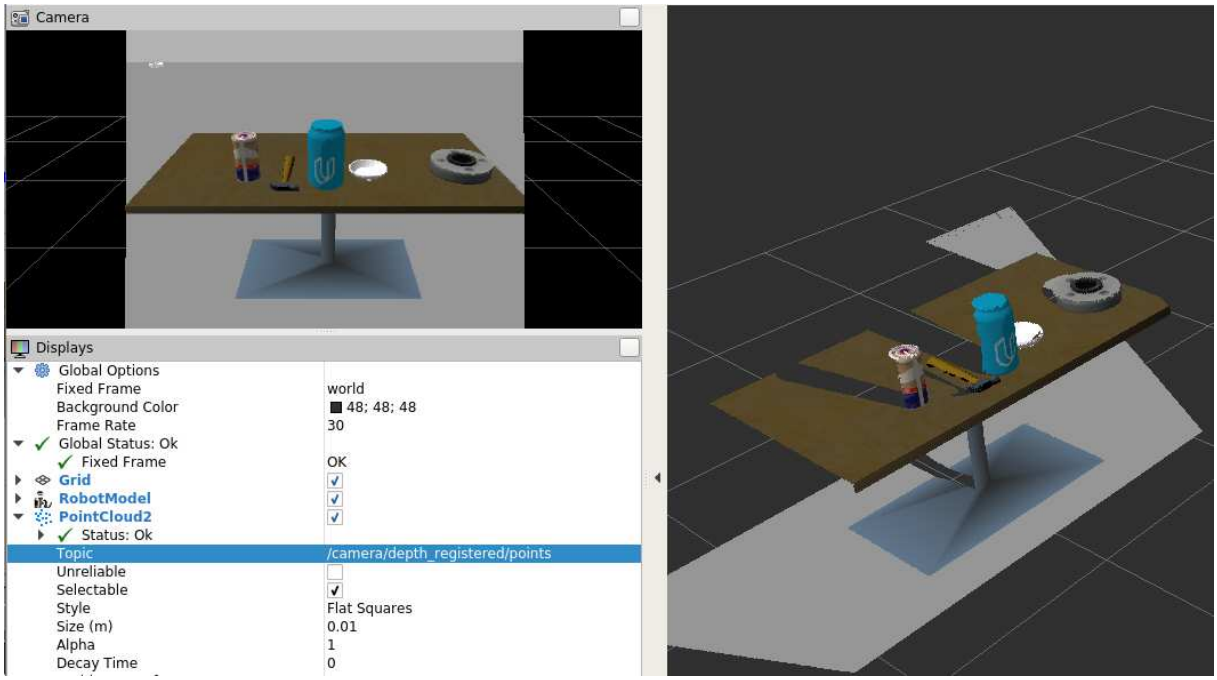
See attached the File: **segmentation.py** with the code.

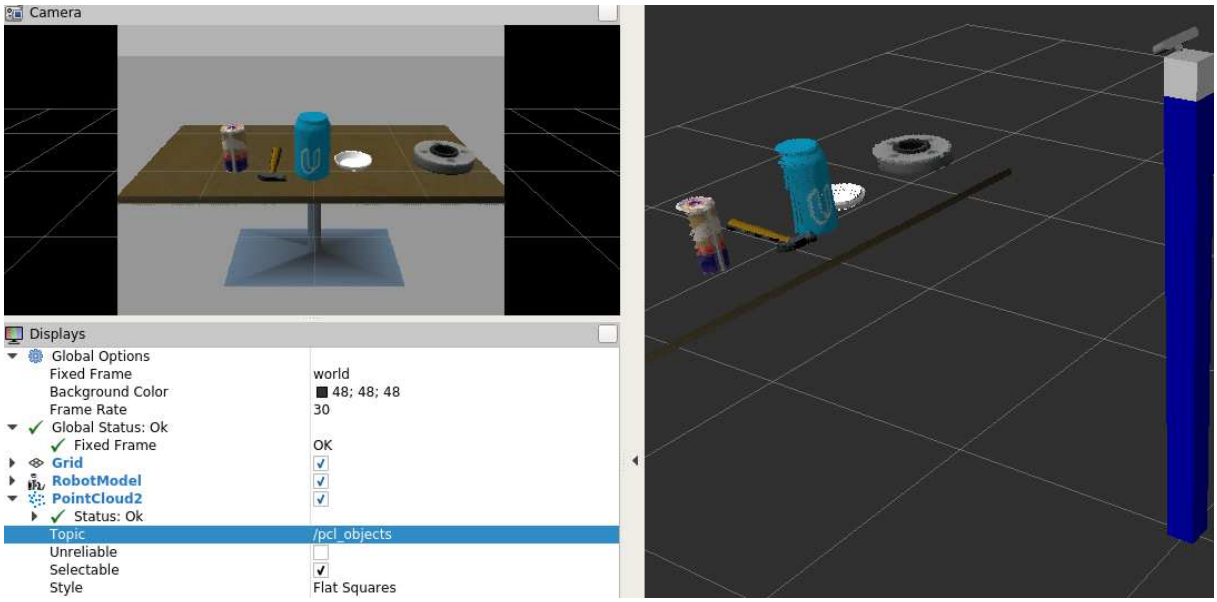
Visualization in Gazebo and RViz

$ roslaunch sensor_stick robot_spawn.launch
$ cd ~/catkin_ws/src/sensor_stick/scripts/

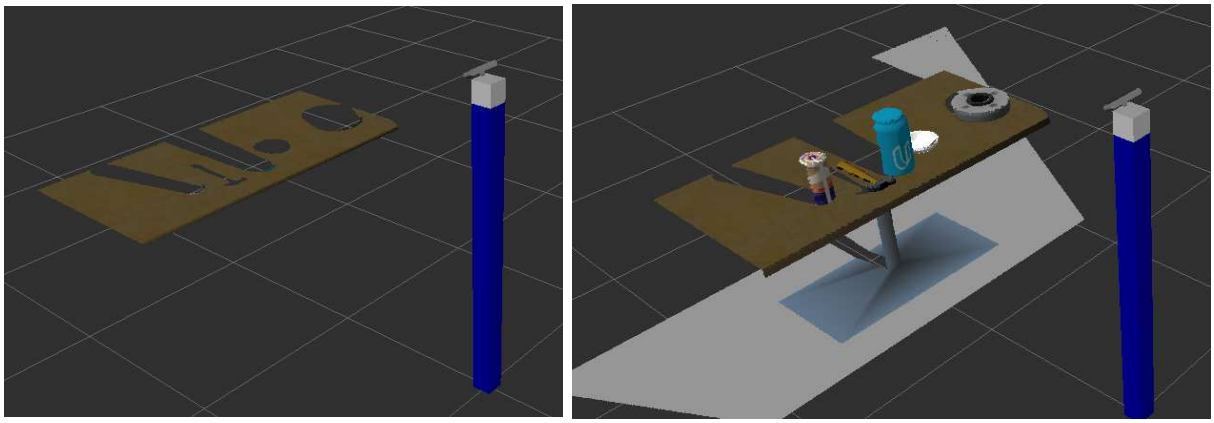
$ ./segmentation.py

Separation of the Objects

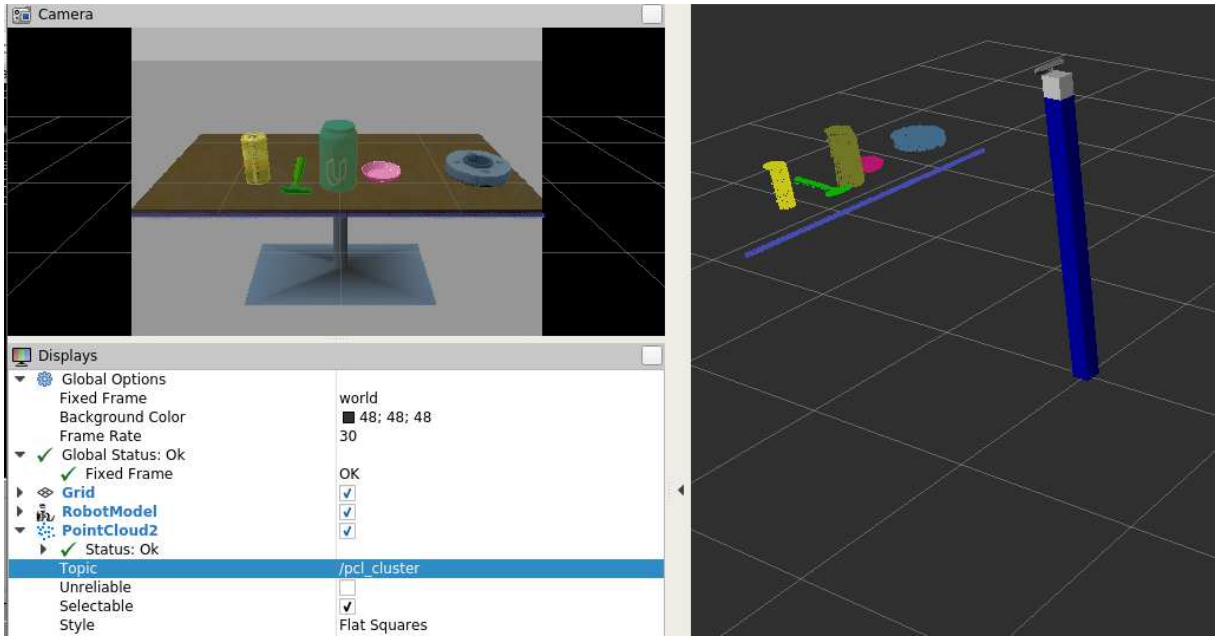




Separation of the Table



Clustering

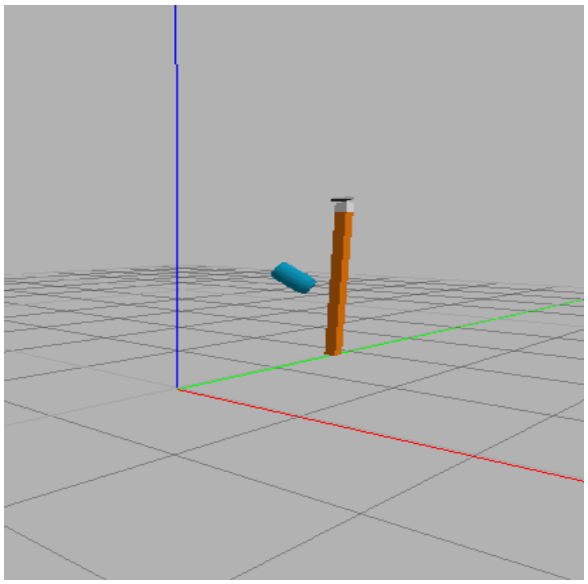


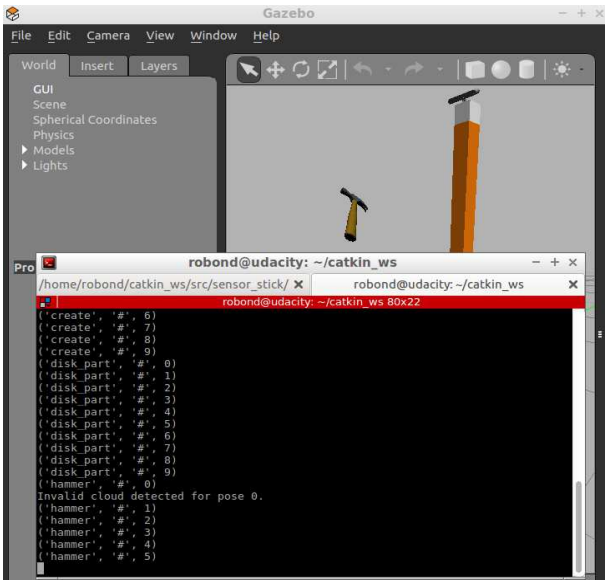
3. Object recognition

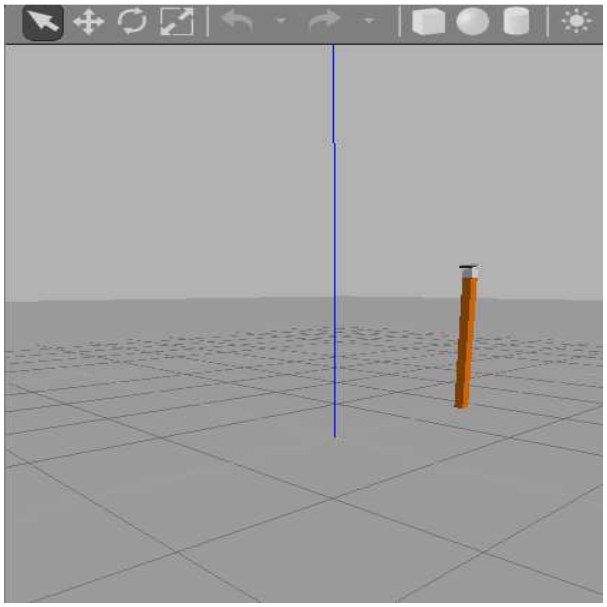
First I generated the features using the ***training.launch***. Next, in a new terminal, I have run the ***capture_features.py*** script to capture and save features for each of the objects in the environment. This script spawns each object in random orientations (I have used 10 orientations per object) and computes features based on the point clouds resulting from each of the random orientations.

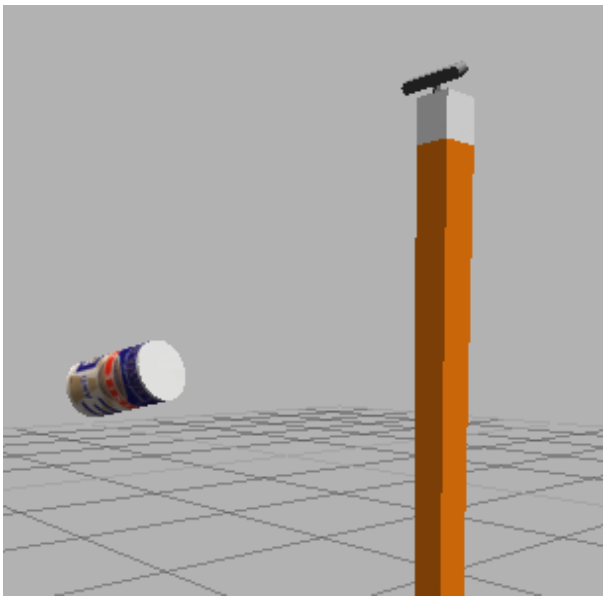
Visualization:

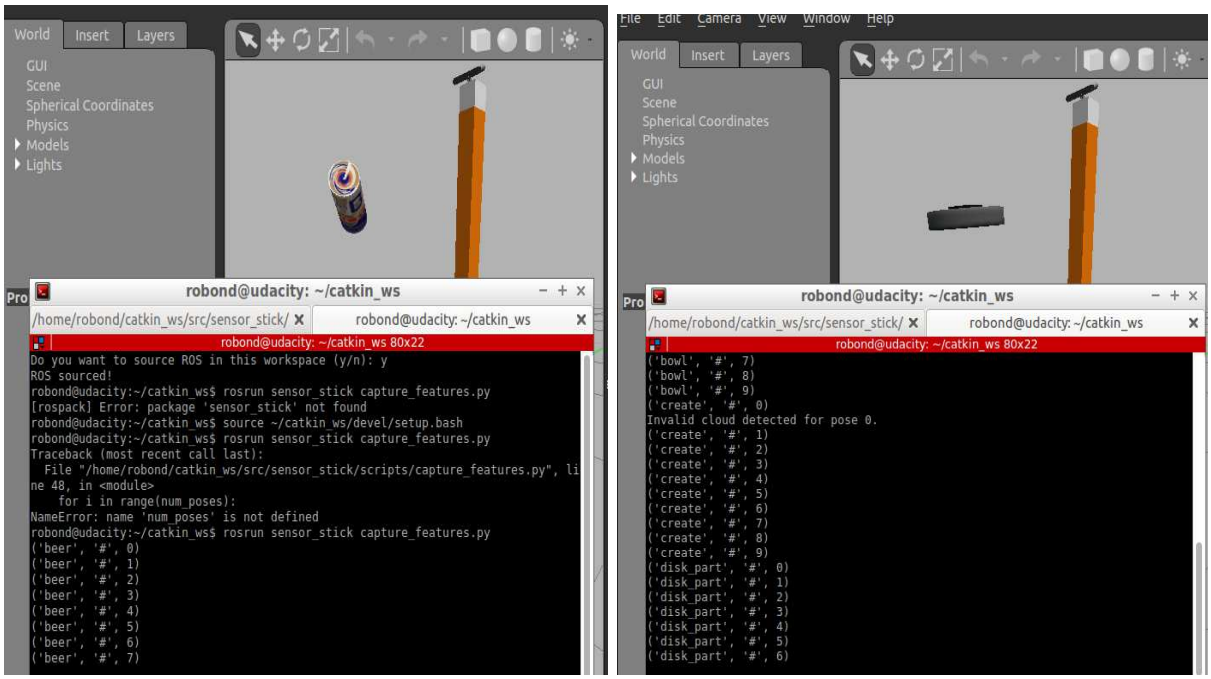
$ rosrun sensor_stick capture_features.py



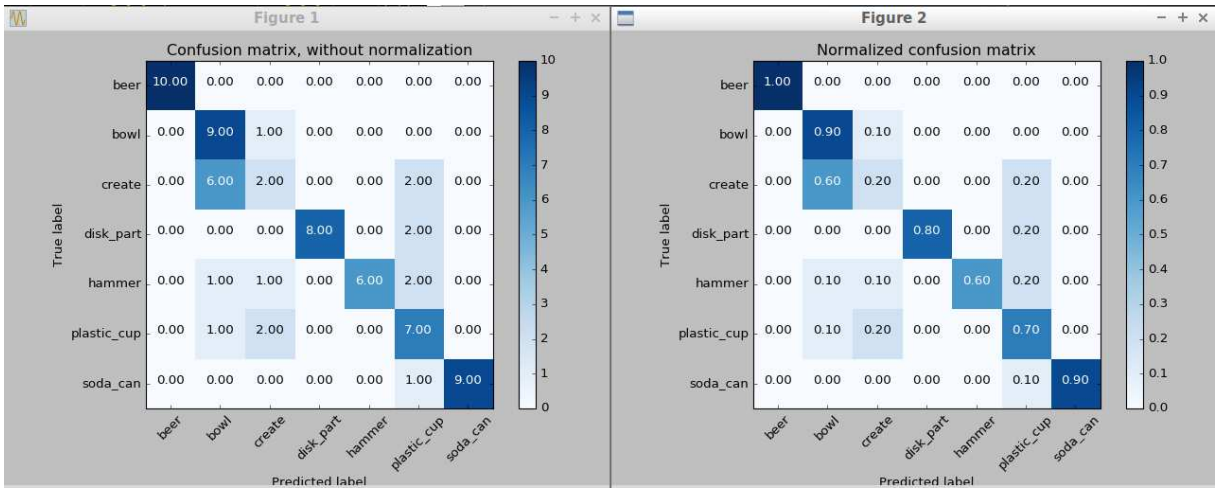






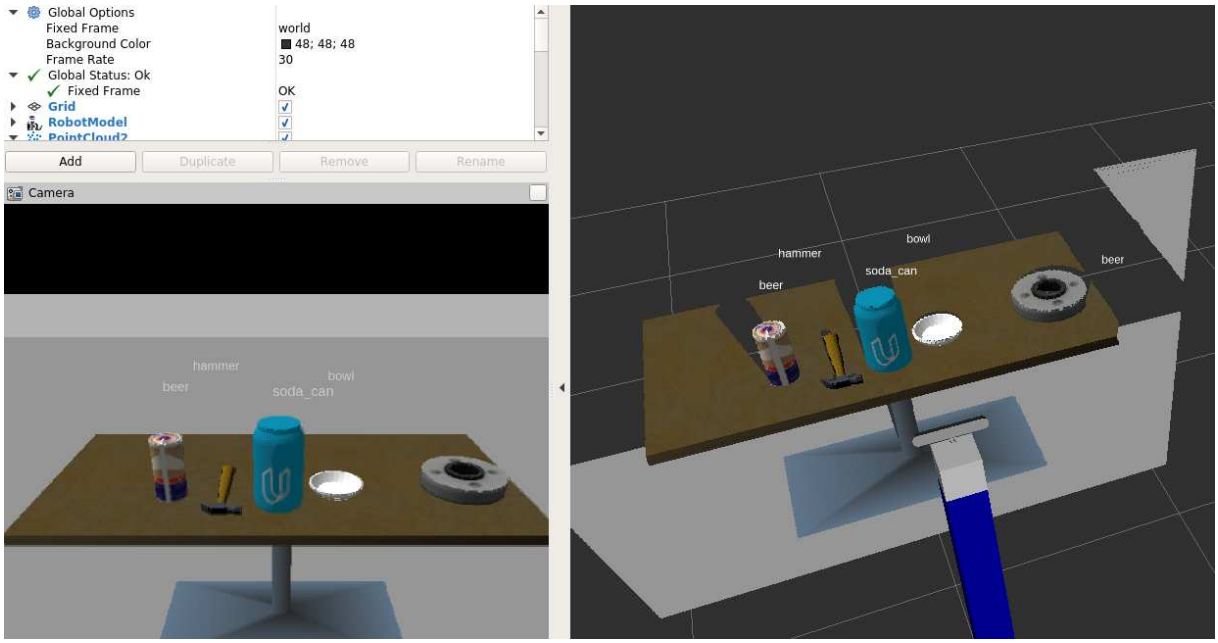


Now I have a ***training_set.sav*** file containing the features and labels. By running the train_svm.py script I could train an SVM classifier with the defined features receiving the confusion matrix, which was improved by computing color and normal histograms. See features.py code. The resulting file ***model.sav*** was used for the recognition task.



Visualization with object recognition feature:

$ roslaunch sensor_stick robot_spawn.launch
$ chmod +x object_recognition.py
$ ./object_recognition.py



The result was not yet perfect. The code was improved in the following 3 D Perception Project.

Project 3D Perception

Pick and Place Setup

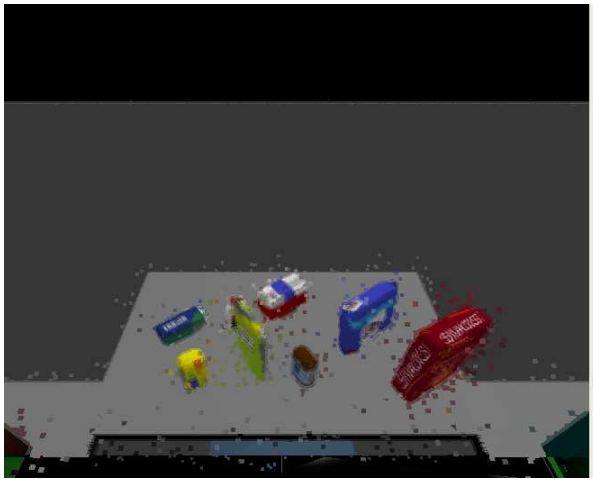
For all three tabletop setups (test.world), perform object recognition, then read in respective pick list (pick_list_*.yaml). Next construct the messages that would comprise a valid PickPlace request output them to .yamlformat.*

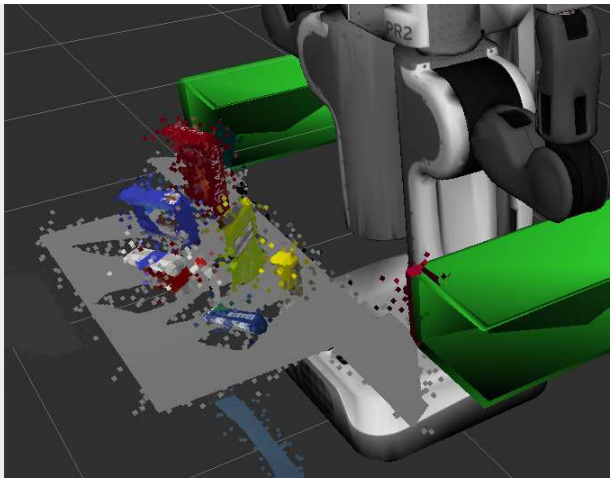
You can add this functionality to your already existing ros node or create a new node that communicates with your perception pipeline to perform sequential object recognition. Save your PickPlace requests into output_1.yaml, output_2.yaml, and output_3.yaml for each scene respectively. Add screenshots in your writeup of output showing label markers in RViz to demonstrate your object recognition success rate in each of the three scenarios.

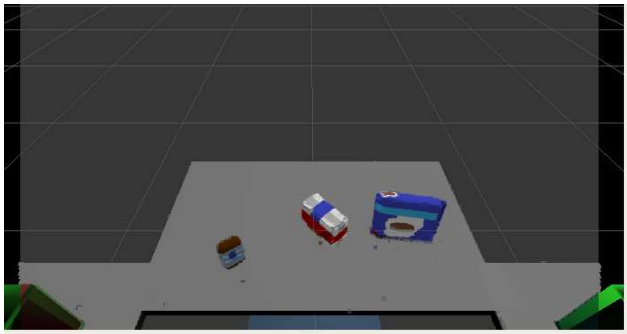
Note: for a passing submission, your pipeline must correctly identify 100% of objects in test1.world, 80% (4/5) in test2.world and 75% (6/8) in test3.world.

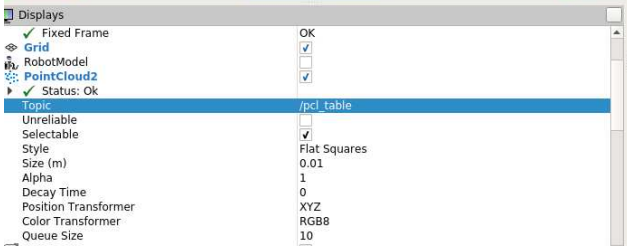
For the Project code I have used the ***project_template.py*** with the important TODOs.

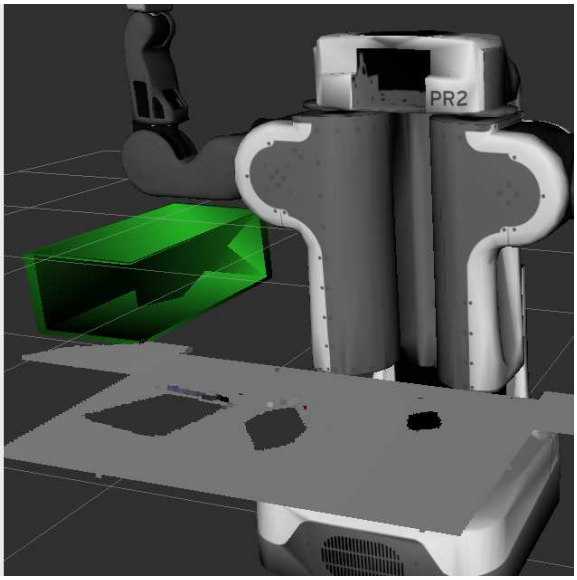
In the first part I applied the exercises 1-3 to filter the environment, perform RANSAC plane fitting to segment the table in the scene and have used the the Euclidean Clustering technique to separate the objects into distinct clusters for the segmentation process.

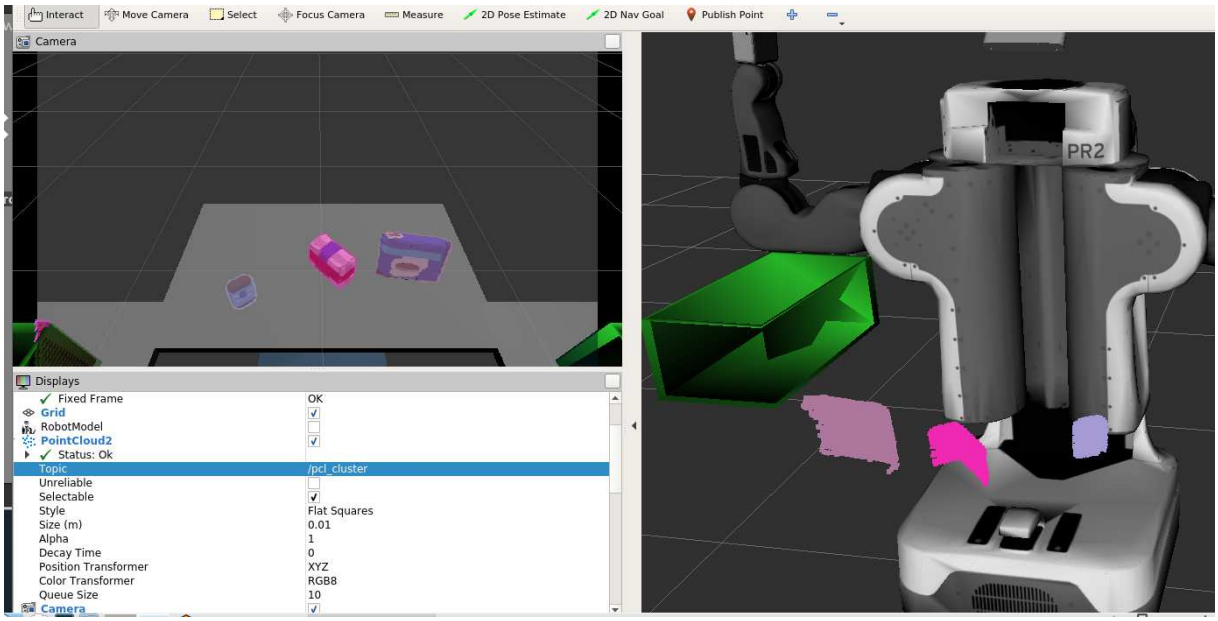


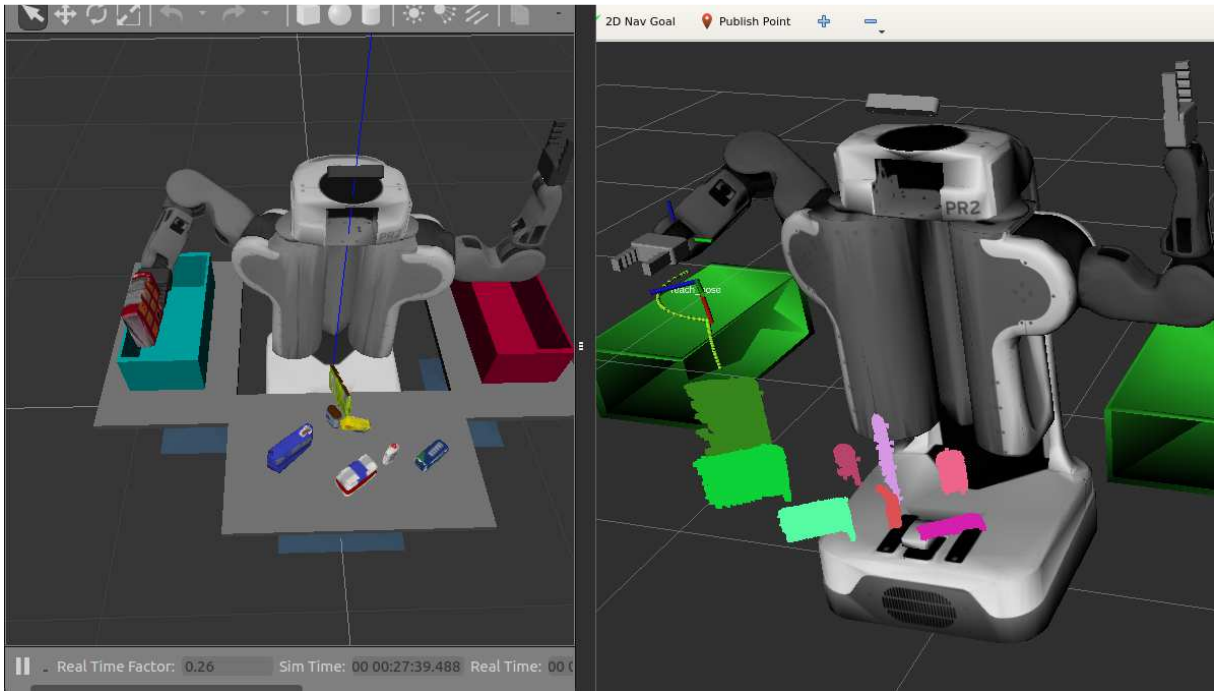




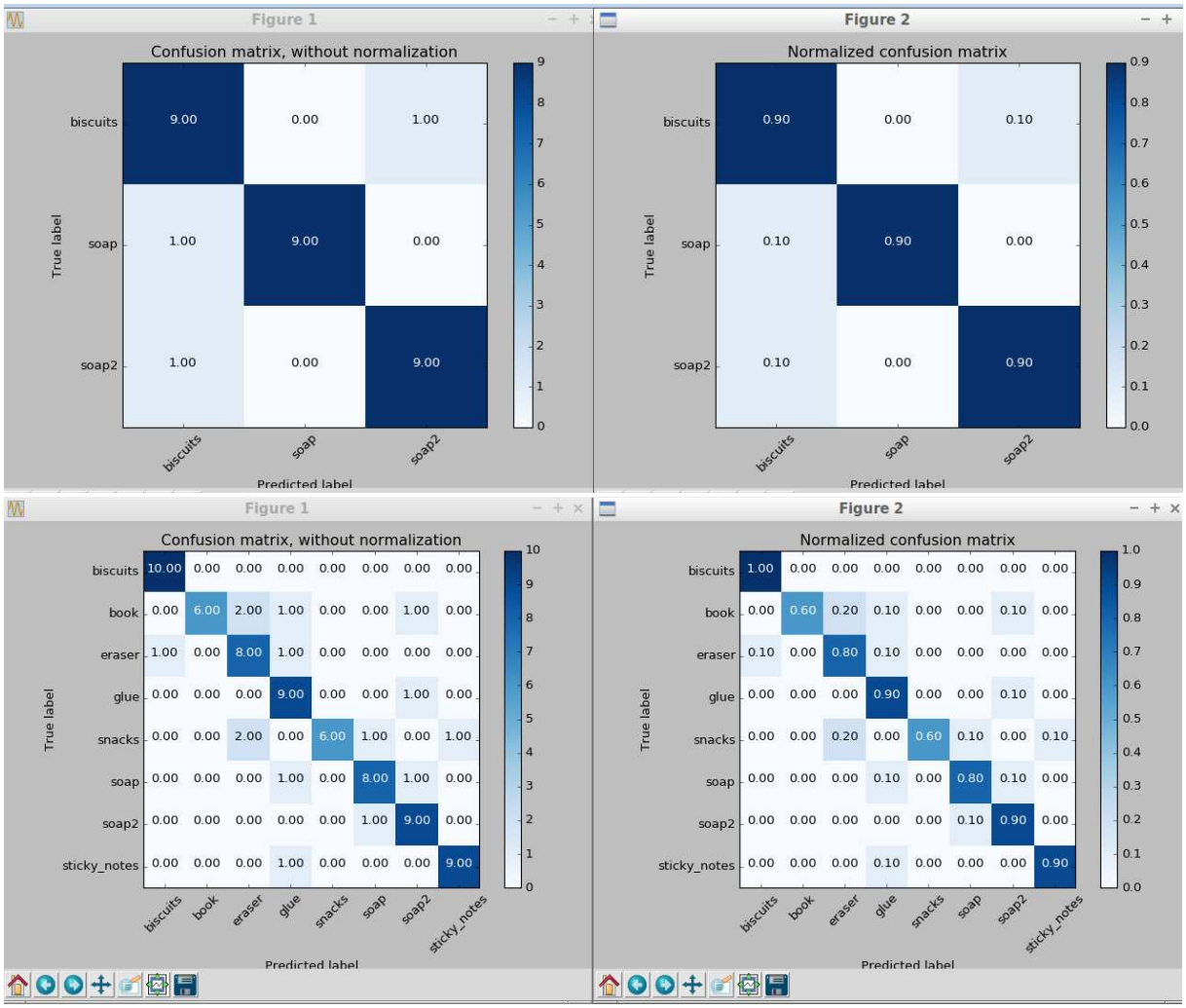








I have trained the SVM classifier with features from the new models list from 3 pick_lists (pick_list_*.yaml). See also ***capture_features.py***. I have added my object recognition code to my perception pipeline and tested the actual project scene.

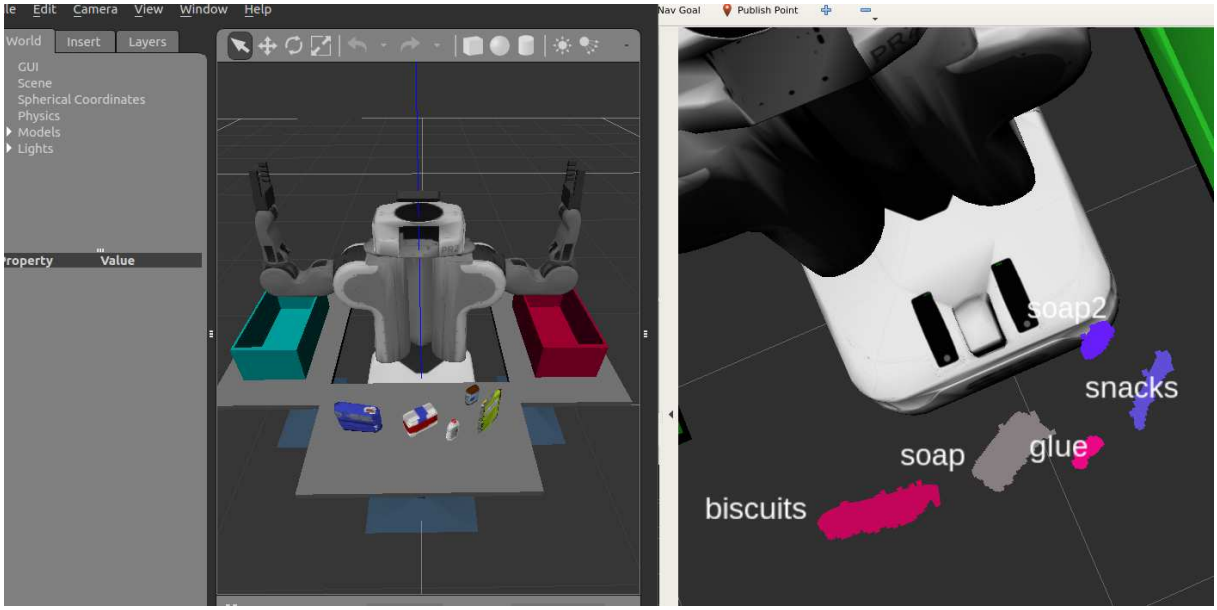


The test was completed 3 time with 3 different lists in 3 test worlds correspondently.

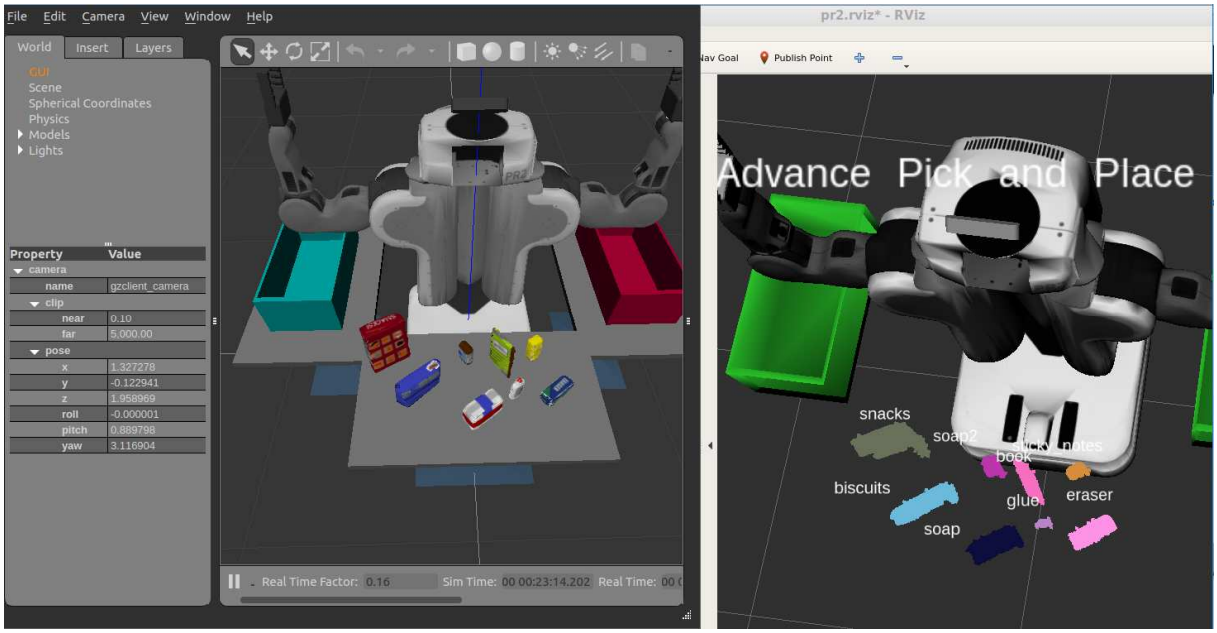
See **project_template_g.py**

Test 1

Test 2


Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property Value
Nav Goal Publish Point
soap2
snacks
soap
glue
biscuits

Test 3

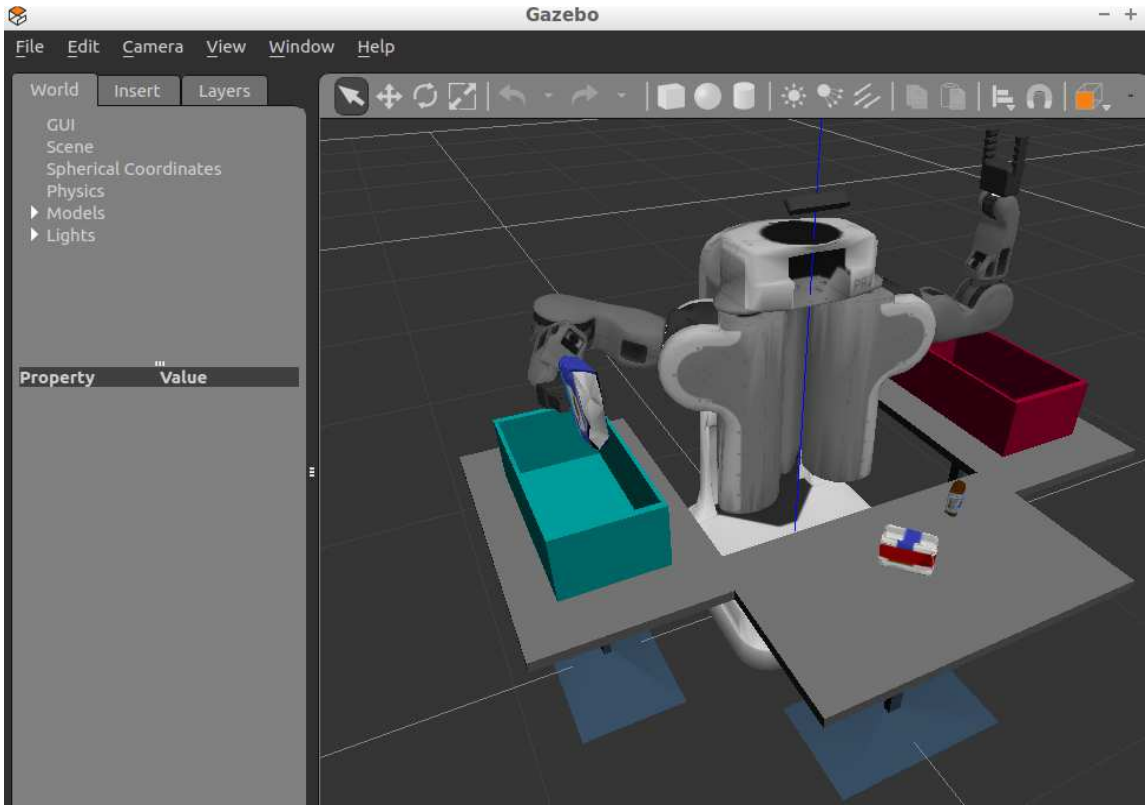

File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property Value
camera
name gzclient_camera
clip
near 0.10
far 5,000.00
pose
x 1.327278
y -0.122941
z 1.958969
roll -0.000001
pitch 0.889798
yaw 3.116904
Real Time Factor: 0.16 Sim Time: 00 00:23:14.202 Real Time:
pr2.rviz* - RViz
Nav Goal Publish Point
Advance Pick and Place
snacks
soap2
book
notes
biscuits
glue
eraser
soap

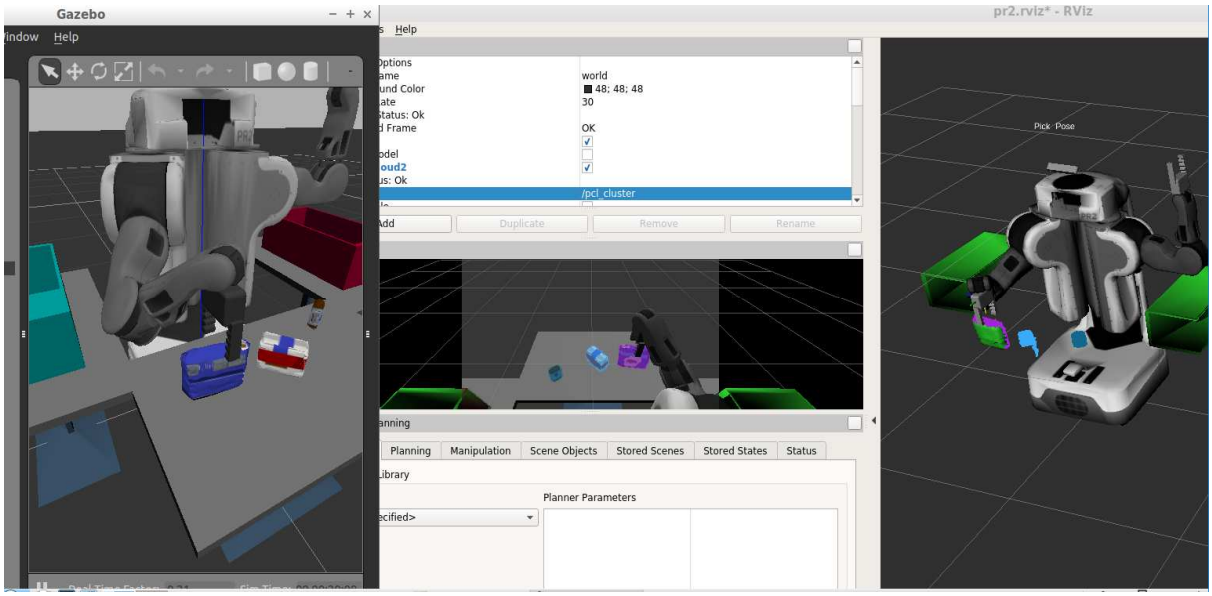
The Output files ***output_1.yaml***, ***output_2.yaml***, ***and output_3.yaml*** contain:

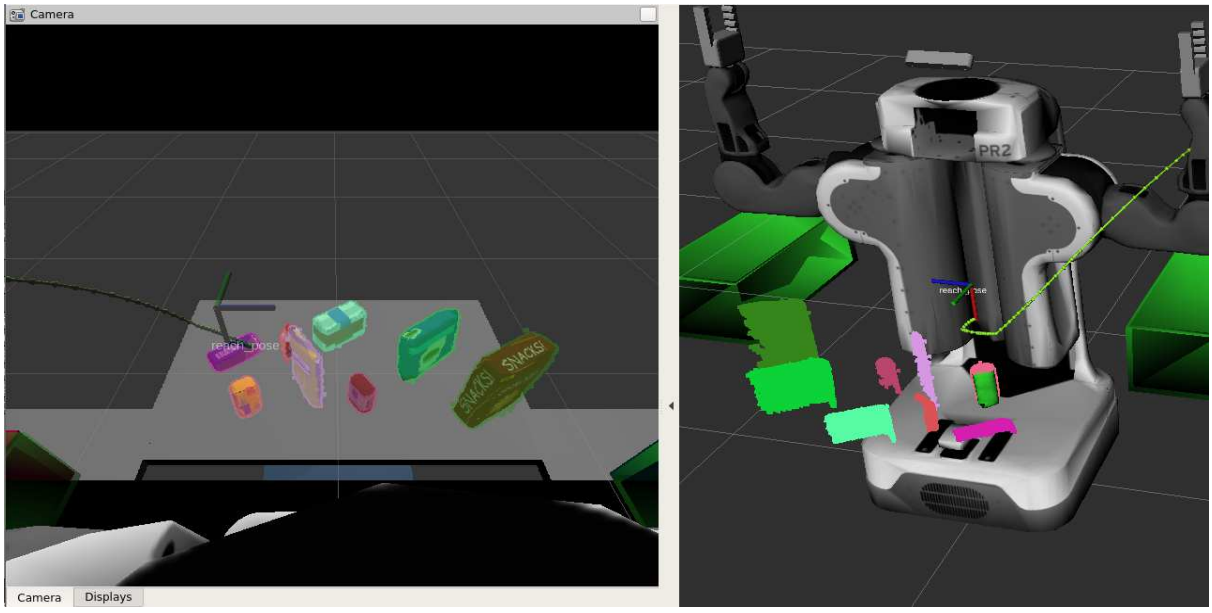
- Object Name
- Arm Name
- Pick Pose (Centroid of the recognized object)
- Place Pose
- Test Set Number

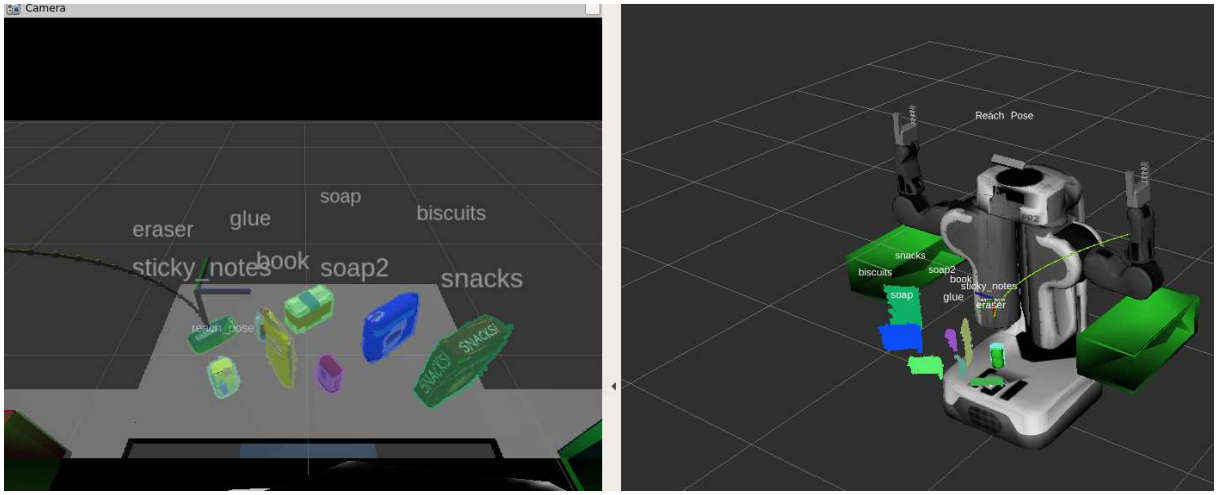
100% recognition for all 3 test runs.

Calling ***'pick_place_routine'*** service robot was performing the pick and place tasks, placing the objects in different red and green dropboxes.




Gazebo
pr2.rviz* - RViz
Help
world
48; 48; 48
30
OK
/pcl_cluster
Duplicate
Remove
Rename
Planning
Manipulation
Scene Objects
Stored Scenes
Stored States
Status
Planner Parameters
Pick Pose


Camera
reach_pose
Camera
Displays
PR2


Camera
soap
eraser
glue
biscuits
sticky_notes
book
soap2
snacks
reach_pose
Reach Pose
snacks
biscuits
soap2
book
sticky_notes
soap
glue
eraser