Contents

Table of Figures

Exercise 1

Followed these steps for setting up and implementing Exercise -1.

- **Step-1.** Download or clone the repository in the provided course VM and follow the setup instructions.
  It is shared in the below path in shared project folder
  RoboND-Perception-Project\RoboND-Perception-Exercises\Exercise-1

- **Step-2.** Experiment with the various filters in the upcoming sections.
  I Created a voxel grid filter object with LEAF_SIZE = 0.01 and applied to the point cloud
  Voxel_downsampled.pcd file looks like this in pcl viewer:

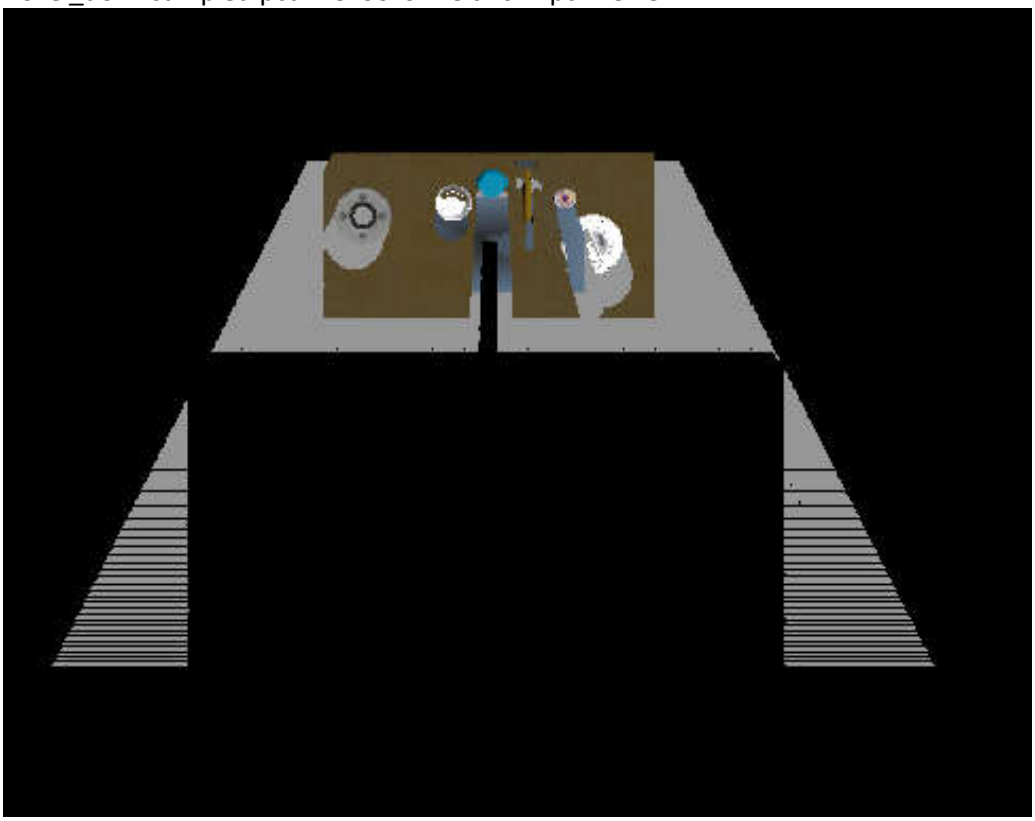

Figure 1: Voxel downsampled pcd

I created a pass-through filter along z axis to remove all the noise data except the tabletop and the objects. For this, the axis_min and axis_max values are chosen to 0.6 and 1.1 respectively, after few iterations. This pass-through filter is applied to the voxel downsampled point cloud data. Filter data looks like this when viewed in pcl viewer:

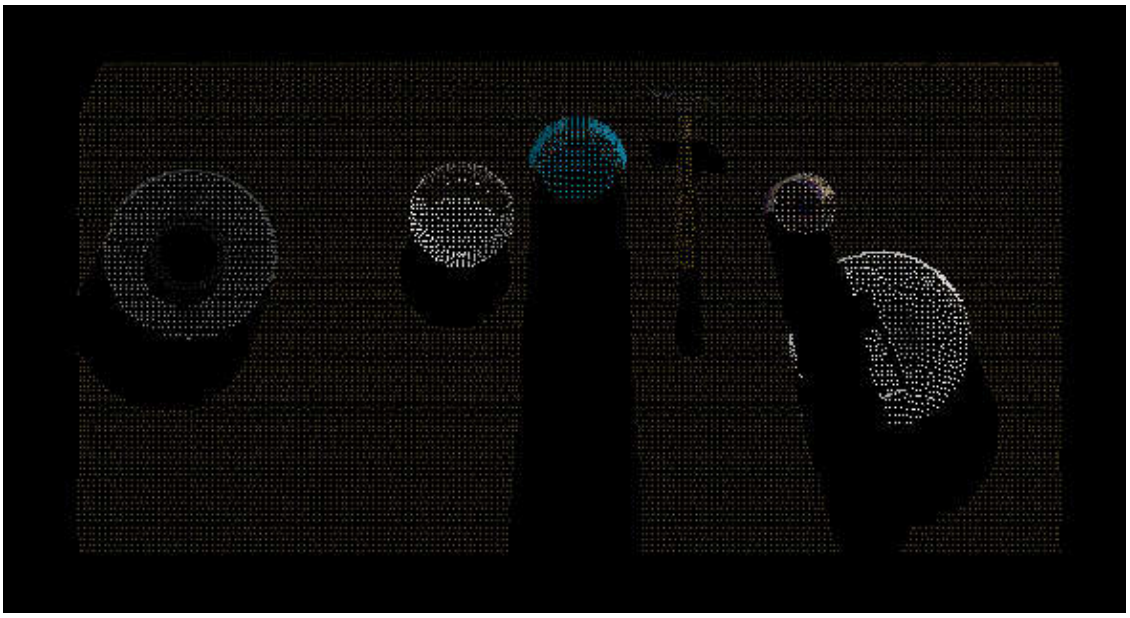

Figure 2: Pass through filtered pcl

- **Step-3.** Perform RANSAC plane fitting to segment the table in the scene
  In order to remove the table from scene, I applied RANSAC plane fitting model because all the table point cloud data can fit into plane model. This gives the set of inliers. All other pint cloud data that doesn't fit the model constitutes the set of outliers. In this scenario, all table data makes the inlier set while all the objects placed on the tabletop makes the outlier sets. Many values for the max_distance (Max distance for a point to be considered fitting the model) were tried and the vale of 0.01 seems to work best in this case.

- **Step-4.** Save two new point cloud files containing only the table and the objects, respectively.
  In the repository, you are provided with Point Cloud data for the simulated tabletop scene in the form of a single file: tabletop.pcd. In the end, your goal is to create two separate Point Cloud files:
- Table.pcd - Containing only the points that belong to the table
- Objects.pcd - Containing all the tabletop objects
  Table.pcd is the set of inlier points and is obtained by making the negative flag to be True while calling the RANSAC model. Extracted_inlier.pcd looks like this when viewed in pcl viewer.

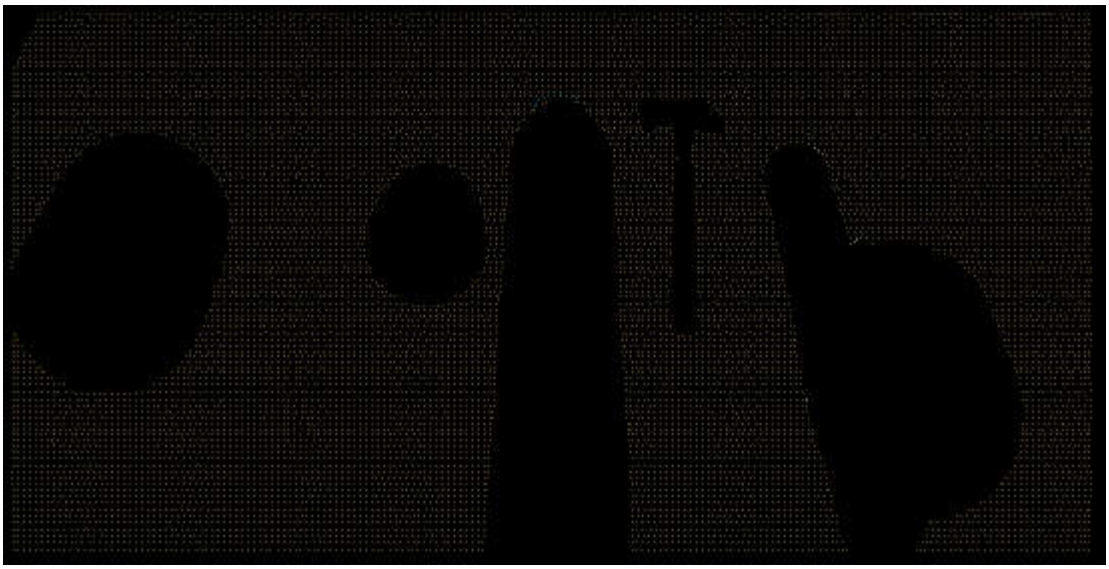

Figure 3: Extracted inlier pcd : Table data

Objects.pcd is the set of outlier points and is obtained by making the negative flag to be False. It contains all the tabletop objects. Extracted_outlier looks like this when viewed in pcl viewer.

Figure 4: Extracted outlier pcd: Tabletop objects

Exercise 2

Followed following steps for setting up and implementing Exercise 2

A. Copy or move the `/sensor_stick` directory and all of its contents to `~/catkin_ws/src`.

```
cp -r ~/RoboND-Perception-Exercises/Exercise-2/sensor_stick ~/catkin_ws/src/
```

B. Next, use `rosdep` to grab all the dependencies you need to run this exercise.

```
$ cd ~/catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y
```

C. Run catkin make to build all the libraries

```
$ catkin_make
```

D. Add the following lines to your .bashrc file

```
export GAZEBO_MODEL_PATH=~/catkin_ws/src/sensor_stick/models
source ~/catkin_ws/devel/setup.bash
```

E. Now, you should be all setup to launch the environment! Run the following command to launch the scene in Gazebo and RViz:

```
roslaunch sensor_stick robot_spawn.launch
```

Your RViz window should look like this:

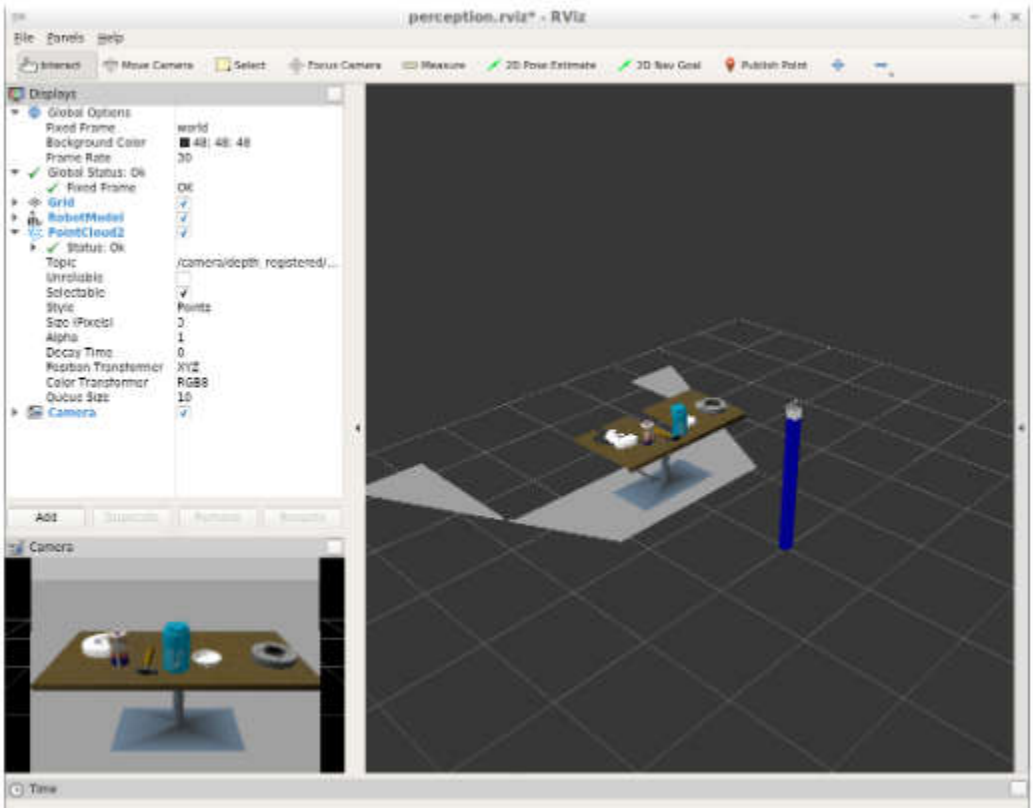

Figure 5:Rviz environment with excercises

Ultimately in this exercise, our goal is to write a ROS node that takes in the camera data as a point cloud, filters that point cloud, then segments the individual objects using Euclidean clustering. In this step, you'll begin writing the node by adding the code to publish your point cloud data as a message on a topic called `/sensor_stick/point_cloud`.
The code is implemented in segmentation.py file which lies In the `RoboND-Perception-Exercises/Exercise-2/sensor_stick/scripts`.

I implemented Euclidian clustering as mentioned in the classroom after doing voxel down sampling, pass through filter and RANSAC filter. The tolerance, minimum number of points and maximum number o f points for a Euclidean cluster is chosen to be 0.02,100 an d50000 respectively.

Finally, the extracted_outliers data (objects set), extracted_inliers data (table) and the clustered data is converted to ros message and published to the respective publishers.

```
# TODO: Convert PCL data to ROS messages
ros_cloud_objects =pcl_to_ros(extracted_outliers)
ros_cloud_table = pcl_to_ros(extracted_inliers)
ros_cloud_cluster = 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_cloud_cluster)
```

Visualizing the Results in Rviz

To see the results of the segmentation in RViz, we need to create a new publisher (/pcl_cluster))and publish our `ros_cluster_cloud` to it

We can see the cluster cloud data in Rviz by simply changing the topic dropdown for the `PointCloud2` display from `/sensor_stick/point_cloud` to `/pcl_cluster`. We can see the clusters in pcl viewer as shown below:

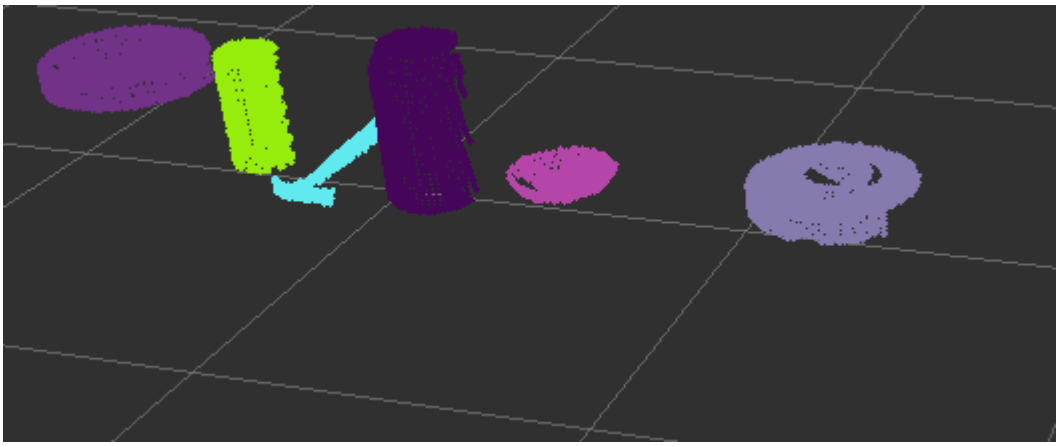

Figure 6 :Cluster cloud after segmentation of objects in the excercises

Exercise 3

Getting set up:

- If you completed Exercises 1 and 2 you will already have a `sensor_stick` folder in your `~/catkin_ws/src` directory. You should first make a copy of your Python script (`segmentation.py`) that you wrote for Exercise-2 and then replace your old `sensor_stick` folder with the new `sensor_stick` folder contained in the Exercise-3 directory from the repository.
- If you do not already have a `sensor_stick` directory, first copy / move the `sensor_stick` folder to the `~/catkin_ws/src` directory of your active ros workspace. From inside the `Exercise-3` directory:

```
cp -r sensor_stick/ ~/catkin_ws/src/
```

- Make sure you have all the dependencies resolved by using the `rosdep install` tool and running `catkin_make`:

```
$ cd ~/catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y
$ catkin_make
```

- If they aren't already in there, add the following lines to your `.bashrc` file

```
$ export GAZEBO_MODEL_PATH=~/catkin_ws/src/sensor_stick/models
$ source ~/catkin_ws/devel/setup.bash
```

You're now all setup to start the exercise!

Generating Features

To get started generating features, launch the `training.launch` file to bring up the Gazebo environment. An empty environment should appear with only the sensor stick robot in the scene:

```
$ cd ~/catkin_ws
$ roslaunch sensor_stick training.launch
```

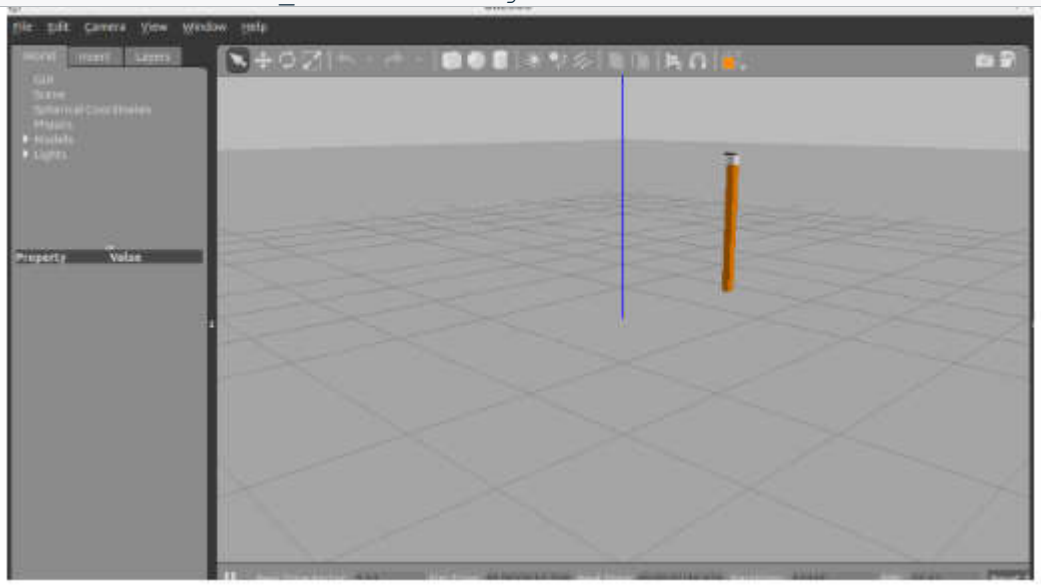

Figure 7:Sensor stick training in Gazebo

Capturing Features

Next, in a new terminal, 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 (default 5 orientations per object but changed it to 10) and computes features based on the point clouds resulting from each of the random orientations.

```
$ cd ~/catkin_ws
$ rosrun sensor_stick capture_features.py
```

The features will now be captured and you can watch the objects being spawned in Gazebo. It should take 5-10 seconds for each random orientation (depending on your machine's resources). There are 7 objects total so it takes a while to complete. When it finishes running you should have a `training_set.sav` file containing the features and labels for the dataset. **Note:** The `training_set.sav` file will be saved in your `catkin_ws` folder.

Train Your SVM

Once your feature extraction has successfully completed, you're ready to train your model. First, however, if you don't already have them, you'll need to install the `sklearn` and `scipy` Python packages. You can install these using `pip`:

```
$ pip install sklearn scipy
```

After that, you're ready to run the `train_svm.py` script to train an SVM classifier on your labeled set of features.

```
$ rosrun sensor_stick train_svm.py
```

When this runs, you'll get some text output at the terminal regarding overall accuracy of your classifier and two plots will pop up showing the relative accuracy of your classifier for the various objects:

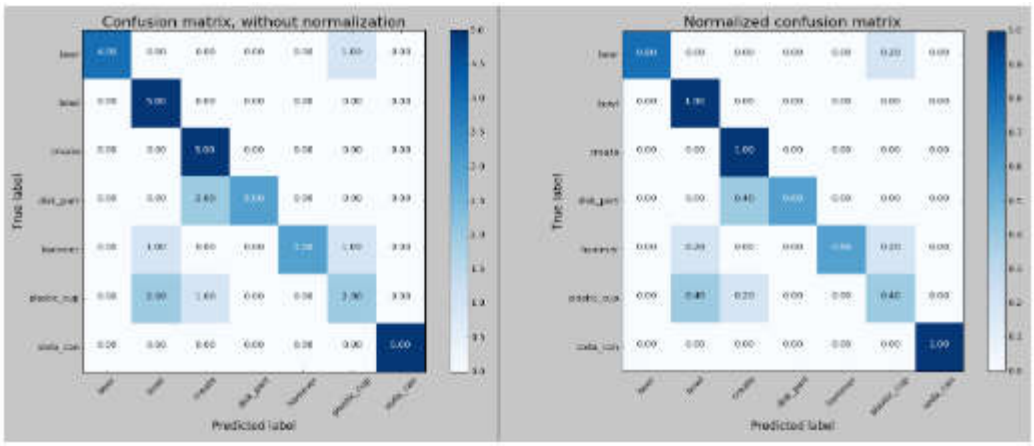

Figure 8: Confusion matrix without filling the features.py

These plots are showing two different versions of the confusion matrix for your classifier. On the left is raw counts and on the right as a percentage of the total.
You may have noticed that the confusion matrices looks highly confused! This is because we haven't actually generated meaningful features yet. To get better features, I open up the `features.py` script in RoboND-Perception-Project`/sensor_stick/src/sensor_stick/` (this might seem like a weird directory structure but this is the preferred ROS way of setting up internal Python packages). In this script you'll find two functions called `compute_color_histograms ()` and `compute_normal_histograms()`.

Improvements:

1. Filled the compute_color_histograms() and compute_normal_histograms() function s
2. Updated HSV= True in captures_features.py file
3. Updated Kernel to 'linear'

Updated Confusion Matrix looks like this:

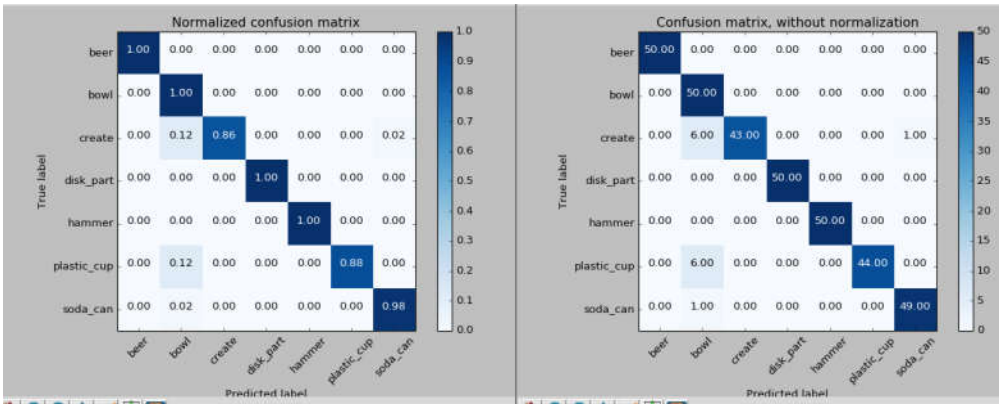

Figure 9:Updated Confusion matrix with computed histograms

Features in Training Set: 140

Invalid Features in Training set: 0

Scores: [0.82142857 0.85714286 0.92857143 0.85714286 0.82142857]

Accuracy: 0.86 (+/- 0.08)

accuracy score: 0.857142857143

Object Recognition

We now have a trained classifier and we're ready to do object recognition!

Followed following instructions:

you can shut down your training environment and bring up the tabletop scene again:

```
$ roslaunch sensor_stick robot_spawn.launch
```

In another terminal, run your object recognition node (in `/scripts` if that's where you're working but keep in mind your `model.sav` file needs to be in the same directory where you run this!):

```
$ chmod +x object_recognition.py
$ ./object_recognition.py
```

And you should now see markers in RViz associated with each segmented object as shown below:

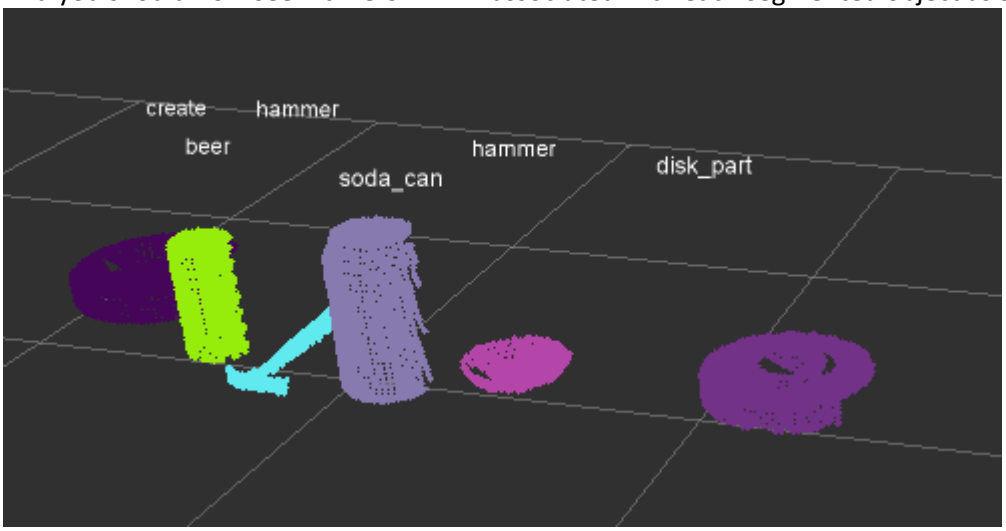

Figure 10:Object recognition with only 20 feature set

As you can see there are errors in this detection as we are getting two hammers. I changed the captures_features.py file to include a greater number of features. After running the for loop for 100 times, we got an almost 98.75% accurate model which could detect all the objects correctly.

3D perception Project

Pick and Place Setup

1. 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.

Perception Pipeline

Creating the perception pipeline

Essentially, there are three different worlds or scenarios that we are going to work with where each scenario has different items on the table in front of the robot. These worlds are located in the `/pr2_robot/worlds/` folder, namely the `test_*. world` files.

Test_world :1

Started out by creating a ROS node just like we did in the exercises. All our code from Exercise-3 is moved to object_recoginition.py and project_template.py located in pr2_robot/scripts/ directory.
First, we changed the subscriber to subscribe to the camera data (point cloud) topic `/pr2/world/points`.

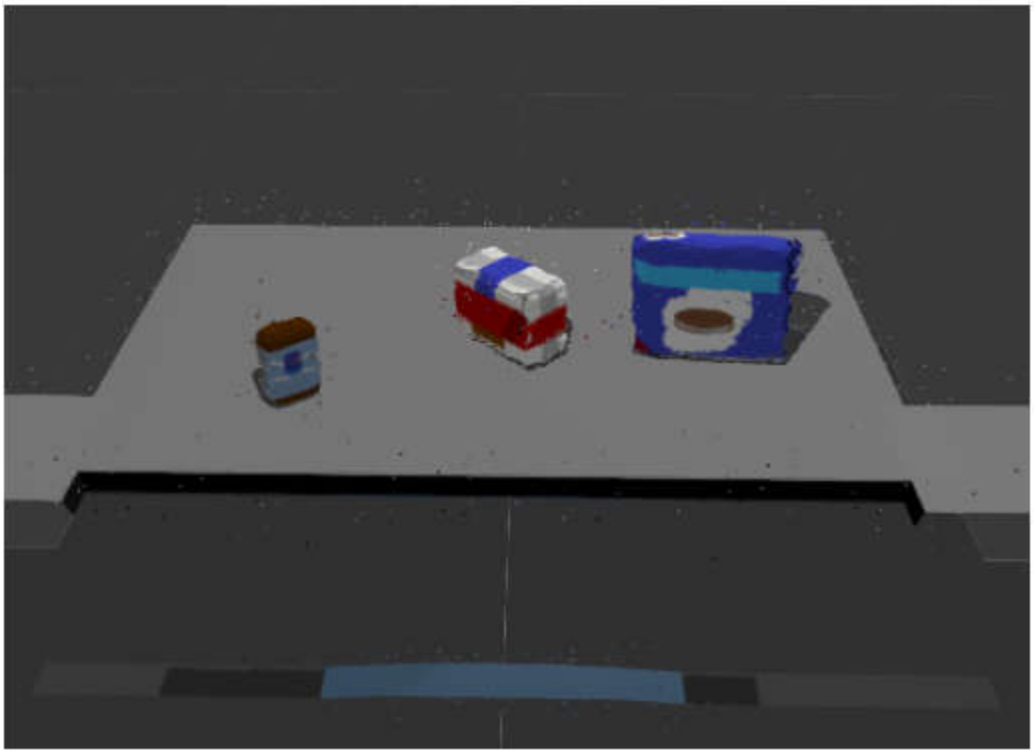

Figure 11:Actual noisy point cloud data

Filtering

Once we have our camera data, start out by applying various filters we have looked at so far. We need to tweak the parameters to accommodate this new environment. Also, this new dataset contains noise! To clean it up, the first filter we should apply is the statistical outlier filter.

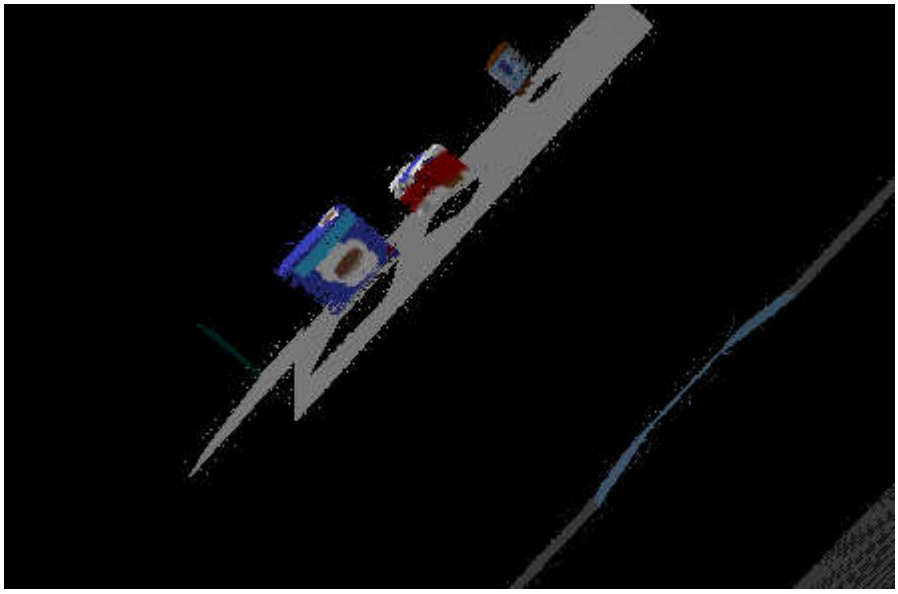

Figure 12:Filtered data after applying statistical outlier filter.

Considering the noisy data, we should also apply pass filter first along z-axis and then along y-axis. Min and max values along y-axis is chosen to be -0.5 and 0.5 respectively as all the objects lie along this length of the table.

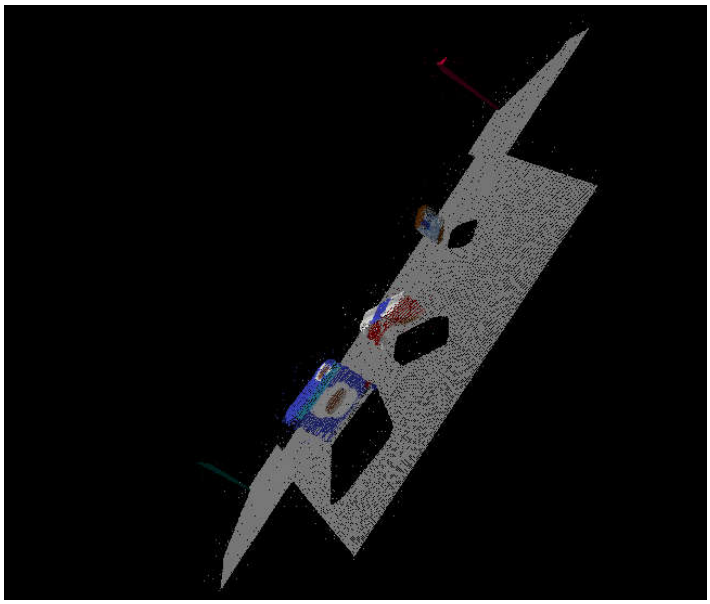

Figure 13:Cloud data after applying pass through filter along z-axis

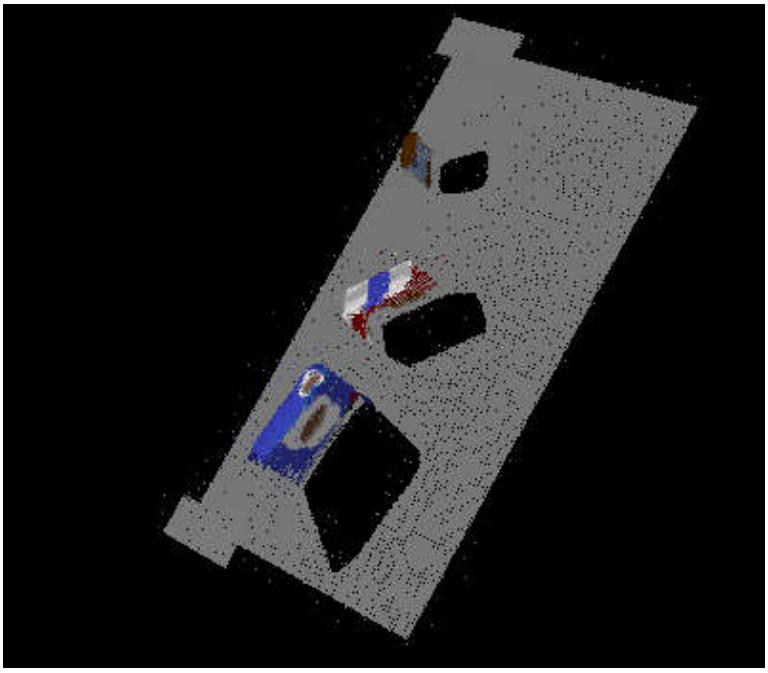

Figure 14:cloud data after applying pass through filter along y-axis

Cloud data after applying pass through filter along y-axis

Table Segmentation

Next, perform RANSAC plane fitting to segment the table in the scene as it was done in Exercise 3. The code block remains same as that of exercise 3. This gives us the outlier (objects on tabletop) and the inliers (table) set of point cloud data.

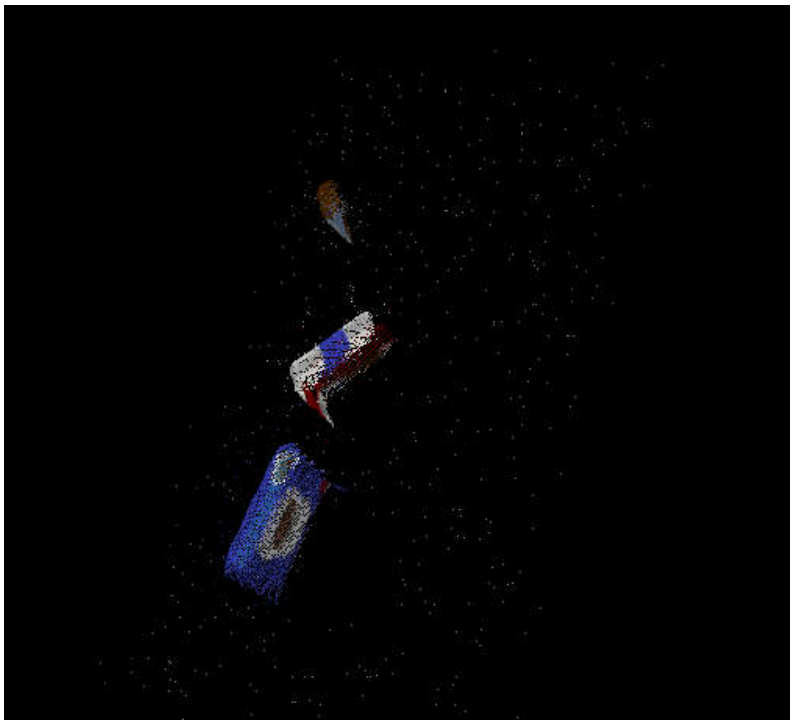

Figure 15:Extracted outlier in test_world_1

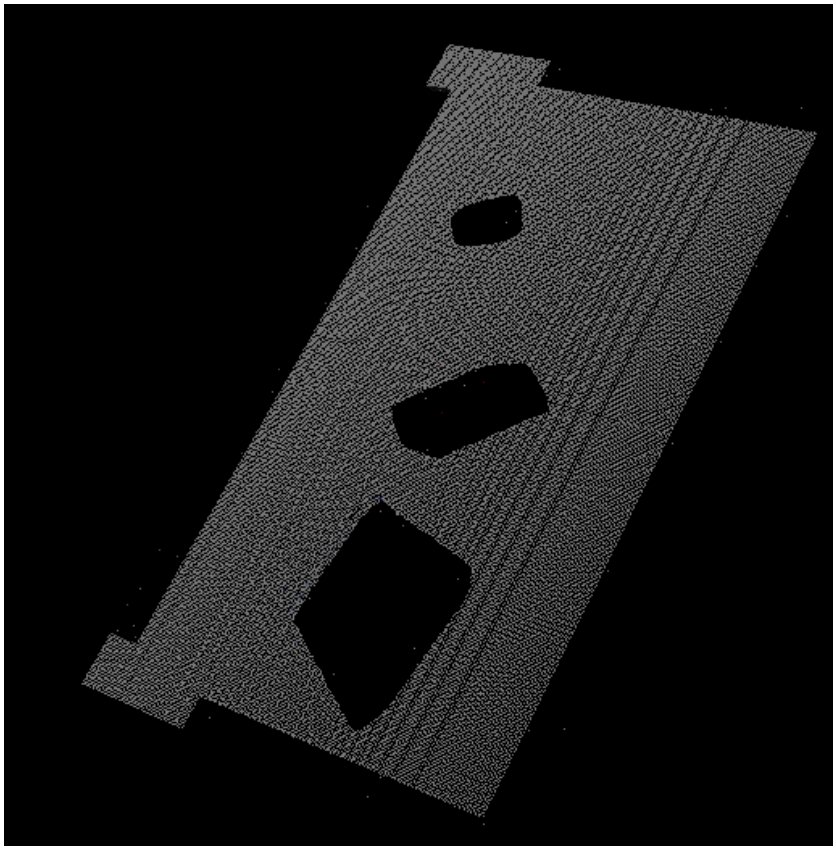

Figure 16:Extracted inlier in test_world 1

Clustering

Use the Euclidean Clustering technique described as in Exercise 3 to separate the objects into distinct clusters, thus completing the segmentation process. As seen in the image below, all the clusters will be represented by different colors.

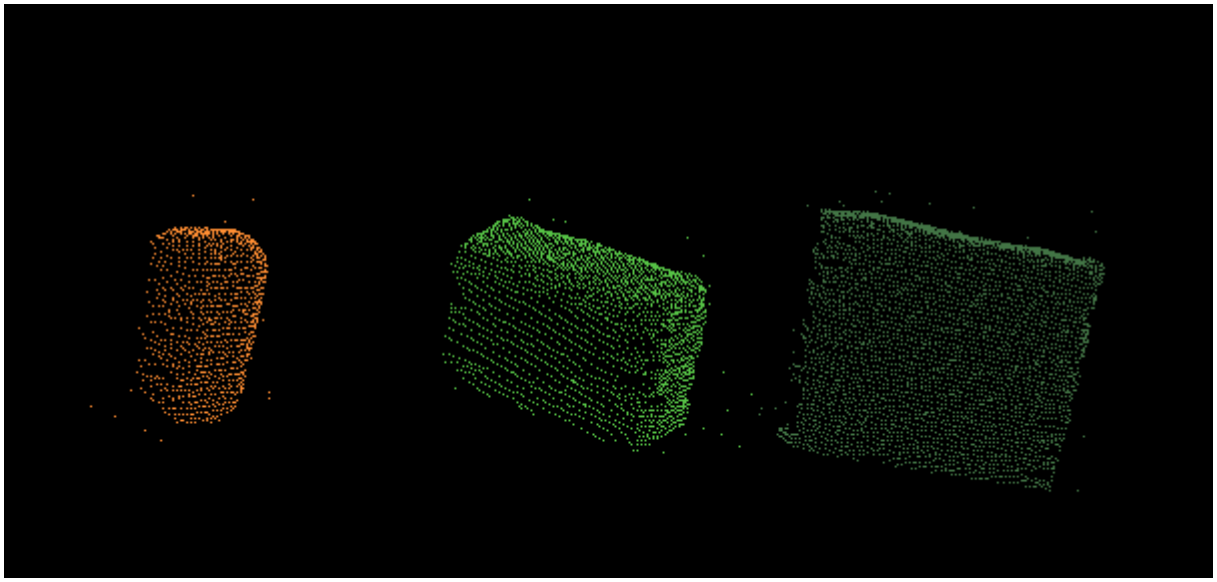

Figure 17:Cluster cloud after Euclidean clustering in test_world 1

Object Recognition

Generate a training set of features for the objects in our pick lists (see the `pick_list_*.yaml` files in `/pr2_robot/config/`). Each pick list corresponds to a world and thus indicates what items will be present in that scenario.

1.To generate the training set, we will have to modify the `models` list in the `capture_features.py` script and run it as you did for Exercise-3:
Model list in capture_features.py is changed to the following for getting training set. This file lies in RoboND-Perception-Project\sensor_stick\scripts in the shared folder.

```
models = [ 'biscuits', ‘soap', 'soap2', ‘book’, ’glue’, ’sticky_notes’, ’eraser’, ’snacks’]
```

2. Train your SVM with features from the new models.

Following changes were done to obtain improved model as suggested in the classroom.

- I tried with various number of feature sets to get the most accurate model. Running the for loop 300 times in captures_features.py file with ‘Linear’ kernel gives the most accurate model for all the three test_world scenarios.

```
for i in range(300):
    # make five attempts to get a valid a point cloud then give up
    sample_was_good = False
    try_count = 0
    while not sample_was_good and try_count < 10:
      sample_cloud = capture_sample()
      sample_cloud_arr = ros_to_pcl(sample_cloud).to_array()
```

- Changed the using_hsv flag to ‘True’ to use hsv feature sets

```
# Extract histogram features
    chists = compute_color_histograms(sample_cloud, using_hsv=True)
    normals = get_normals(sample_cloud)
    nhists = compute_normal_histograms(normals)
    feature = np.concatenate((chists, nhists))
    labeled_features.append([feature, model_name])
```

- Changed the features.py file located in the path below to complete the histogram calculation: RoboND-Perception-Project\sensor_stick\src\sensor_stick . I tried with different bin sizes but bin_size =32 works best for all the three test scenarios.

- Changed the output file name in captures_features.py to training_set_bin32_300pr2.sav.

- Changed the file name in train_svm.py located in RoboND-Perception-Project\sensor_stick\scripts to load the training set and generate the model.

- Tried with different kernels like 'linear','sigmoid' and 'rbf' but the 'linear' kernel worked best for all the test world scenarios. After running the following command, I generated the model named as model_bin32_300pr2_linear.sav

  rosrun sensor_stick train_svm.py

Accuracy results for the above model:

Invalid Features in Training set: 0
Scores: [0.99583333 0.98958333 0.9875 0.98958333 0.98958333]
Accuracy: 0.99 (+/- 0.01)
accuracy score: 0.990416666667

The confusion matrix becomes more accurate with this model. The updated confusion matrix is shown below:

Updated Confusion Matrix:

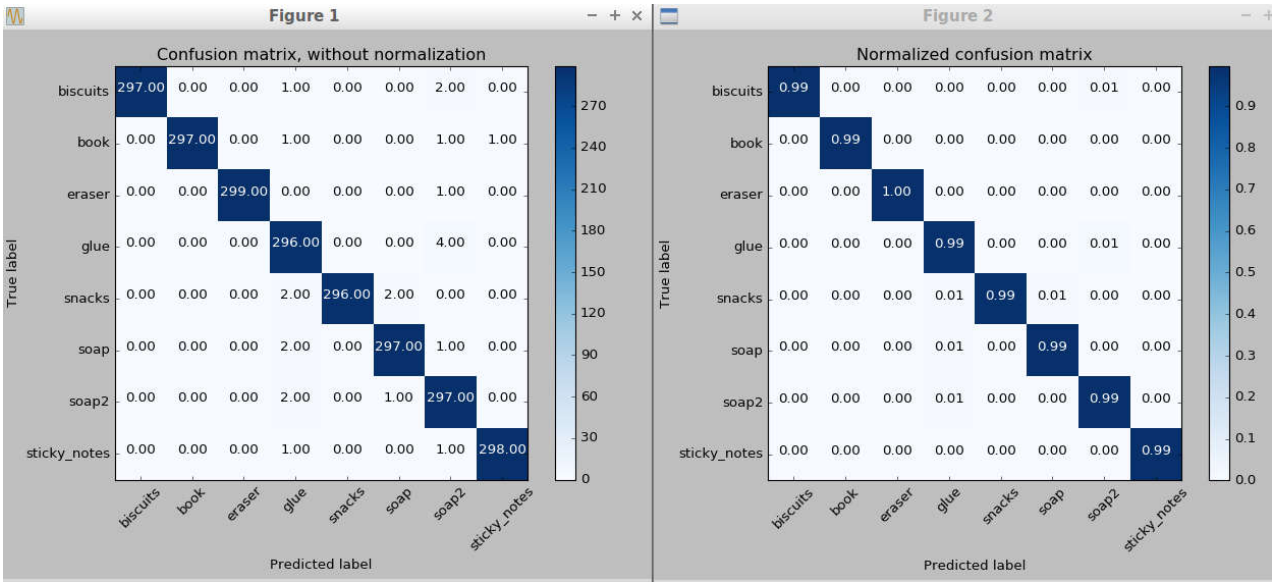

Figure 18:Updated confusion matrix for 3D perception project

- Finally moved the generated model to RoboND-Perception-Project\pr2_robot\scripts folder. Changed the model file name inside the project_template.py and object_recognition.py file too.

3. Added our object recognition code from exercise 3 to our perception pipeline.
4. Tested with the actual project scene to see if our recognition code was successful.
5. To test with the project, follow these steps:
    - Go to the following directory RoboND-Perception-Project\pr2_robot\launch and open the file pick_place_project.launch
    - Change the test_world scenes to 1,2 and 3 for validating all the three test cases one-by-one.

<arg name="world_name" value="$(find pr2_robot)/worlds/test1.world"/>

- Change the pick_list_* to 1,2, and 3 for each of the test cases one-by-one.

<rosparam command="load" file="$(find pr2_robot)/config/pick_list_1.yaml"/>

- Now, launch the pick place project in Gazebo and Rviz environment by running the following command

roslaunch pr2_robot pick_place_project.launch

- Go to RoboND-Perception-Project\pr2_robot\script folder and run the following to perform object recognition

```
rosrun pr2_robot object_recognition.py
```

Cluster cloud images and console messages for each of the three test cases are shown below:

Test1_World:

Console message

Detected 3 objects: ['biscuits', 'soap', 'soap2']

Rviz image for Test1_World:

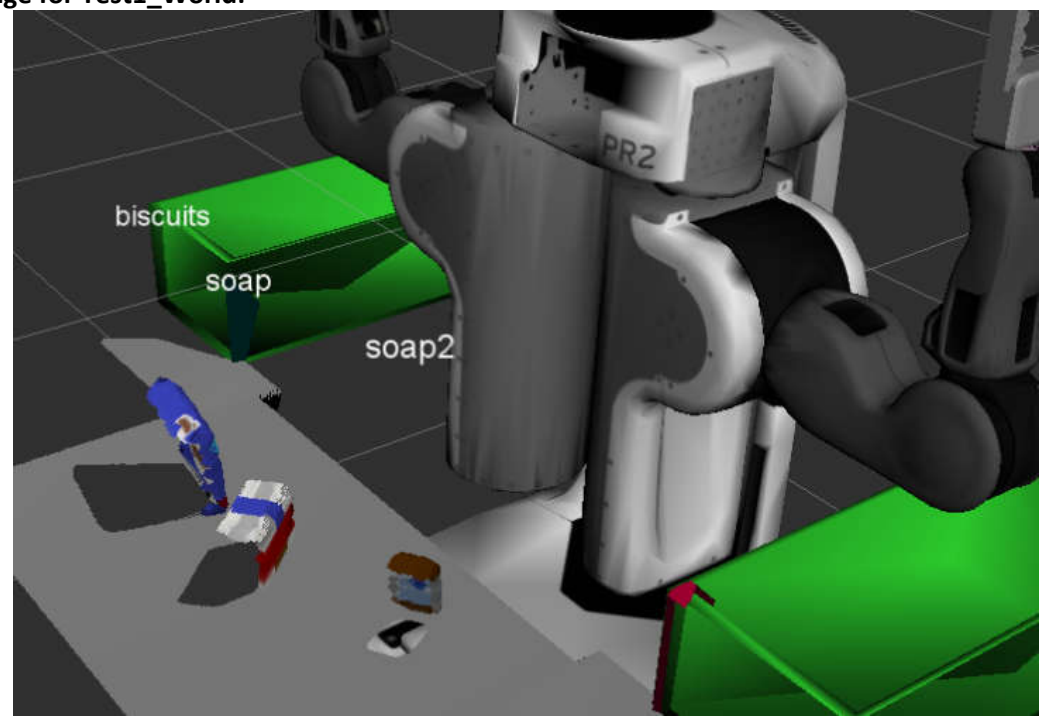

Figure 19: Rviz environment for test1_world with all the object labels

Test2_world:

Console message:

Detected 5 objects: ['biscuits', 'book', 'soap', 'soap2', 'glue']

Rviz image for test2_world:

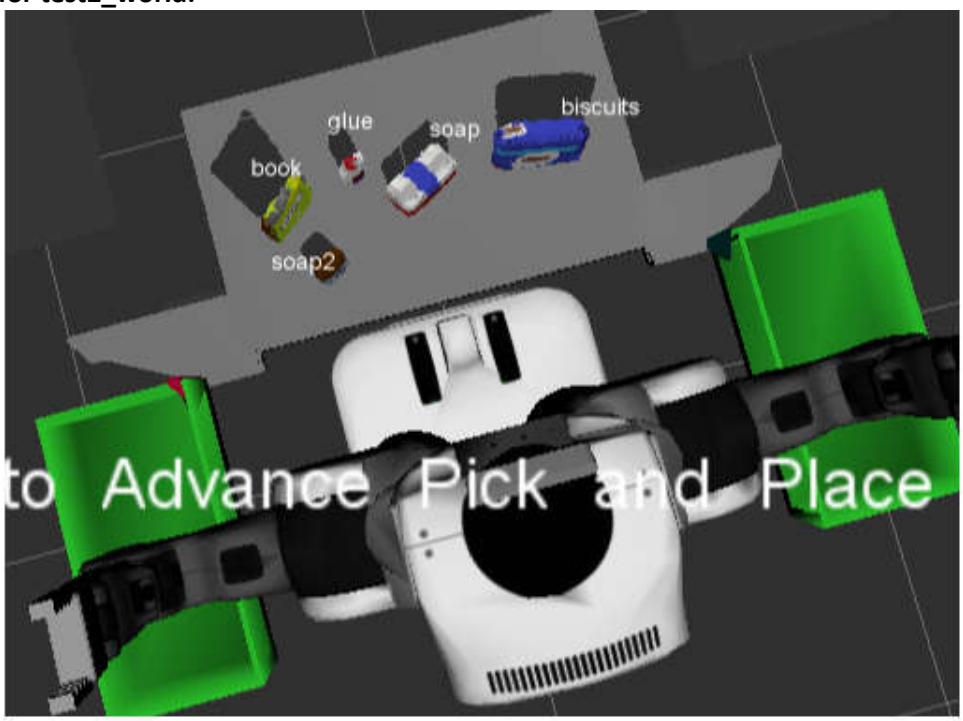

Figure 20: Rviz environment for test2_world with all the object labels

Test3_world:

Console message:

[INFO] [1549356566.145051, 1320.008000]: Detected 8 objects: ['snacks', 'biscuits', 'soap', 'book', 'eraser', 'sticky_notes', 'soap2', 'glue']

Rviz environment for test3_world:

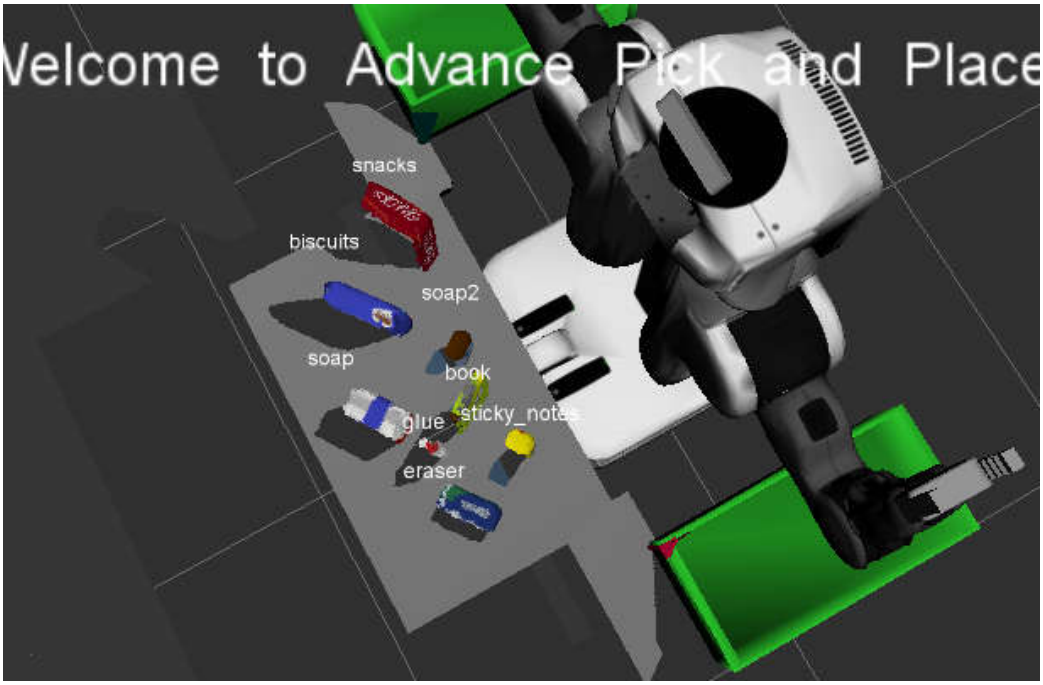

Figure 21:Rviz environment for test3_world with all the object labels

Generating Output.yaml files

These pick lists exist as yaml files under /pr2_robot/config and are loaded to the ros parameter server with the following line in your project launch file:

```
<rosparam command="load" file="$(find
pr2_robot)/config/{PICK_LIST_NAME}.yaml"/>
```

Followed following steps to generate the output.yaml files:

1.Added all the code blocks from object.recogintion.py to project_template.py file.

2.Updated the pr2_mover () function inside the project_template.py file

Followed following instructions as mentioned in classroom and updated the project_template file:

4. Check if the detected labels are same as pick_list objects.
5. If yes, invoke pr2_mover () function call.
6. For every object in pick list, get the object name, object group. Find the respective object name in the detected object list and get the centroid of the object.
7. Update the pick_pose with centroid of the detected object.

8. Keep doing the above, unless all the objects in pick list are covered.
9. For each object, find out the place position, from dropbox list.
10. Update ros message with all the required parameters e.g test_scene_num, object_name, arm_name , pick_pose and the place_pose
11. Change the test_scene_num and output_*.yaml file name to generate output for each test cases.

Output_1. yaml, output_2. yaml and output_3.yaml files are generated and shared here :

RoboND-Perception-Project\pr2_robot\config folder.

Discussion and Future Improvements:

I tried lot of iterations to make the model accurate. I tried to grab various feature sets of an object by running the for loop in capture_features.py for 100,200,250 and 300 times. I also tried with various kernel e.g. sigmoid, linear and rbf. Most of the time the model is as accurate as 99% but it is unable to detect the objects correctly in all the test world scenarios. Finally, the 300 feature sets (for loop running 300) of an object with linear kernel gives most accurate result and it is able to detect all the object in three test world scenarios with 100% accuracy.

I have to develop more understanding regarding why a model even if 99% accurate, fails to detect all the objects at least with 80% accuracy. On a similar note, I tried different parameters for Euclidean clustering and finally it worked best with setting of 0.02,100 and 25000 for cluster tolerance, minimum cluster size and maximum luster size respectively.

I propose to improve the modelling algorithm so as to improve the model accuracy with a smaller number of feature sets. Moreover, it might help to reduce the latency of the system. The pr2 robot is not able to pick and place the object accurately into the bins as the object collides with the environment and falls out of the table> we can implement the challenge section to detect the Collison map and should be able to complete the pick and place project in true essence.