3 D Perception Project

First, follow readme.md to setup environment and copy sensor_stick project from exercises to ~/catkin_ws/src.

Preparing for training models

Launch the `training.launch` file to bring up the Gazebo environment:

```
$ roslaunch sensor_stick training.launch
```

Capturing Features

```
$ rosrun sensor_stick capture_features.py
```

Under ~/catkin_ws/src/sensor_stick/src/sensor_stick, couple of utility functions are implemented in features.py. Function get_normalized_hist computes normalized histograms from three channel values with bins size and range.

Function compute_color_histograms calls get_normalized_hist with default bins 32 and range (0,256) for either RGB or HSV to computer color histograms.

Function compute_normal_histograms calls get_normalized_hist with default bins 32 and range (-1,1) to computer normal vector histograms.

Capture_features.py in ~/catkin_ws/src/sensor_stick/scripts uses function compute_color_histograms and compute_normal_histograms to compute features for models [‘sticky_notes','biscuits','soap','soap2','book','glue','snacks','eraser'] with 600 positions and 10 retires to get training set.

Training

Training needs `sklearn` and `scipy` Python packages.

```
$pip install sklearn scipy
$ rosrun sensor_stick train_svm.py
```

Train_svm.py takes features from training set to train with a linear SVM kernel and penalty parameter C, 0.05.

```
svm.SVC(kernel='linear', C=0.05)
```

I got accuracy score 0.9325. Trained model is saved in model.sav. The corresponding normalized confusion matrix is shown in Fig. 1.

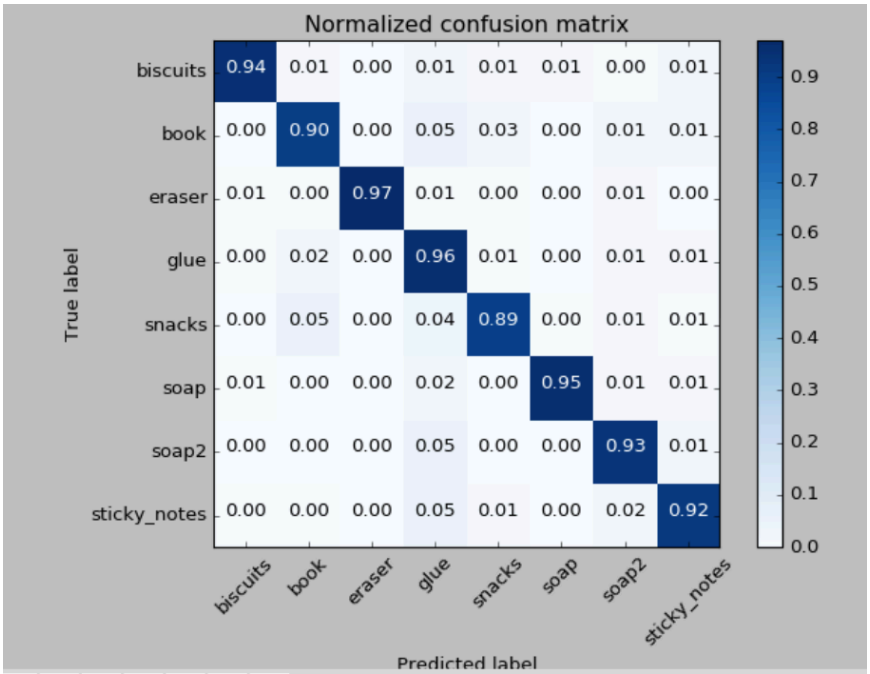

Fig.1 Normalized Confusion Matrix

Recognize Objects

To recognize objects, we need to pass cloud points through filter pipelines to get objects on the table. It's implemented in project_template.py of ~/catkin_ws/src/RoboND-Perception-Project/pr2_robot/scripts. I copied model.sav from prior step to this directory.

Filter Pipelines

1. Statistical Outlier Filtering

Applying statistical outlier filter with mean k = 20 and standard deviation multiplier to 0.1 to eliminate white noise

2. **Voxel Grid Downsampling filter**

Set LEAF_SIZE to 0.01 for the Voxel filter

3. Pass Through filter

We need to set pass through filter on both y-axis (-0.4, 0.4) and z-axis (0.61, 0.9).

4. **RANSAC Plane filter**

Set model type to pcl.SACMODEL_PLANE, method type to pcl.SAC_RANSA, and threshold to 0.01.

Then I got table and objects from segmentation.

5. **Euclidean Clustering**

Applying Euclidean Clustering to objects with cluster tolerance 0.05, minimum cluster size 100, maximum cluster size 3500, and kdtree search, cluster indices got extracted from cloud objects for each objects.

Classify Clusters and Recognize Objects

Looping through the cluster indices, transforming point cloud structure to Ros structure, computing features from color and normal histograms, I use the model.sav from prior steps to predict object

Mover and Saving result to Yaml files

Read parameters

- from /object_list about ground truth of objects to be recognized
- from /dropbox about the color and position of two drop boxes
- from /pr2_cloud_transformer/test_num about the test case number

I created two dictionaries, one from the detected objects with object name as key and object as value, the other from drop boxes with color as key and position as value.

Then I loop through the ground truth object list, looking for a match in the detected object dictionary and dropbox dictionary, creating a Yaml structure with test scene number, arm name, object name, pick object position, and place position (dropbox).

Rospy service pick_place_routine can be used to pick an object from pick position to a dropbox by the place position.

The following launch command starts Gazebo and RViz windows.
$roslaunch pr2_robot pick_place_project.launch

On a different terminal, run project_template.py for filter pipeline and object recognition with command rosrun pr2_robot project_template.py.

Output Yaml Files are saved in the ~/catkin_ws/src/RoboND-Perception-Project/pr2_robot/scripts/results.

Test Case	Recognition %	
1	3/3 (100%)	
2	4/5 (80%)	missing glue
3	7/8 (87.5%)	missing glue

Table 1 Object Recognition in Yaml Results

Fig.2, Fig.3, and Fig.4 show screenshots from the three test cases.

Fig. 2 Test Case 1

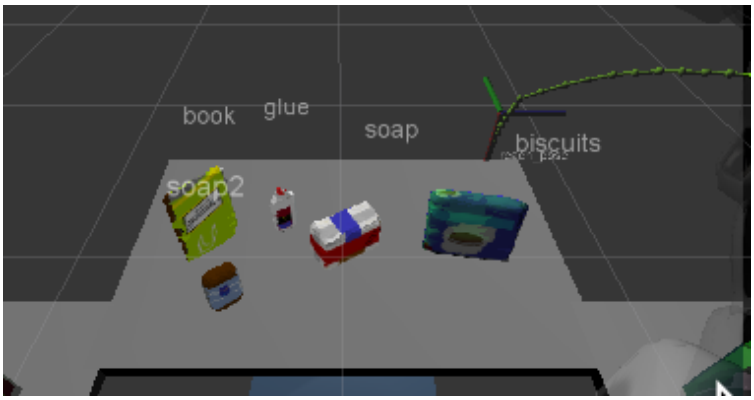

Fig. 3 Test Case 2

Fig. 4 Test Case 3

Future Work

- Improve accuracy, 93% is good but not enough for real-world cases. CNN probably is better way than SVM.
- Pick and place, getting collision with table or objects occasionally
- Optimize trajectory planning