Project: Perception Pick & Place

By Lingzhi Kong

Udacity Robotics Nano Degree

13 September, 2018

Abstract

This project aims to simulate the perception and pick & place problem using PR2 in ROS. In particular, the PR2 is outfitted with RGB-D camera to perceive its surrounding and output the camera data, namely point cloud, which includes some noise. This data is used as an input to develop a perception pipeline. Firstly, filtering and RANSAC plane fitting is applied to isolate the objects of interest from the rest of the scene. Secondly, Euclidean clustering is used to create separate clusters for individual items. Thirdly, extract features and train an SVM model on new objects, and assign them labels. Fourthly, calculate the centroid (average in x, y and z) of the set of points belonging to that each object. Finally, command the PR2 robot to pick and place each object using the position derived from step four.

2 Details

2.1 Pipeline for filtering, RANSAC plane fitting, and clustering

2.1.1 Outlier Removal Filter

While calibration takes care of distortion, noise due to external factors like dust in the environment, humidity in the air, or presence of various light sources lead to sparse outliers which corrupt the result even more. Thus, it is advantageous to perform filtering techniques to remove the noise. One of the most frequently used algorithms is PCL's Statistical Outlier Removal filter, which computes the distance to all of its neighbors, and then calculate a mean distance. By assuming a Gaussian distribution, all points whose mean distance is outside of an intrivel defined by a global mean and a standard deviation are considered to be outliers and removed from the point cloud. The following figure shows the result of applying the Statistical Outlier Removal Filter to noisy point cloud data. We can see that the noisy points are reduced significantly.

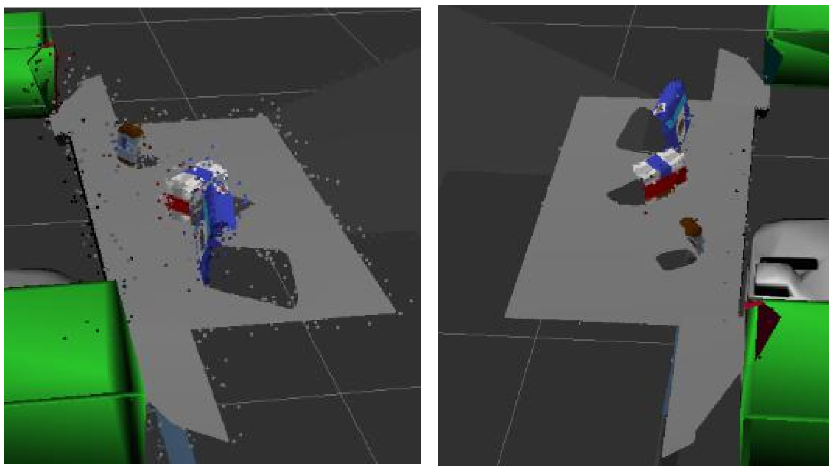


Fig. 2.1 Outlier Removal Filter

2.1.2 Voxel Grid Downsampling

The reason why performing downsampling is that in many cases, it is advantageous to downsample the data because in this way, we can process fewer data points, and therefore increasing the processing speed. In particular, I am going to use a VoxelGrid Downsampling Filter to derive a point cloud that has fewer points but should still do a good job of representing the input cloud as a whole. The center idea of this algorithm is that by dividing the point cloud into voxel equally, we can downsample all the points within the boundary of one particular voxel to one point. When using the algorithm, notice the parameter LEAF_SIZE which we should tweak, representing the size of the voxel. The following figure shows the result of Voxel Grid Downsampling.

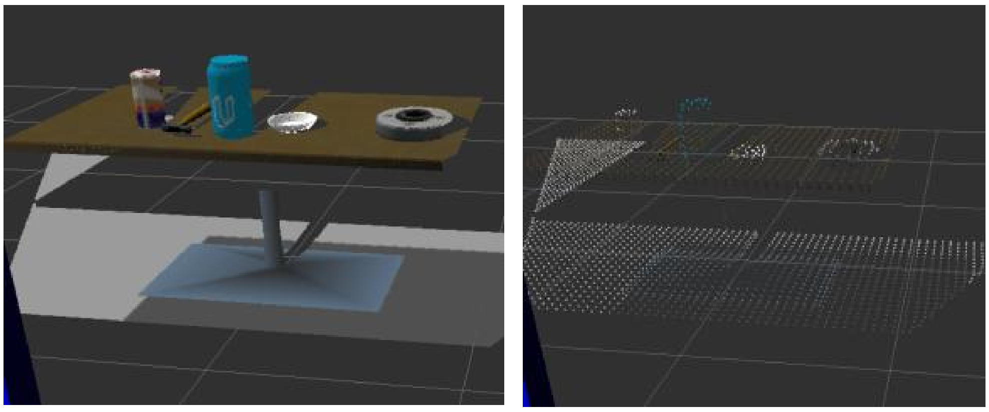


Fig. 2.2 Voxel Grid Downsampling

2.1.3 Pass Through Filtering

If we have previous information about the location of our target in the scene, we can simply remove the useless data by Pass Through Filtering. The region you allow to pass through, is often referred to as “region of interest”. The following figure shows the result of applying Pass Through Filtering along z axis.

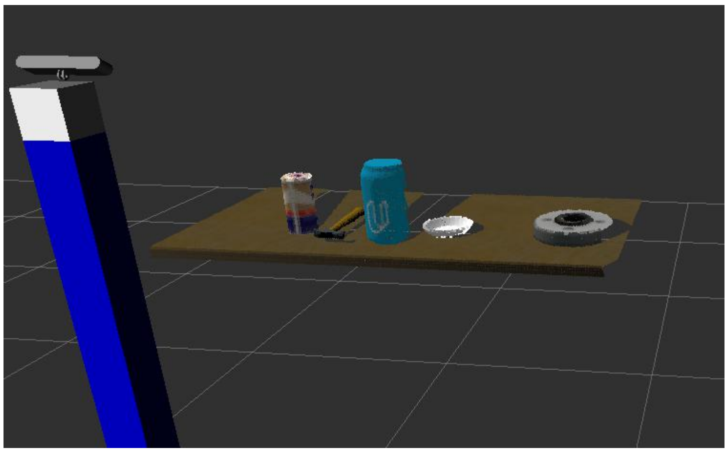


Fig. 2.3 Pass Through Filtering

2.1.4 Random Sample Consensus

RANSAC is an algorithm, which we can use to identify the points cloud that belong to a particular model. In the case we are working with, the models are a cylinder, a plane, a box, or any other common shape. The following figure shows the result of applying RANSAC.

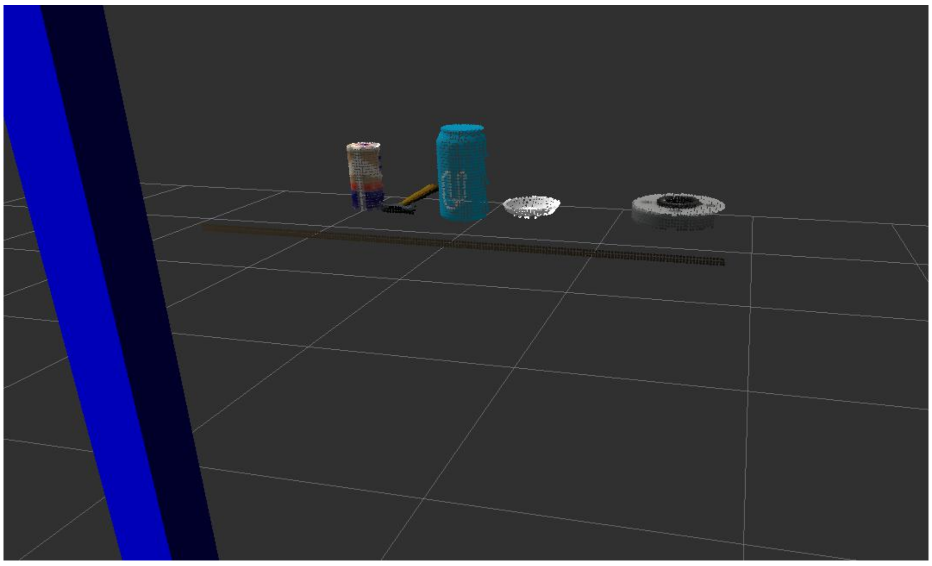


Fig. 2.4 Random Sample Consensus

2.1.5 Density-Based Spatial Clustering of Applications with Noise

DBSCAN is an algorithm, which we can use to cluster the points when we do not know how many clusters to expect in our data. The following figure shows the result of applying DBSCAN.

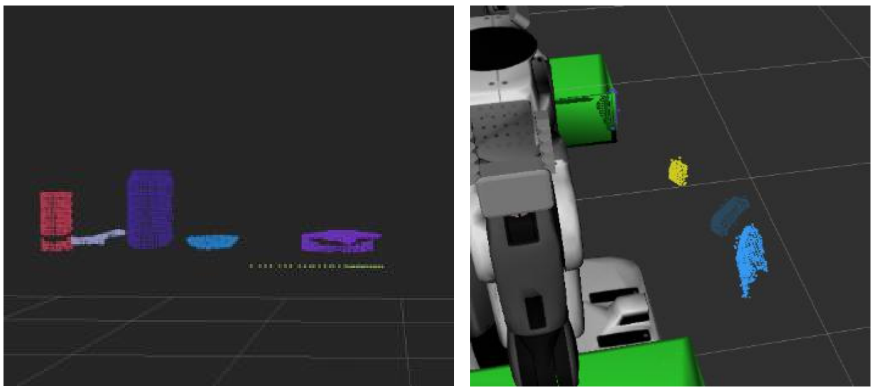


Fig. 2.5 DBSCAN

2.2 Features extraction, SVM training, and object recognition

2.2.1 Features extraction

The goal of features extraction is to obtain features of each item, and then assign a label to each of them. In this way, we can derive a dataset containing the items' features and their labels, which become the training set of SVM. In the case we are working with, we simply extract the color histograms features and the histogram of gradient. In particular, we need to divide up the range of the data color values 0-255 into discrete bins, and then count up how many values fall into each bin. To compare histograms of different total numbers of points, we'd like to normalize the result, which is to say, make it such that the sum of all the bins in the histogram is equal to 1.

2.2.2 SVM training

Now that we get the training data set from the previous step, we can step forward to train our SVM model. SVM is a typical supervised machine learning algorithm that allows us to characterize the parameter space of our dataset into discrete classes. SVMs work by applying an iterative method to a training dataset, where each item in the training set is characterized by a feature vector and a label. The following figure shows the confusion matrix of the test result.

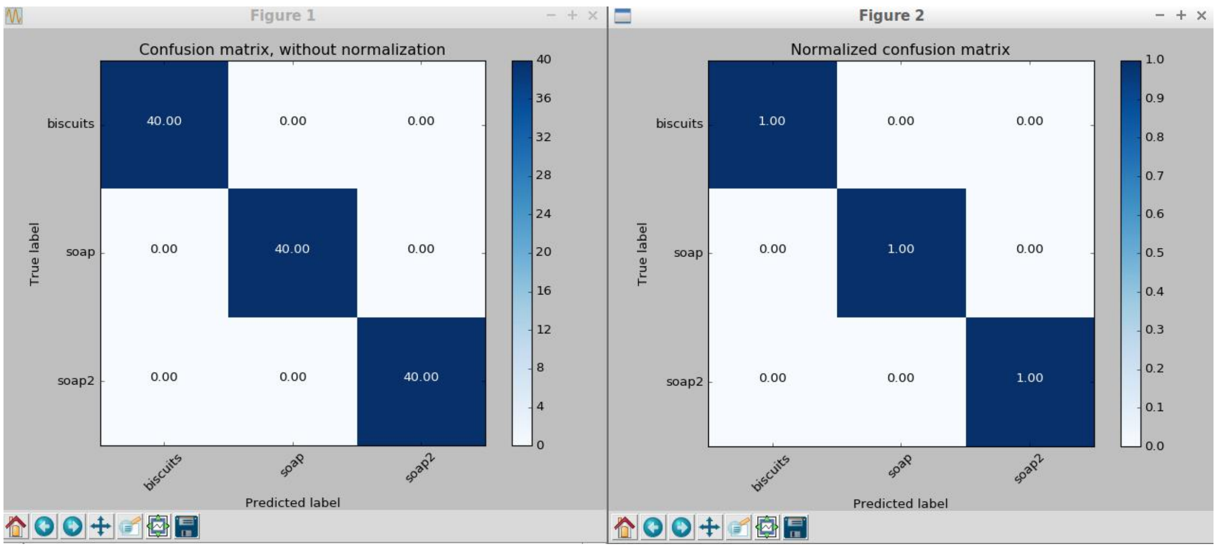


Fig. 2.6 SVM confusion matrix(scene 1)

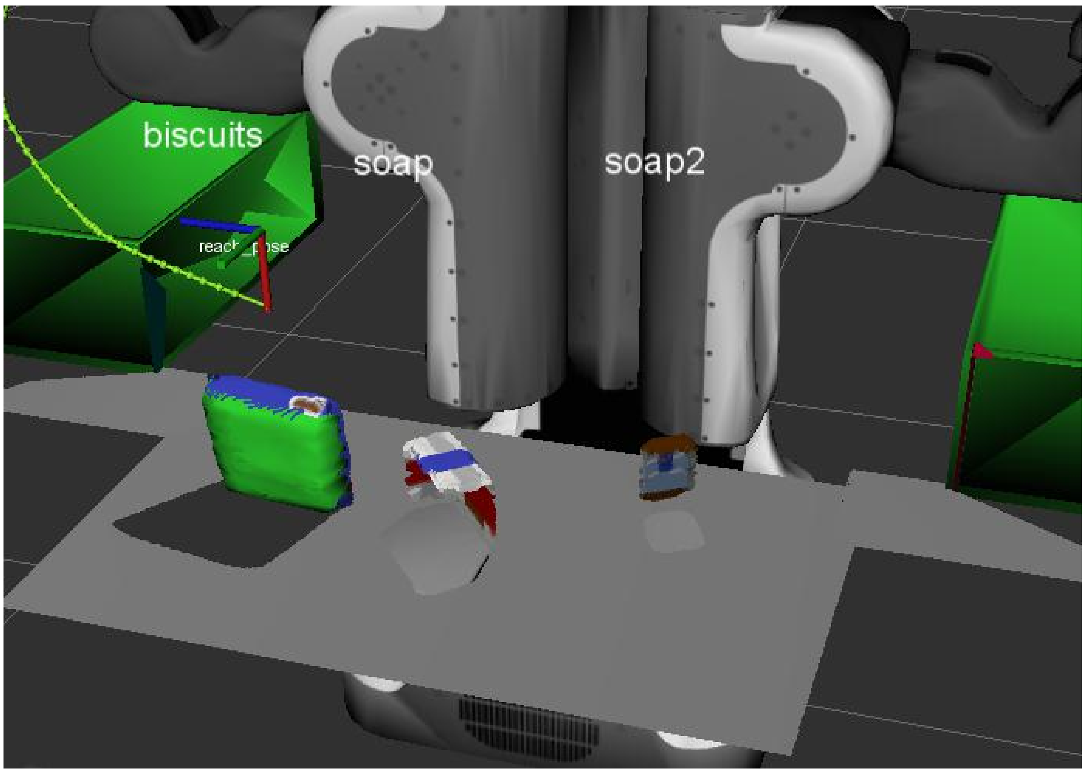


Fig. 2.7 scene 1

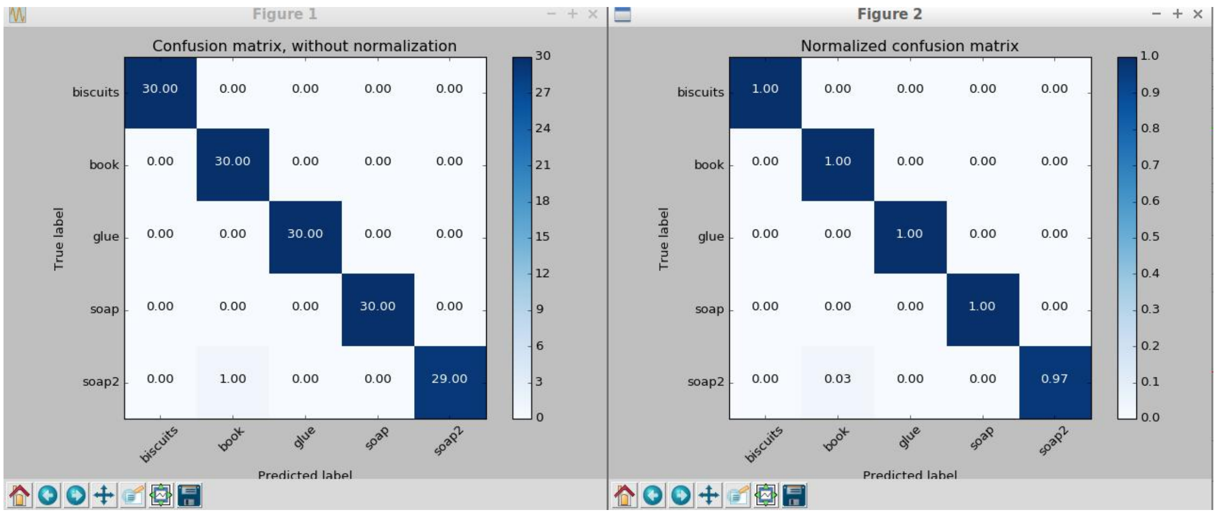


Figure 1
Figure 2
Confusion matrix, without normalization
Normalized confusion matrix
True label
Predicted label

Confusion matrix, without normalization

	biscuits	book	glue	soap	soap2
biscuits	30.00	0.00	0.00	0.00	0.00
book	0.00	30.00	0.00	0.00	0.00
glue	0.00	0.00	30.00	0.00	0.00
soap	0.00	0.00	0.00	30.00	0.00
soap2	0.00	1.00	0.00	0.00	29.00

Normalized confusion matrix

	biscuits	book	glue	soap	soap2
biscuits	1.00	0.00	0.00	0.00	0.00
book	0.00	1.00	0.00	0.00	0.00
glue	0.00	0.00	1.00	0.00	0.00
soap	0.00	0.00	0.00	1.00	0.00
soap2	0.00	0.03	0.00	0.00	0.97

Fig. 2.8 SVM confusion matrix(scene 2)

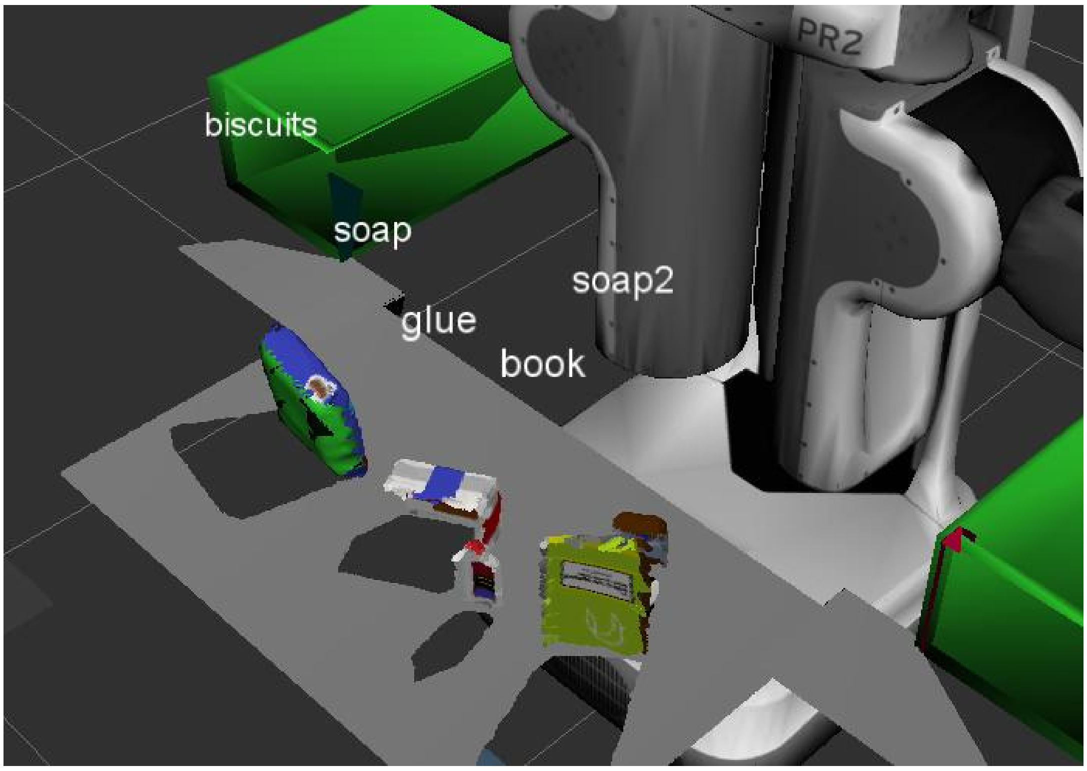
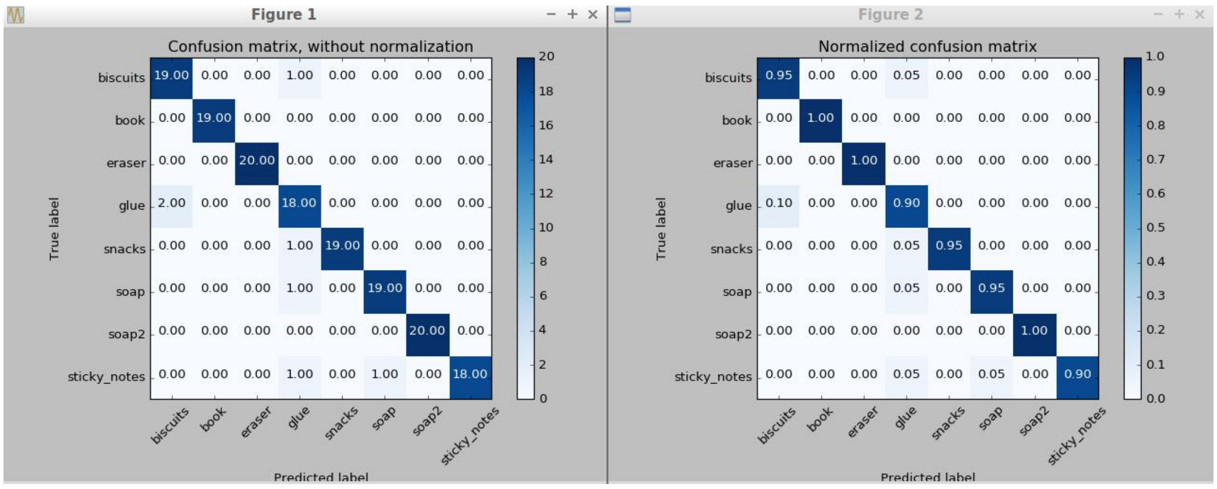


Fig. 2.9 scene 2



Confusion matrix, without normalization

	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	19.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
book	0.00	19.00	0.00	0.00	0.00	0.00	0.00	0.00
eraser	0.00	0.00	20.00	0.00	0.00	0.00	0.00	0.00
glue	2.00	0.00	0.00	18.00	0.00	0.00	0.00	0.00
snacks	0.00	0.00	0.00	1.00	19.00	0.00	0.00	0.00
soap	0.00	0.00	0.00	1.00	0.00	19.00	0.00	0.00
soap2	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.00
sticky_notes	0.00	0.00	0.00	1.00	0.00	1.00	0.00	18.00

Normalized confusion matrix

	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	0.95	0.00	0.00	0.05	0.00	0.00	0.00	0.00
book	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
eraser	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
glue	0.10	0.00	0.00	0.90	0.00	0.00	0.00	0.00
snacks	0.00	0.00	0.00	0.05	0.95	0.00	0.00	0.00
soap	0.00	0.00	0.00	0.05	0.00	0.95	0.00	0.00
soap2	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
sticky_notes	0.00	0.00	0.00	0.05	0.00	0.05	0.00	0.90

Fig. 2.10 SVM confusion matrix(scene 3)

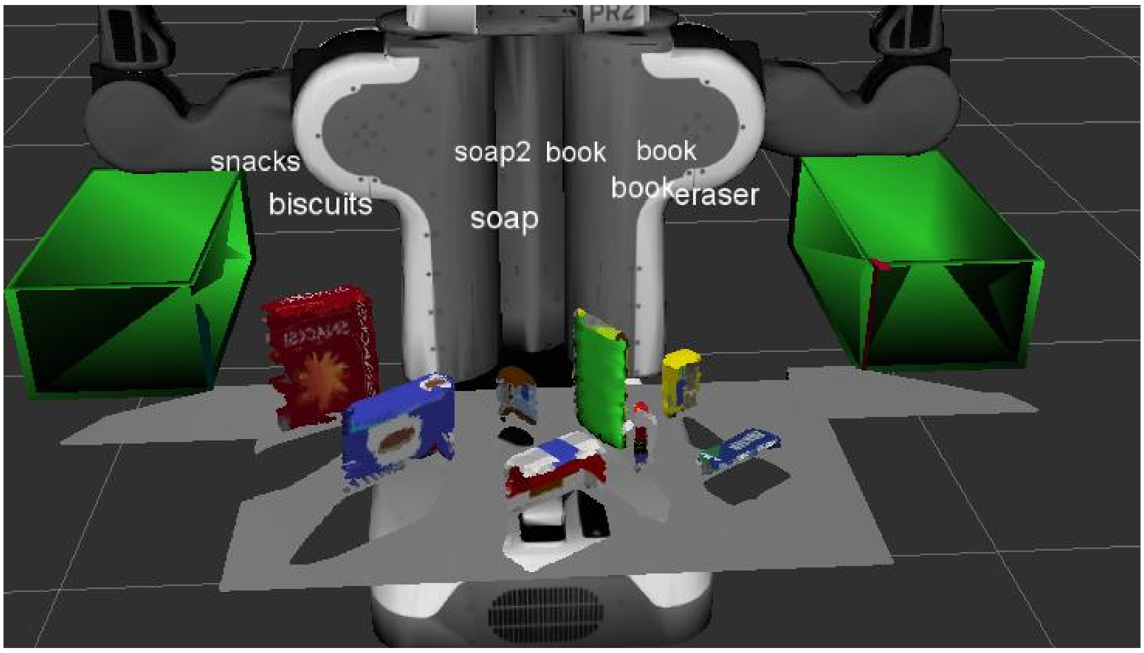


Fig. 2.11 Scene 3

2.2.3 Object recognition

When implementing the project for each of the three scenes, note that there are 5 lines of code we should change:

1. pr2_robot/launch/pick_place_project.launch Line 14 <arg name="world_name" value="$(find pr2_robot)/worlds/test2.world"/>

2. pr2_robot/launch/pick_place_project.launch Line 40 <rosparam command="load" file="$(find pr2_robot)/config/pick_list_2.yaml"/>

3. /pr2_robot/scripts/project_imp line 329 model = pickle.load(open('/home/robond/model_pick_2.sav', 'rb'))

4. /pr2_robot/scripts/project_imp line 234 yaml_filename = 'output_2.yaml'

5. /pr2_robot/scripts/project_imp line 235 test_scene_num.data = 2

After finishing the steps above, we are ready to launch the project. The following figure shows the pick and place scene of world_3.

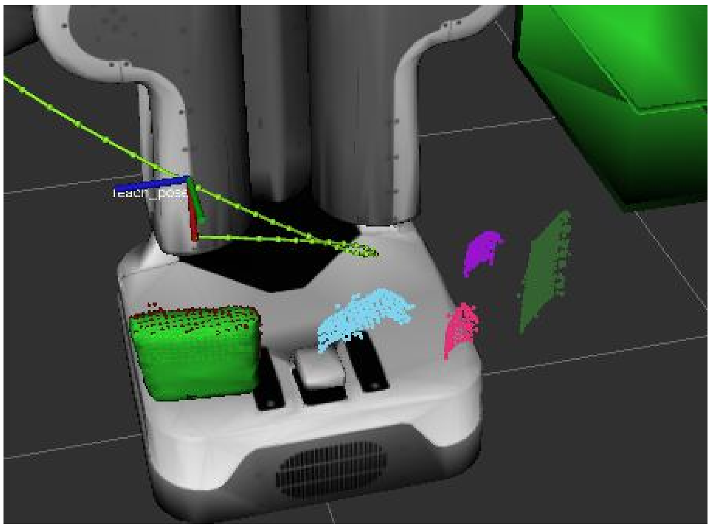


Fig. 2.12 Perception and pick & place

3 Conclusion & Discussion

5.1 Conclusion

In this project, we developed a perception pipeline and implemented it in ROS simulation environment. The outcome shows the feasibility of our method. However, some improvements need to be made to perfect the project. Firstly, we could capture more other features of the items to derive the training dataset. In this way, the SVM training model would be more robust. Secondly, we could also try to use various SVM kernel to see which one is the best. Last but not least, we could build a collision map for the robot to avoid collision when conducting pick and place task.