Introduction

This report is for the 3-D perception project, for Robo-ND from Udacity. The project is for PR2 Robot perception pipeline for object detection of point cloud (pcl). The perception pipeline will go through some pcl filtering and Ecludian clustering for segmentation then object detection using a SVM training set.

The perception pipeline will be tested on 3 given scenes with different numbers of objects.

Perception Pipeline

the perception code is written in a pcl_callback() function which is a callback function of point cloud subscriber.

pcl_callback()

the callback function, contains the perception pipeline. We will go through the techniques used for filtering, clustering and detection of the point cloud.

When the point cloud is sent to the function as ROS message. First, using the pcl_helper function to convert it to a PCL. then later, we use it again to convert to ROS message after the perception to be sent back to ROS.

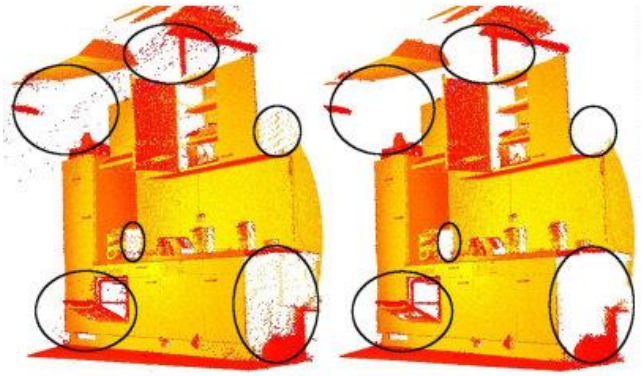
Statistical Outlier Filtering

Since the point cloud contains some noise. We use the Statistical Outlier Filtering as a technique to remove the noise and get a clear PCL ready for correct object detection.

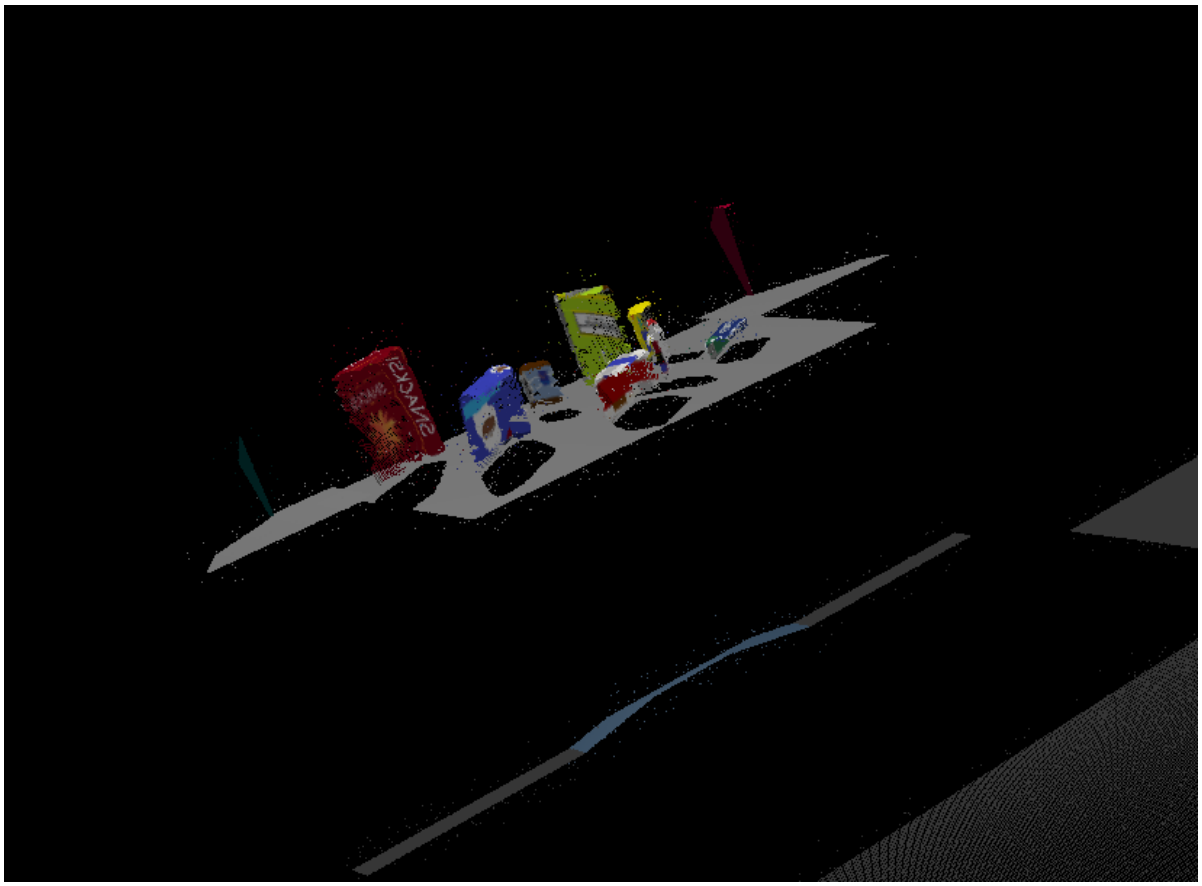
One of the filtering techniques used to remove such outliers is to perform a statistical analysis in the neighborhood of each point, and remove those points which do not meet a certain criteria. PCL's StatisticalOutlierRemoval filter is an example of one such filtering technique. For each point in the point cloud, it computes the distance to all of its neighbors, and then calculates a mean distance.

By assuming a Gaussian distribution, all points whose mean distances are outside of an interval defined by the global distances mean+standard deviation are considered to be outliers and removed from the point cloud.

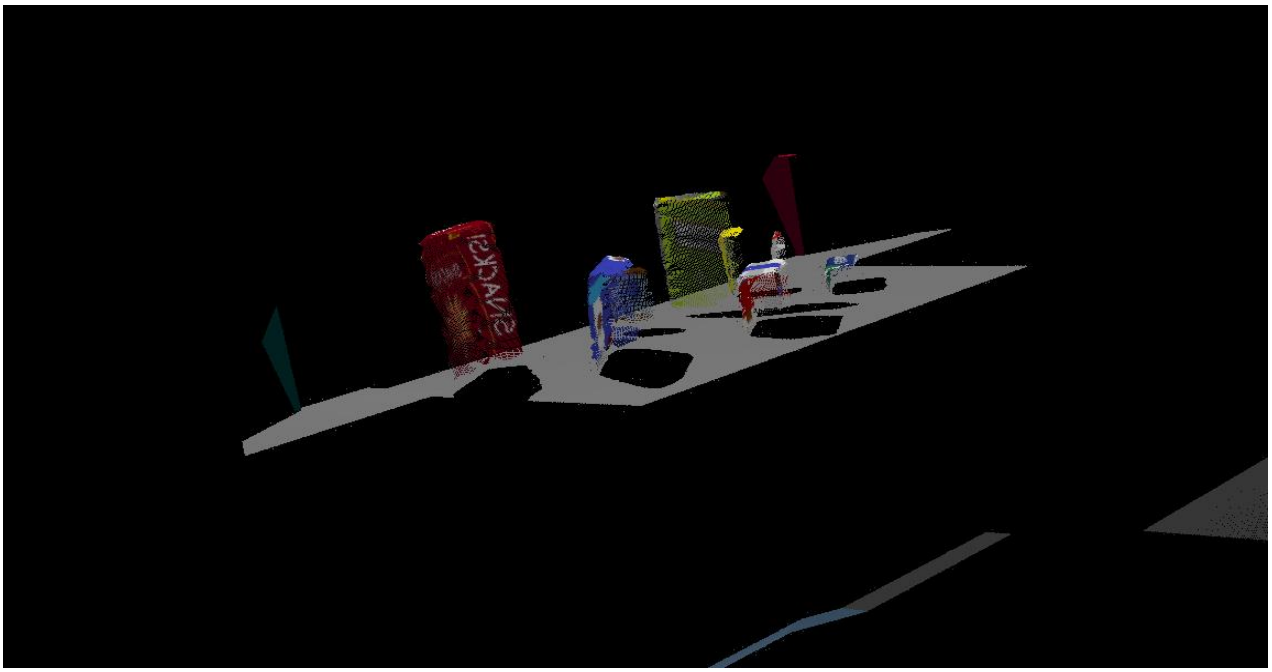
Code line (58:70)



Original Point



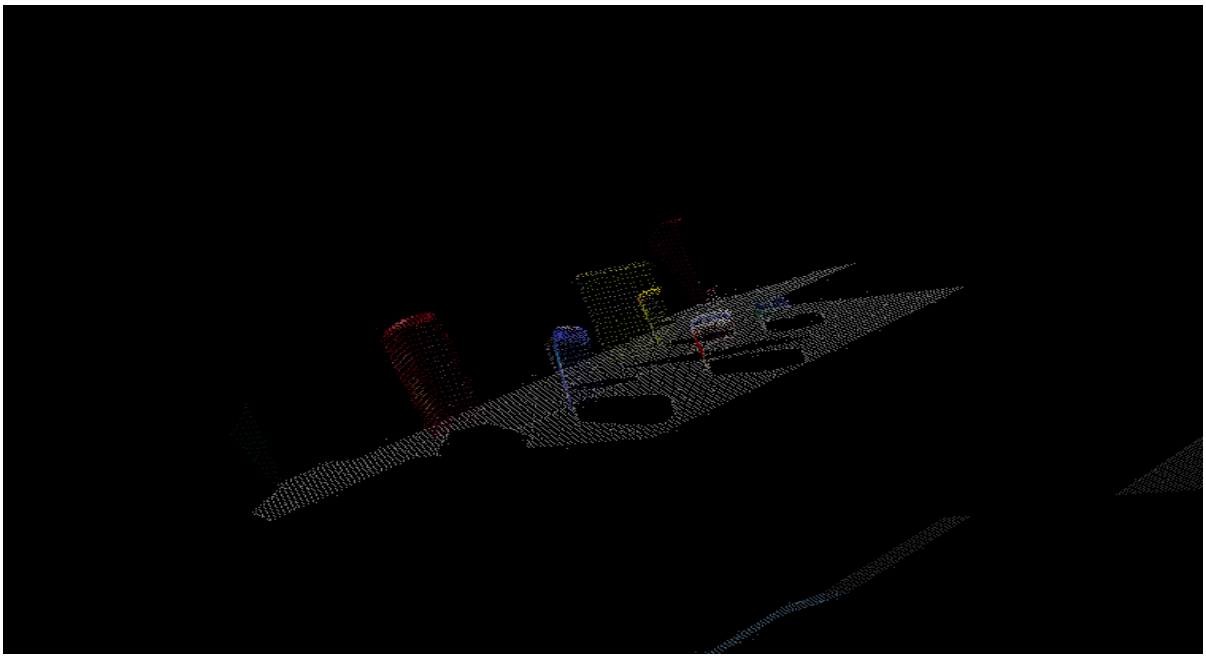
Cloud After filter



Voxel Grid

A voxel grid filter allows you to downsample the data by taking a spatial average of the points in the cloud confined by each voxel. You can adjust the sampling size by setting the voxel size along each dimension. The set of points which lie within the bounds of a voxel are assigned to that voxel and statistically combined into one output point.

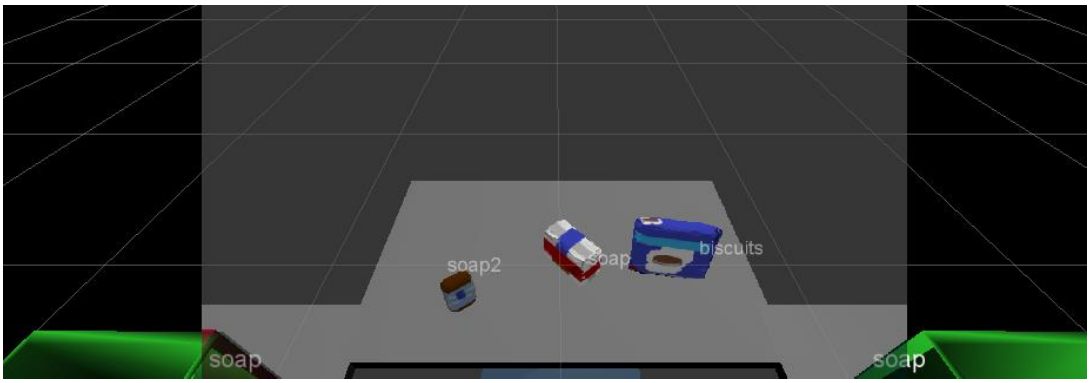
Code line (72:81)

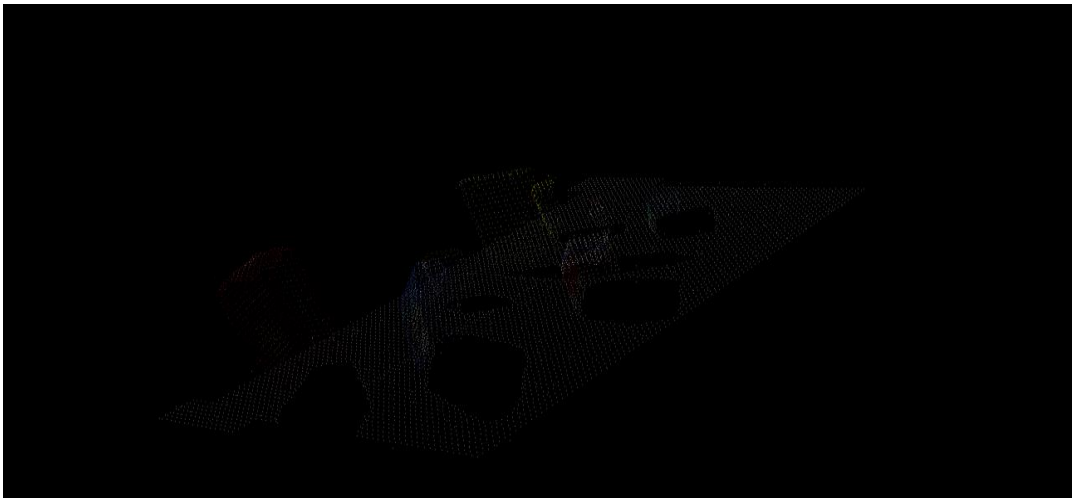


Pass Through Filtering

The Pass Through Filter works much like a cropping tool, which allows you to crop any given 3D point cloud by specifying an axis with cut-off values along that axis. The region you allow to *pass through*, is often referred to as ***region of interest***.

Since we have an interest of tabletop object we crop the cloud in z and y axis. After trying z axis only, the detection was getting false around the table. So, its good idea to have another filter across the y axis, **Code line (85:106).**

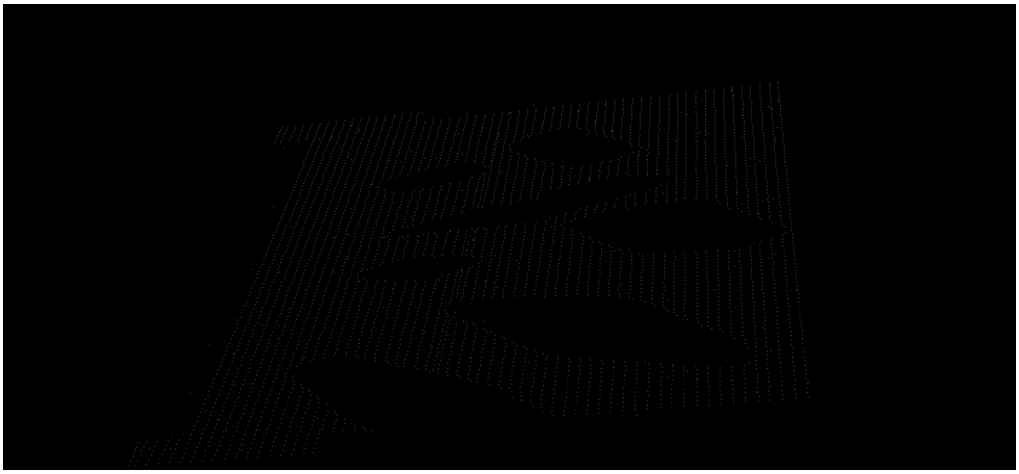




RANSAC plane segmentation

Since we need the objects only, we can use RASAC to find the points along a plane which belongs to the table. Then separate it from the cloud leaving the objects only in the scene.

Code line (110:139)



Euclidean Clustering

The DBSCAN algorithm creates clusters by grouping data points that are within some threshold distance dthd_{th}dth from the nearest other point in the data.
The algorithm is sometimes also called “Euclidean Clustering”, because the decision of whether to place a point in a particular cluster is based upon the “Euclidean distance” between that point and other cluster members.

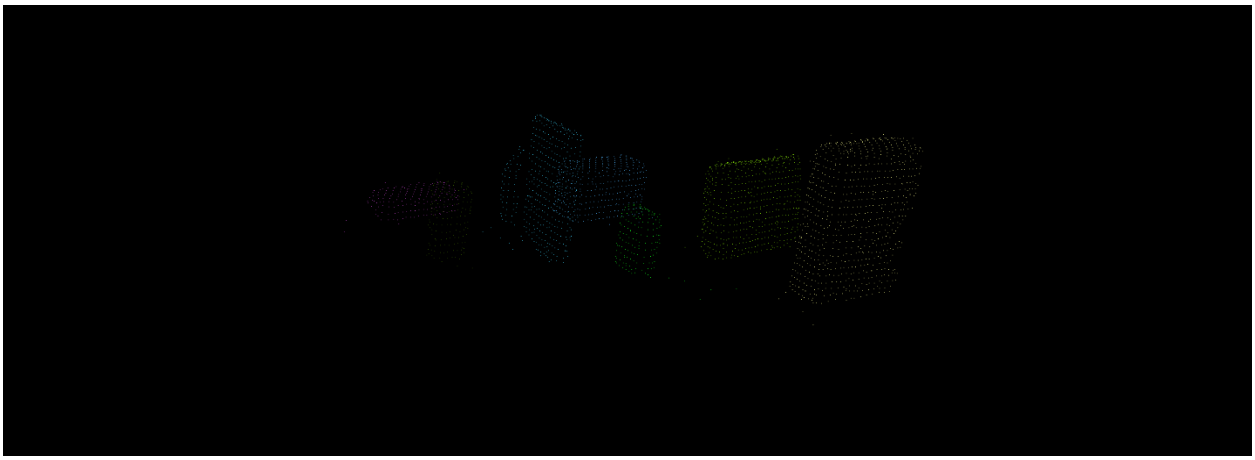
The Euclidean distance between points p \bold{p} p and q \bold{q} q in an n-dimensional dataset, where the position of p \bold{p} p is defined by coordinates (p1,p2,...,pn) (p_1, p_2, ..., p_n) (p1 ,p2,...,pn) and the position of q \bold{q} q is defined by (q1,q2,...,qn) (q_1, q_2, ..., q_n) (q1,q2,...,qn) then the distance between the two points is just:

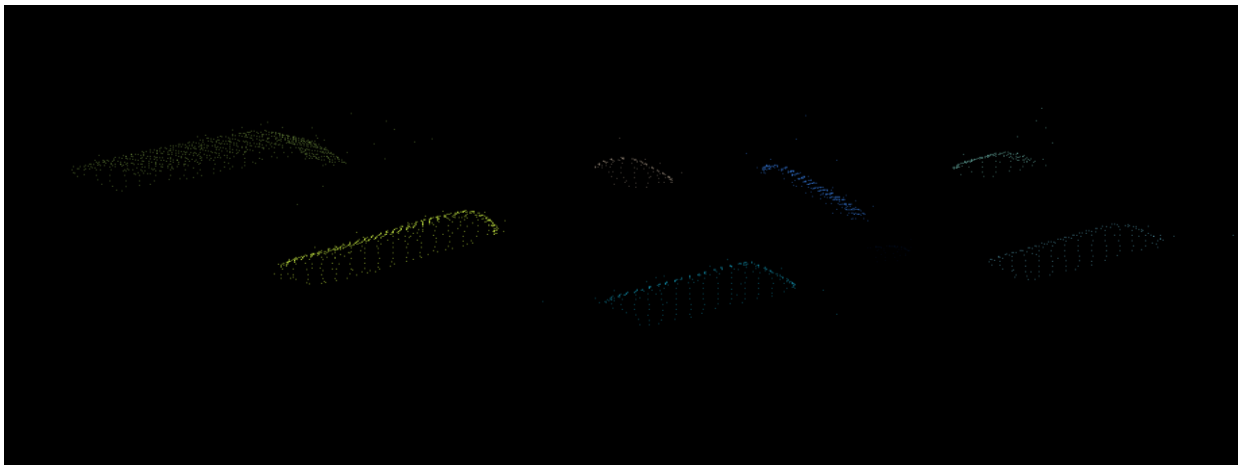
$$\mathbf{D} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

By the adjusting the parameter of the algorithm

- ec.set_ClusterTolerance(0.05)
- ec.set_MinClusterSize(20)
- ec.set_MaxClusterSize(6000)

then giving each cluster a unique color. **Code line (144:168).**





Objects Detection

Now, we have the objects cloud and ready for detection. But first, we should extract features from the object and train these features with SVM.

Thanks to functions from features code, the responsible to find the color histogram and surface normal for all the objects, compute_color_histograms() & compute_normals_histograms().

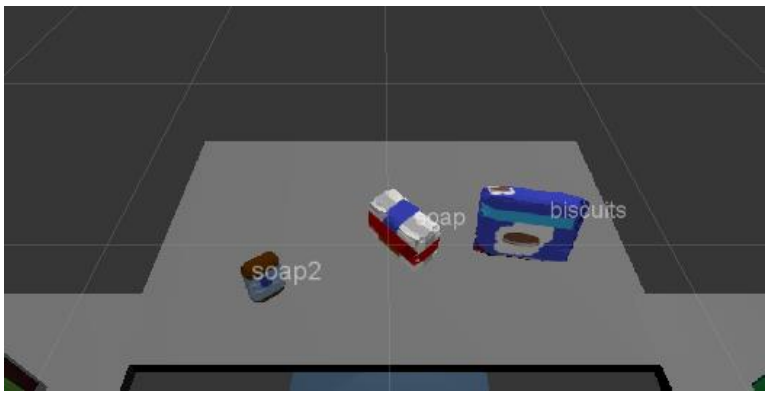
Then train the SVM, then perform the detection. **Code lines (184:218)**

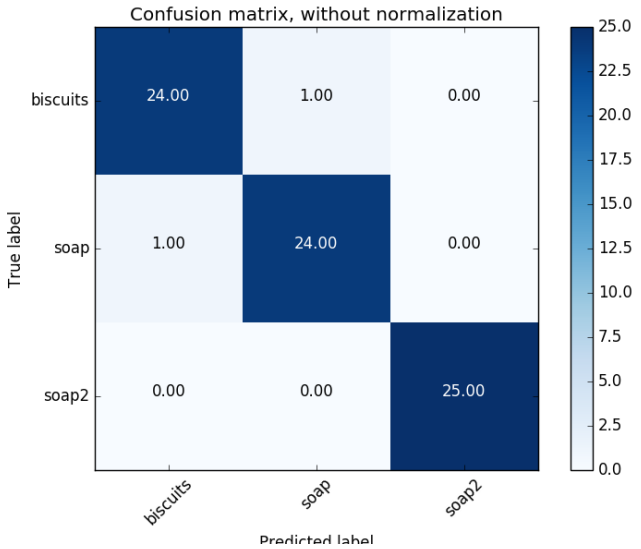
Increasing the features captured from the objects played a great rule to get a great accuracy level.

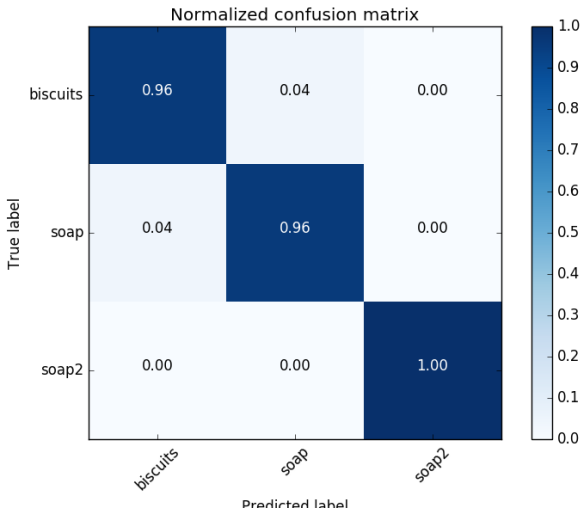
Results for each scene.

World 1:

- Features in Training Set: 75
- Invalid Features in Training set: 0
- Scores: [1. 1. 0.93333333 1. 0.93333333]
- Accuracy: 0.97 (+/- 0.07)
- accuracy score: 0.973333333333

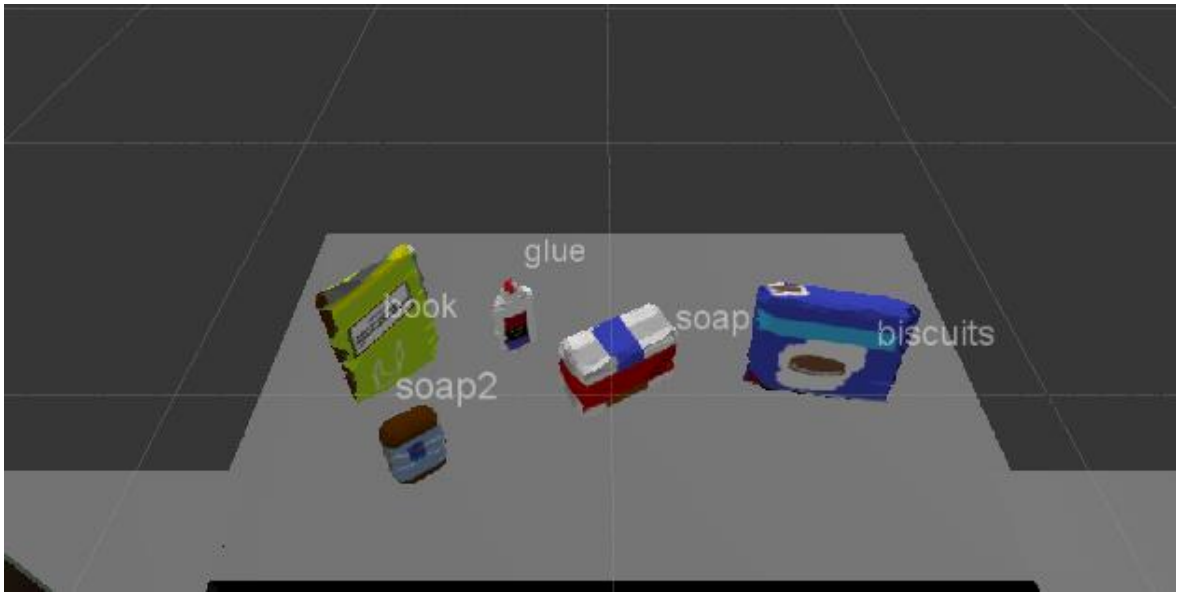


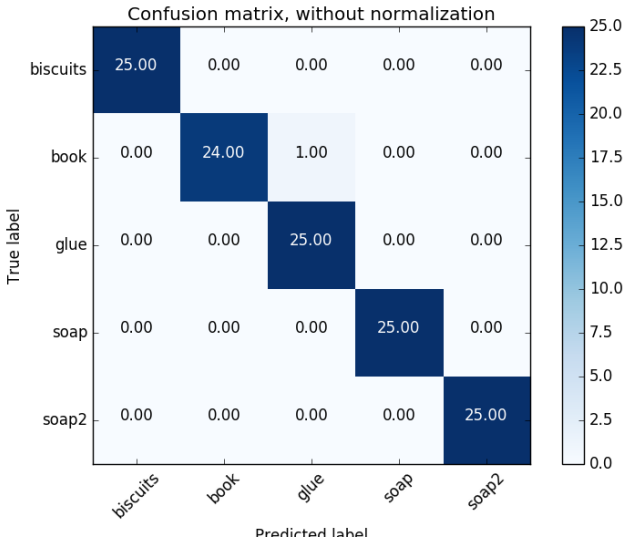


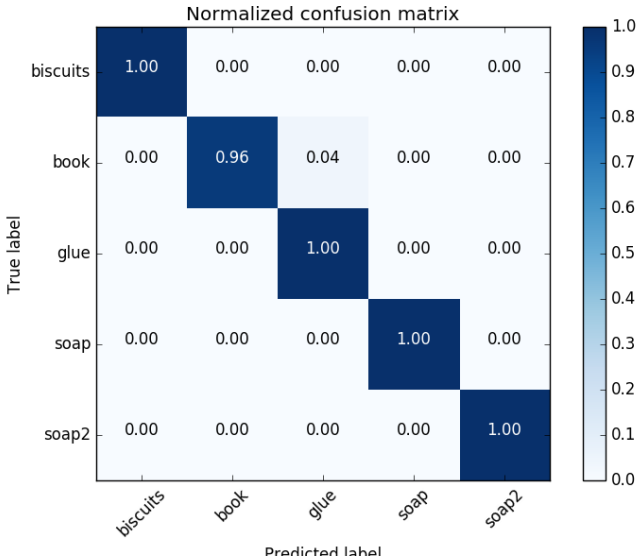


World 2:

- Features in Training Set: 125
- Invalid Features in Training set: 0
- Scores: [1. 1. 1. 1. 0.96]
- Accuracy: 0.99 (+/- 0.03)
- accuracy score: 0.992

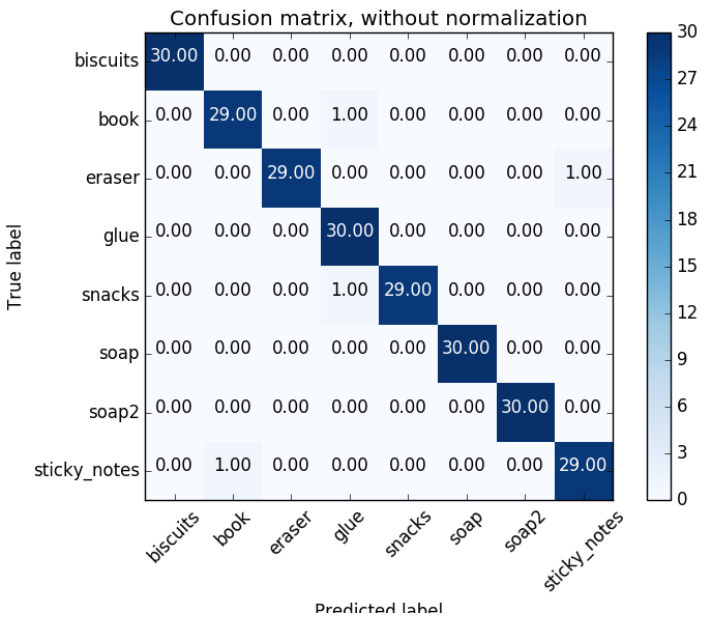


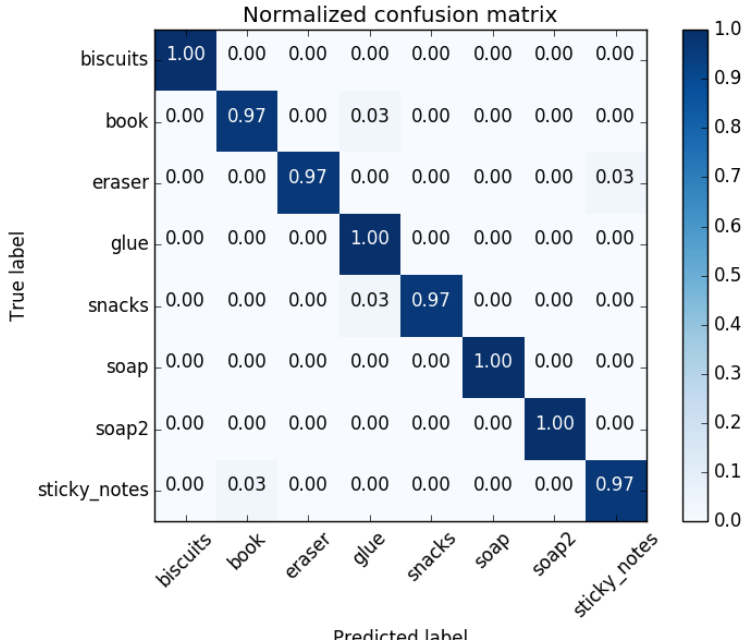




World 3:

- Features in Training Set: 240
- Invalid Features in Training set: 0
- Scores: [0.97916667 0.97916667 1. 0.97916667 0.97916667]
- Accuracy: 0.98 (+/- 0.02)
- accuracy score: 0.983333333333





Future Progress Plan

- Using different types of SVM kernels (sigmoid, rbf, …)
- Increasing the capturing features when expecting similar color objects
- Control the robot motion to avoid collision