

Project 3: 3D Perception

3D Perception with machine learning

Christopher J. Rukas | Robotic Software Engineer | 12/06/2018

# Project Description

Project 3: 3D Perception is a recreating of an Amazon competition in which a robot must correctly identify objects and place them into the correct bins without human input. In this course/project a Support Vector Machine (SVM) is trained to identify an assortment of items. To achieve the identification, clusters of point cloud data are segregated from a the raw point cloud data. Pass through filters are applied to segregate data, RANSAC is applied to identify a planar surface, and Euclidean Clustering (DBSCAN) is used to identify individual objects. The robot then creates a motion plan to pick and place each object, in the desired order, into the appropriate bin.

# Exercise 1 – Tabletop Segmentation

Exercise 1 requires identifying the tabletop data and separating objects from the tabletop. This is done in several steps:

1. Downsample the point cloud by applying a Voxel Grid Filter
2. Apply a Pass Through Filter to isolate the table and objects
3. Perform RANSAC plane fitting to identify the table’s points.
4. Extract indices of the plane from the objects and create two separate clouds.

Voxelizing data points

# Voxel Grid filter

# Create a VoxelGrid filter object for input point cloud

vox **=** cloud\_outlier\_filtered**.**make\_voxel\_grid\_filter**()**

# Choose a voxel (aka leaf) size

# Note: 1 is a poor choice of leaf size

LEAF\_SIZE **=** 0.01

# Set the leaf size

vox**.**set\_leaf\_size**(**LEAF\_SIZE**,** LEAF\_SIZE**,** LEAF\_SIZE**)**

# Call the filter function

cloud\_vox\_filtered **=** vox**.**filter**()**

A pass through filter identifies only data points within a specific zone with respect to the camera. In the following filter, two planes are created and only datapoints between those two planes are kept.

# PassThrough filter

# Create a PassThrough filter object

passthrough **=** cloud\_vox\_filtered**.**make\_passthrough\_filter**()**

#Assign axis and range to the passthrough filter object

filter\_axis **=** 'z'

passthrough**.**set\_filter\_field\_name**(**filter\_axis**)**

axis\_min **=** 0.55

axis\_max **=** 0.75

passthrough**.**set\_filter\_limits**(**axis\_min**,** axis\_max**)**

#Use the filter function to obtain the filtered cloud

cloud\_filtered **=** passthrough**.**filter**()**

RANdom Sample And Consensus (RANSAC) quickly identifies a set of points that most accurately fits a plane. The tools for RANSAC are included within the Point Cloud Library. A small set of bindings to the C++ library were made available within Python. Utilizing this, a plane is applied to identify the plane of the table surface. Points within a certain distance of the plane are assumed be a part of the table. The remaining points are assumed to be a objects.

# RANSAC plane segmentation

# Create segmentation of the object

seg **=** cloud\_filtered**.**make\_segmenter**()**

# Set the model you wish to fit

seg**.**set\_model\_type**(**pcl**.**SACMODEL\_PLANE**)**

seg**.**set\_method\_type**(**pcl**.**SAC\_RANSAC**)**

# Maximum distance for a point to be considered fitting the model

max\_distance **=** 0.01

seg**.**set\_distance\_threshold**(**max\_distance**)**

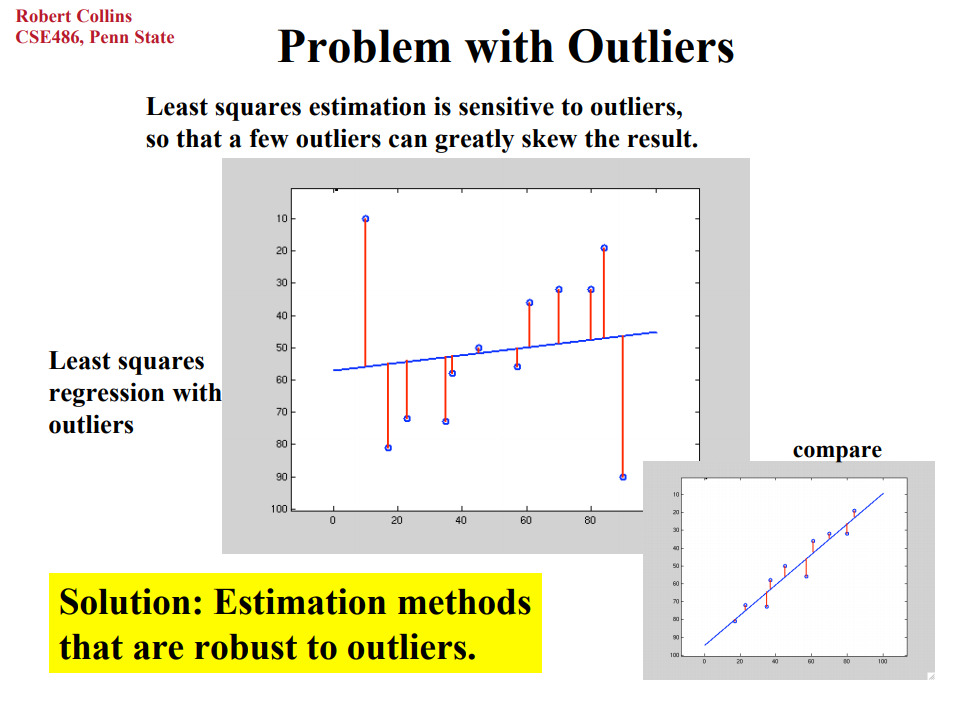
# Call the segment function to obtain set of inlier indicies and model coefficients

inliers**,** coefficients **=** seg1**.**segment**()**

extracted\_inliers **=** cloud\_filtered**.**extract**(**inliers**,** negative**=True)**

extracted\_outliers **=** cloud\_filtered**.**extract**(**inliers**,** negative**=False)**

RANSAC is used because it is robust in data fitting. Least squares methods can provide poor results.



Two clouds are created. Extracted inliers and Extracted outliers.

# Exercise 2

Exercise 2 builds upon the foundation of Exercise 1 and creates clusters of objects with the points separated from the table. A cloud of XYZ data (no RGB information) is passed to a KD tree. Any point may be applied to a cluster as long as its within 0.05 units from the next object. At least 50 points are required to make a cluster, and no more than 3000 points should be included in that cluster.

# Apply function to convert XYZRGB to XYZ

white\_cloud **=** XYZRGB\_to\_XYZ**(**extracted\_inliers**)**

# Create a K-D Tree

tree **=** white\_cloud**.**make\_kdtree**()**

# Create a cluster extraction object

ec **=** white\_cloud**.**make\_EuclideanClusterExtraction**()**

# Set tolerances for distance threshold

# as well as minimum and maximum cluster size (in points)

#

ec**.**set\_ClusterTolerance**(**0.05**)**

ec**.**set\_MinClusterSize**(**50**)**

ec**.**set\_MaxClusterSize**(**3000**)**

# Search the k-d tree for clusters

ec**.**set\_SearchMethod**(**tree**)**

# Extract indices for each of the discovered clusters

cluster\_indices **=** ec**.**Extract**()**

Colors are assigned to each cluster to visualize the results.

# Assign a color corresponding to each segmented object in scene

cluster\_color **=** get\_color\_list**(**len**(**cluster\_indices**))**

color\_cluster\_point\_list **=** **[]**

**for** j**,** indices **in** enumerate**(**cluster\_indices**):**

**for** i**,** indice **in** enumerate**(**indices**):**

color\_cluster\_point\_list**.**append**([**white\_cloud**[**indice**][**0**],**

white\_cloud**[**indice**][**1**],**

white\_cloud**[**indice**][**2**],**

rgb\_to\_float**(**cluster\_color**[**j**])** **])**

# Create new cloud containing all clusters, each with unique color

cluster\_cloud **=** pcl**.**PointCloud\_PointXYZRGB**()**

cluster\_cloud**.**from\_list**(**color\_cluster\_point\_list**)**

Utilizing pcl\_helper.py, the objects and the table are published as ROS messages.

# Exercise 3 – Object Recognition with Python, ROS, and PCL

Exercise 3 continues in the same vein as Exercise 1 and 2. The previous tools are utilized and the data is passed into a new virtual environment. An assortment of objects are loaded in random angles at a given position in front of the depth camera.

Histograms are generated for the RGB values and for the surface normals of the objects. The RGB values are distributed into 32 separate bins across a range of 256 values. RGB values are defined from 0 to 255.

**def** compute\_color\_histograms**(**cloud**,** using\_hsv**=False):**

# Compute histograms for the clusters

point\_colors\_list **=** **[]**

# Step through each point in the point cloud

**for** point **in** pc2**.**read\_points**(**cloud**,** skip\_nans**=True):**

rgb\_list **=** float\_to\_rgb**(**point**[**3**])**

**if** using\_hsv**:**

point\_colors\_list**.**append**(**rgb\_to\_hsv**(**rgb\_list**)** **\*** 255**)**

**else:**

point\_colors\_list**.**append**(**rgb\_list**)**

# Populate lists with color values

channel\_1\_vals **=** **[]**

channel\_2\_vals **=** **[]**

channel\_3\_vals **=** **[]**

**for** color **in** point\_colors\_list**:**

channel\_1\_vals**.**append**(**color**[**0**])**

channel\_2\_vals**.**append**(**color**[**1**])**

channel\_3\_vals**.**append**(**color**[**2**])**

# TODO: Compute histograms

r\_hist **=** np**.**histogram**(**channel\_1\_vals**,** bins**=**32**,** range**=(**0**,** 256**))**

g\_hist **=** np**.**histogram**(**channel\_2\_vals**,** bins**=**32**,** range**=(**0**,** 256**))**

b\_hist **=** np**.**histogram**(**channel\_3\_vals**,** bins**=**32**,** range**=(**0**,** 256**))**

# TODO: Concatenate and normalize the histograms

hist\_features **=** np**.**concatenate**((**r\_hist**[**0**],** g\_hist**[**0**],** b\_hist**[**0**])).**astype**(**np**.**float64**)**

norm\_features **=** hist\_features **/** np**.**sum**(**hist\_features**)**

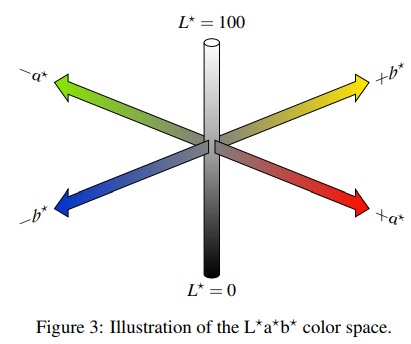
# Generate random features for demo mode.

# Replace normed\_features with your feature vector

#normed\_features = np.random.random(96)

#return normed\_features

**return** norm\_features

RGB are sensitive to lighting conditions and can provide poor results for matching. Different color spaces are available to compare items. Hue Saturation Value (HSV) is popular as it is less sensitive to lighting conditions. L\*a\*b\* is also useful with more sensitivity to what’s considered “human perception.” Machine generated alternative text:
HS\.J 
Saturation E Machine generated alternative text:
RGB 

Histograms for surface normal must also be calculated. For consistency, 32 bins are used again but this time in a range of 0 to 1. Normals are unit vectors and should not exceed 1. The function is extremely similar to the RGB function.

**def** compute\_normal\_histograms**(**normal\_cloud**):**

norm\_x\_vals **=** **[]**

norm\_y\_vals **=** **[]**

norm\_z\_vals **=** **[]**

**for** norm\_component **in** pc2**.**read\_points**(**normal\_cloud**,**

field\_names **=** **(**'normal\_x'**,** 'normal\_y'**,** 'normal\_z'**),**

skip\_nans**=True):**

norm\_x\_vals**.**append**(**norm\_component**[**0**])**

norm\_y\_vals**.**append**(**norm\_component**[**1**])**

norm\_z\_vals**.**append**(**norm\_component**[**2**])**

# TODO: Compute histograms of normal values (just like with color)

norm\_x\_hist **=** np**.**histogram**(**norm\_x\_vals**,** bins**=**32**,** range**=(**0**,** 1**))**

norm\_y\_hist **=** np**.**histogram**(**norm\_y\_vals**,** bins**=**32**,** range**=(**0**,** 1**))**

norm\_z\_hist **=** np**.**histogram**(**norm\_z\_vals**,** bins**=**32**,** range**=(**0**,** 1**))**

# TODO: Concatenate and normalize the histograms

hist\_features **=** np**.**concatenate**((**norm\_x\_hist**[**0**],** norm\_y\_hist**[**0**],** norm\_z\_hist**[**0**])).**astype**(**np**.**float64**)**

norm\_features **=** hist\_features **/** np**.**sum**(**hist\_features**)**

# Generate random features for demo mode.

# Replace normed\_features with your feature vector

#normed\_features = np.random.random(96)

#return normed\_features

**return** norm\_features

# Project Pipeline

## Model Training

roslaunch sensor\_stick training.launch

To train the system run the following sequence of commands

Roslaunch sensor\_stick training.launch

Rosrun sensor\_stick capture\_features.py

Rosrun sensor\_stick train\_svm.py

The first command launches a virtual environment. The second command exports a training set of data. The third command trains object names to their features.

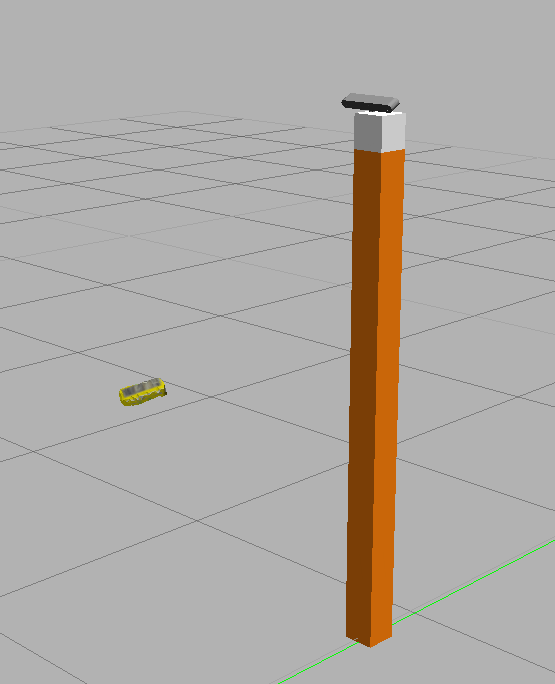


Figure 1: Random orientation of objects for data generation

### Accuracy

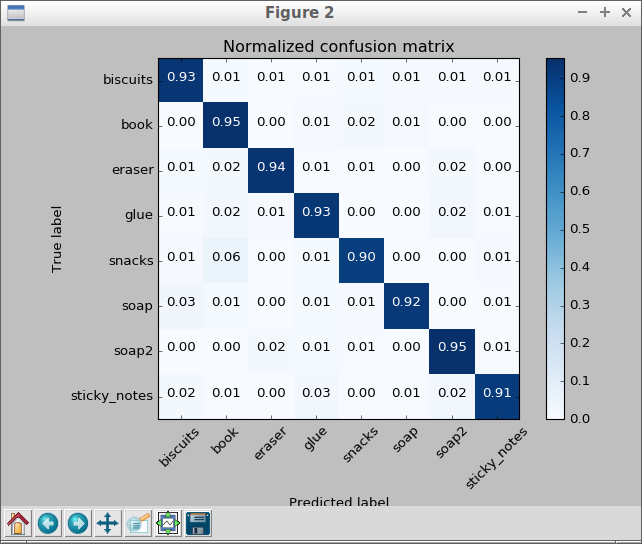
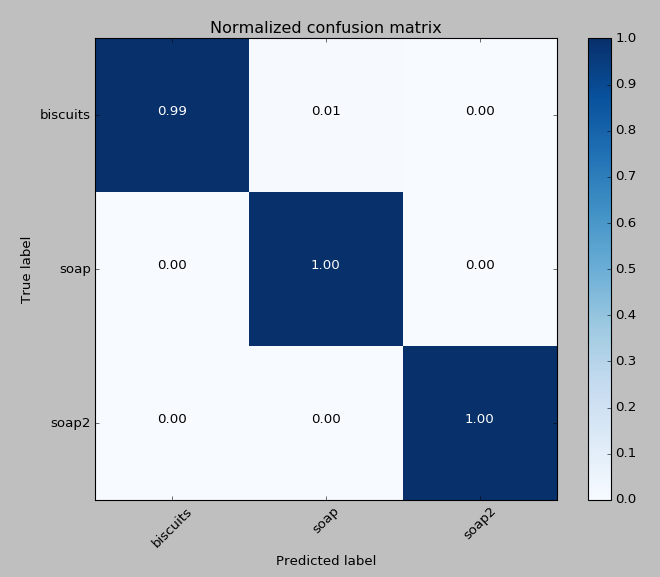
Increasing the number of samples for training improves the capability of classifier across the features of HSV and normal. The value of more training samples decreases exponentially, as the classifier performs with more and more precision.

Figure 2: Confusion Matrix with HSV and Normals as features; 400 training samples

Since each pick list challenge is separate, separate SVMs are trained for each challenge.



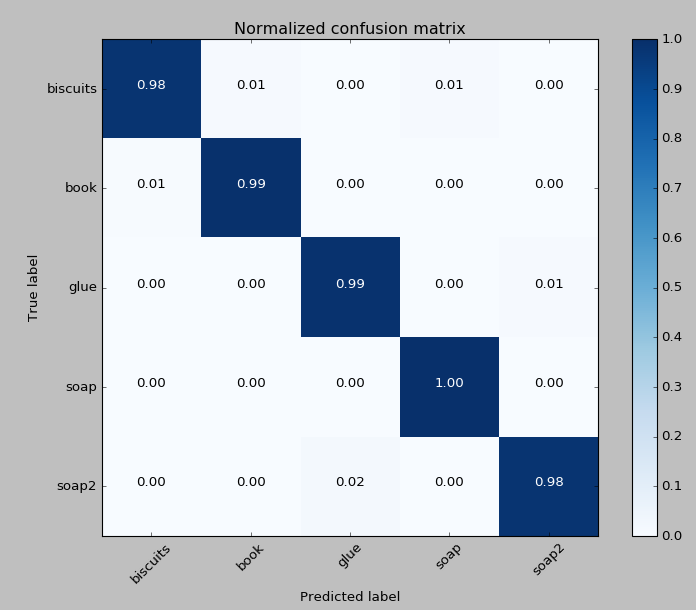


Figure 3: Training on Set 2 : 500 cycles

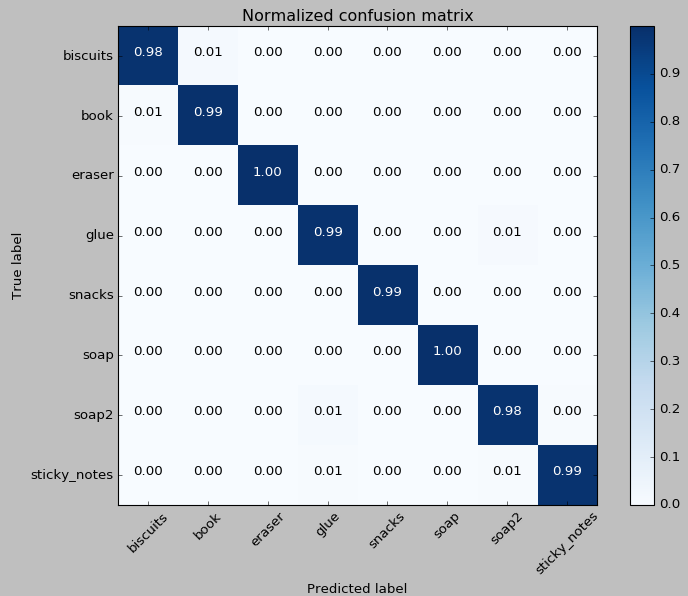


Figure 4: World 3 Trained on 1000 Cycles

# Project Performance

## Launch

The environment is launched with the following command:

roslaunch pr2\_robot pick\_place\_project.launch

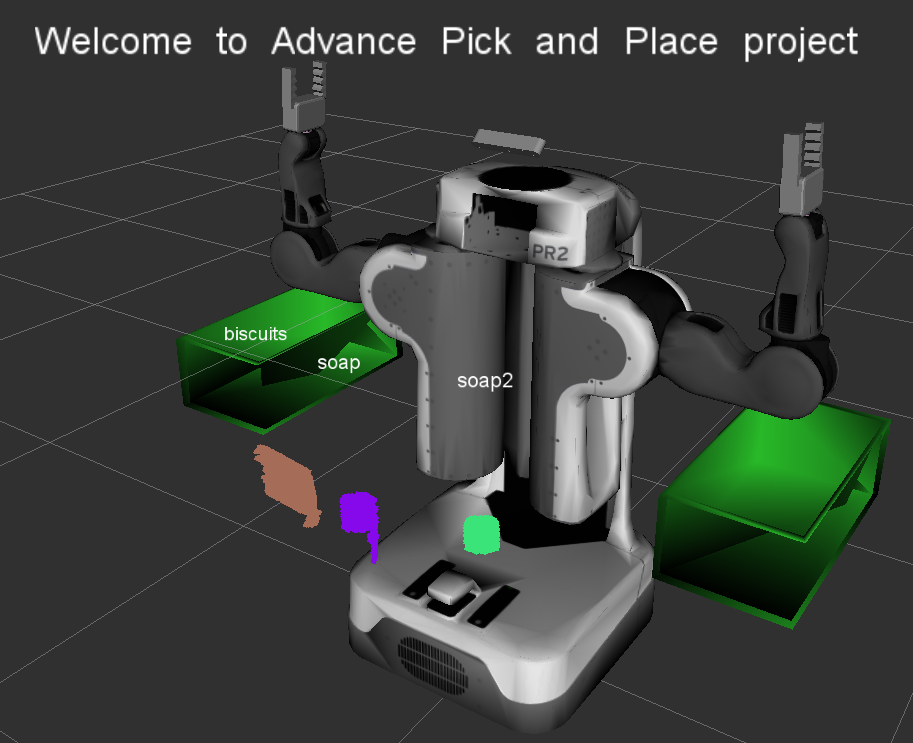
And the perception pipeline is launched with:

rosrun pr2\_robot project\_template.py

The different pick lists are identified by manipulating pr2\_robot/launch/pick\_place\_project.launch. The variables test{}.world and pick\_list\_{}.yaml are modified based off of condition 1, 2, and 3.

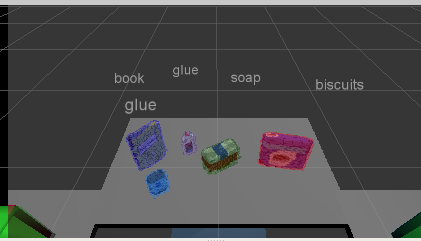
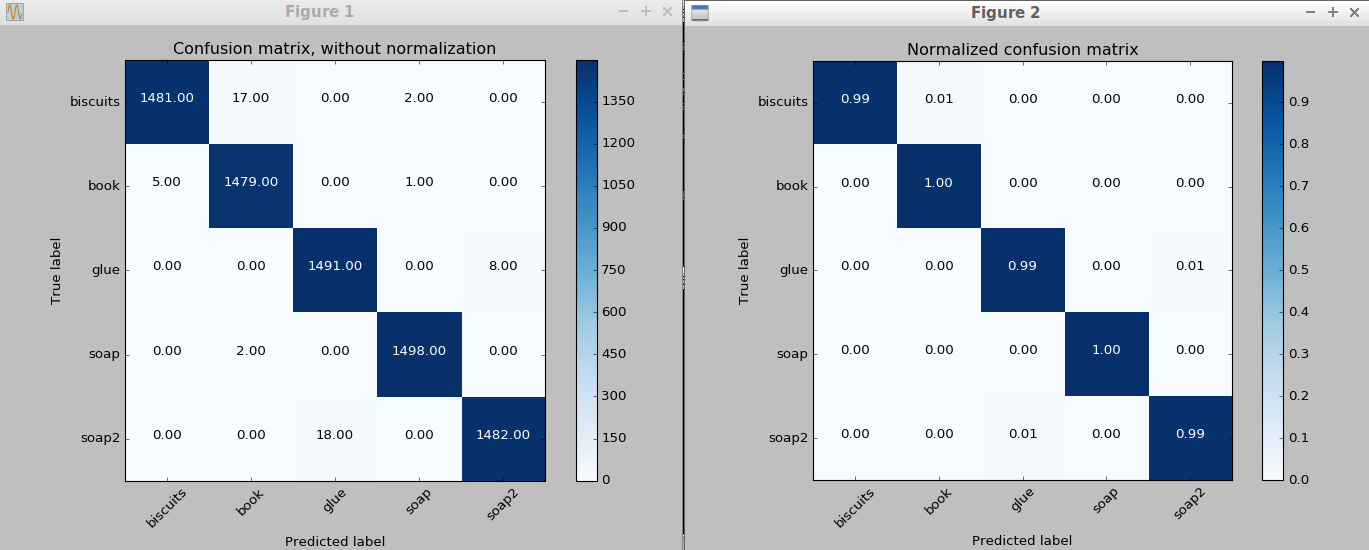
## Pick List 1

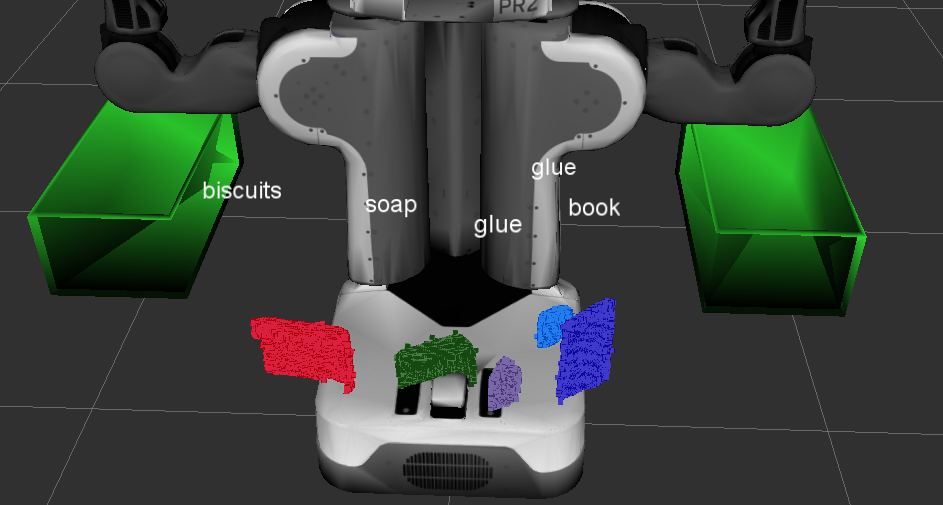
Pick list 1 loads biscuits, soap, and soap2 going into bins green, green, and red respectively. These were easily separated due to their large variance in hue.



## Pick List 2

The yaml for pick list 2 loads biscuits, soap, book, soap2, and glue. Going into bins green, green, red, red, and red respectively.

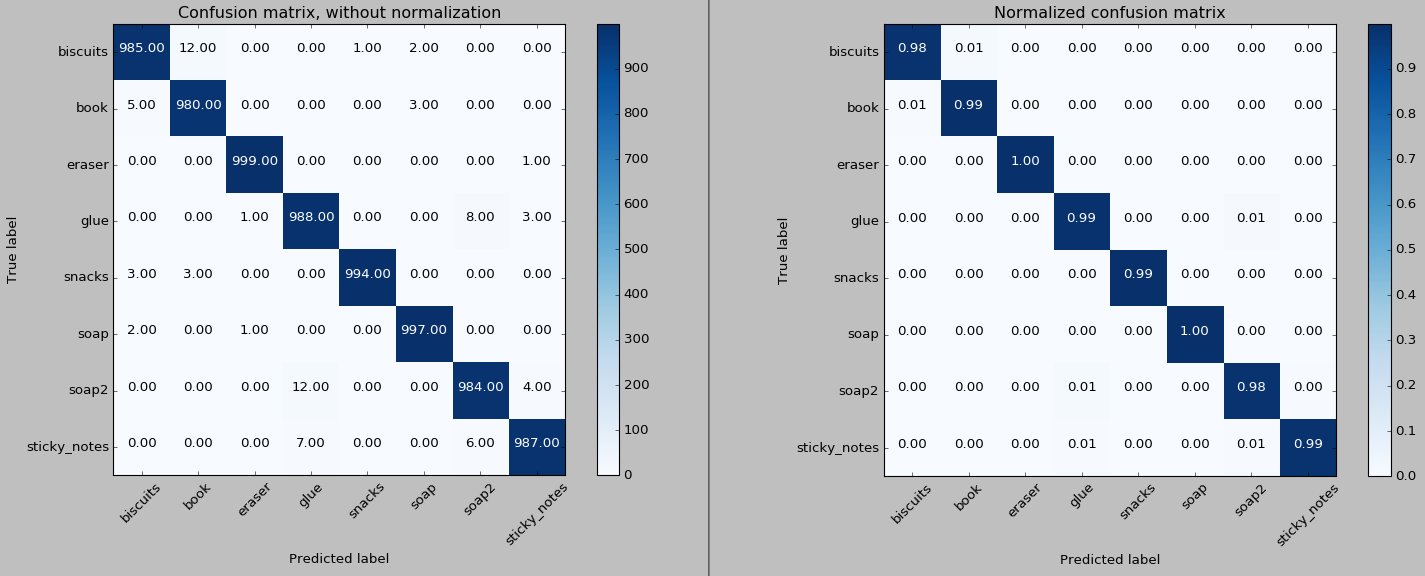


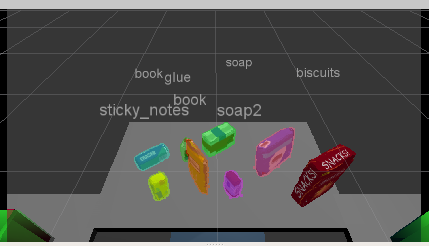


Here we can see we did not identify all of the objects correctly, as there are two “glue” items. Under conditions with less training, sometimes the book and biscuits are confused.

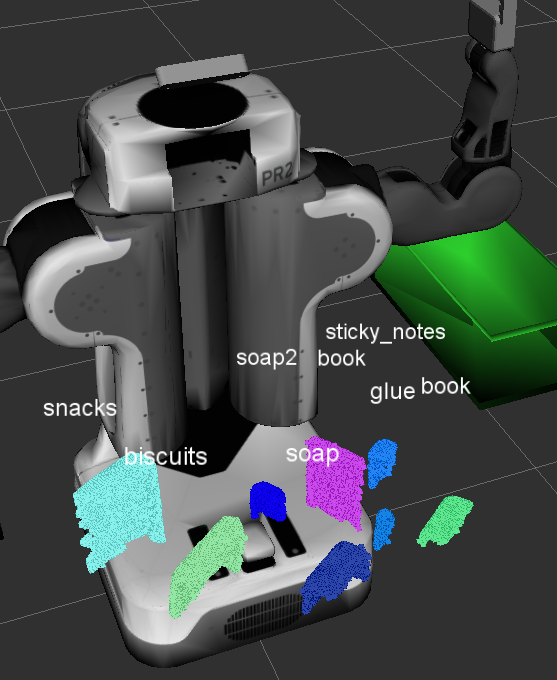
## Pick List 3

The yaml for pick list 3 includes sticky\_notes, book, snacks, biscuits, eraser, soap2, soap, and glue. The “snacks” is larger than any other item and did not get identified immediately due to too small of a cluster size. This had to be updated to prevent the cluster from being ignored.



At first “snacks” were not identified due to a clustering limit.

And then it was fixed here.



## Conclusion

This project reinforces the concept that more data improves classifiers. Support Vector machines successfully identify various objects. The classifiers are only as good as the information provided to them. RGB color space was transitioned to HSV to better discriminate between objects, which led to learning about L\*a\*b\* color space. Additional information could be used to improve the classifiers. In this project I added size information to the classifiers, but more advanced methods could include OCR, or neural networks to identify additional features.