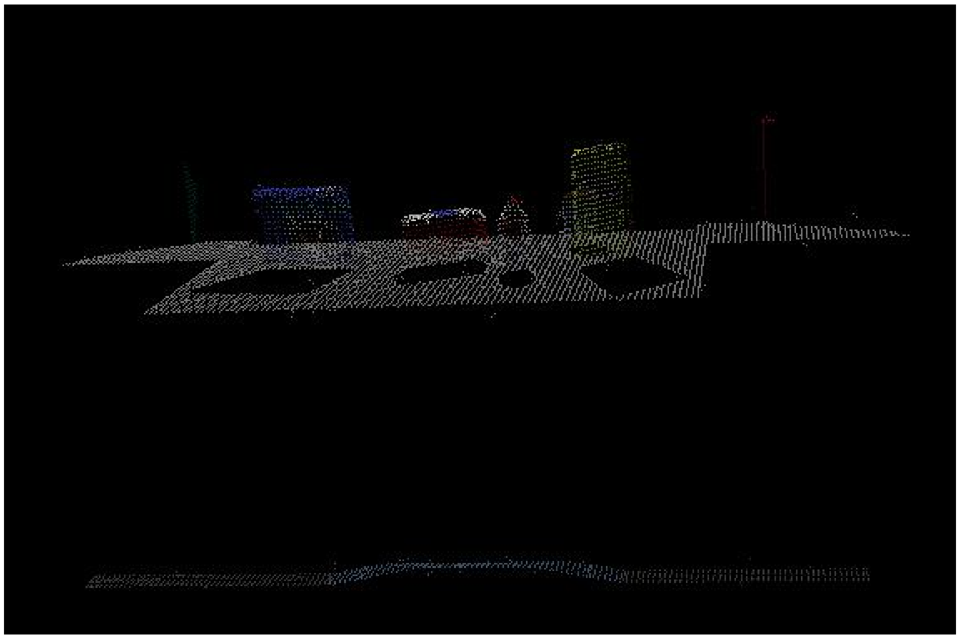
Project 3 - Object Recognition

Exercise 1

The purpose of exercise 1 was to use several filters as well as the RANSAC algorithm to isolate and remove the table from the environment, and to isolate the objects on the table. The outputs of this exercise were two pcd files, one that showed the isolated table and one that showed the isolated objects.

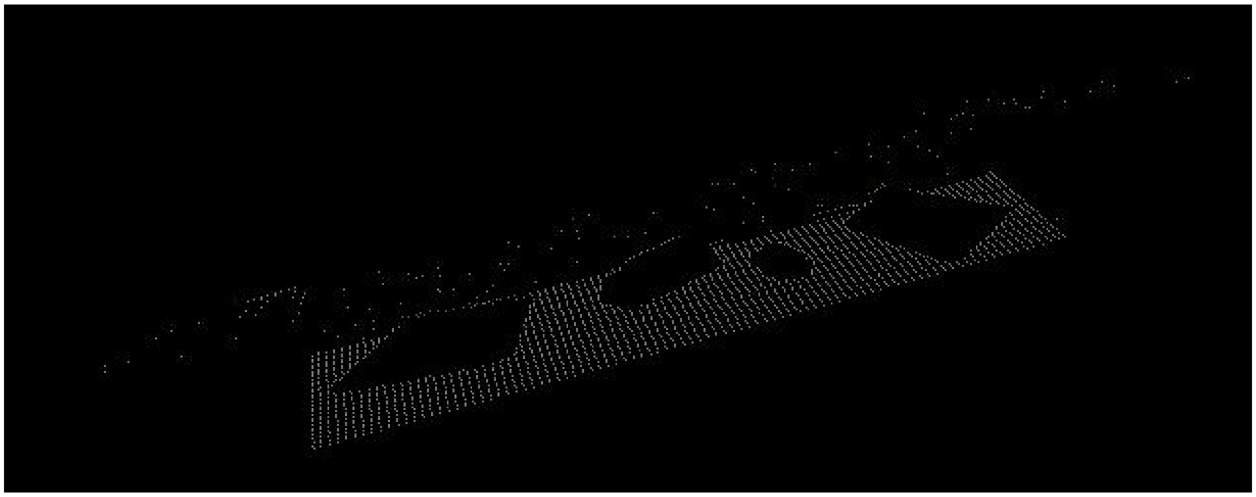
The first filter used was a voxel grid filter. In this project, an RGB-D camera is being used. It provides you pixel data regarding its red, green, and blue color values, as well as depth information, however, the point clouds in an RGB-D camera are quite dense so there is more information than required. We use the voxel grid filter to downsample the data by using a more sparsely sampled point cloud (the point cloud will have fewer points but will be sufficient enough to obtain information on the scene). This allows for faster computations without any loss in being able to identify objects in the point cloud (point clouds are digital representations of 3D objects, ie our data). In this filter, the word voxel is short for volume element. The voxel grid filter will take a spatial average of the points in the cloud confined by each voxel. For this project, I used a leaf size of 0.01 (this is the voxel size in meters)

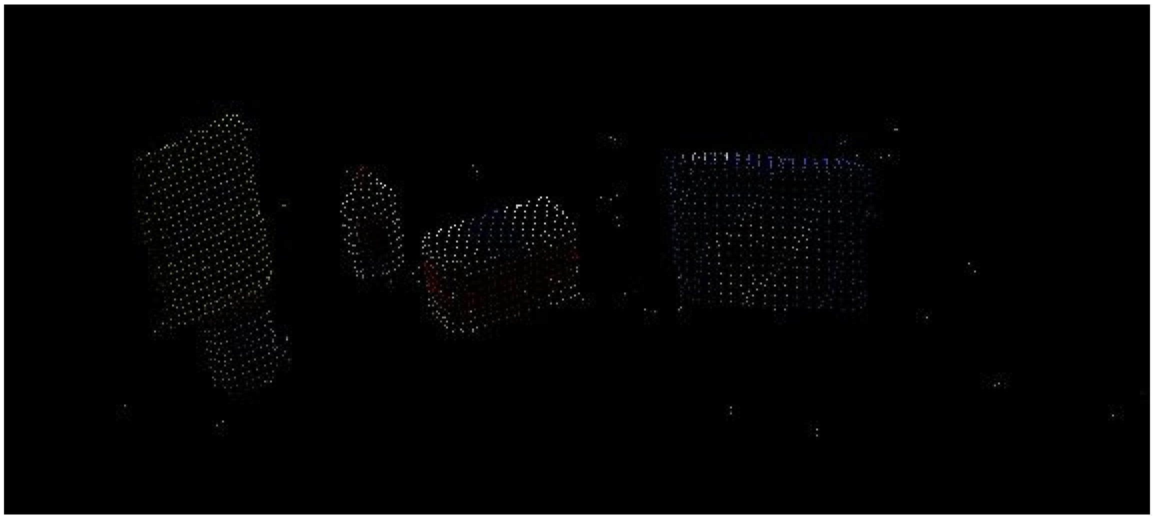


The next filter used is the passthrough filter which creates a region of interest. This region of interest allows us to crop out the areas of the scene we are not interested in (like the portion below the table), and keep the area in the scene that is useful for us (the tabletop and everything on the table). Since the robot and tabletop are stationary in this scene, we can use this filter to easily remove unwanted areas. We apply the pass through filter along the z axis, with a minimum axis of 0.6 and a maximum axis of 1.1. So anything below 0.6 will be ignored and anything above 1.1 will be ignored. We need to take care that we don’t remove the tabletop itself.

Aside: In the project, I was detecting too far along the y-axis, so my object recognition picked up the red and green box as objects in the scene. I tried to create a y-axis pass through filter to limit the scene to just the objects, however, I couldn't get this to work correctly. I kept my code in to show my thinking though.

The last algorithm used is RANSAC. RANSAC stands for Random Sample Consensus, and it's used to identify and remove the tabletop from the scene. RANSAC is used when the dataset contains points that belong to a particular model. Since the tabletop is a planar object, we can use a plane model to eliminate it. In order to fit the model, we need enough inliers, and we need to ignore the outliers (inliers fit the model, outliers do not fit the model), so any table point clouds will become inliers, and anything else will become outliers. I set a max distance of 0.01, which is the maximum distance for a point to be considered fitting the model. Using RANSAC, exercise 1 is completed by producing a cloud_table.pcd and a cloud_objects.pcd.

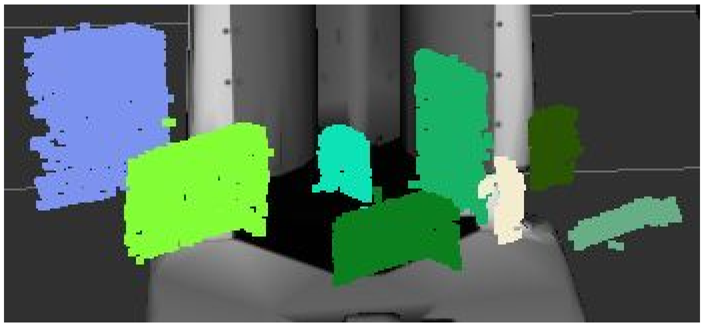




Exercise 2
The purpose of the exercise 2 is to use the Euclidean Clustering algorithm to segment the remaining points into individual objects. If we only rely on RANSAC, then false positives can be produced since it only relies on a shape model. A soda can looks similar to a beer can so it can mix these two objects up if only considering shape. Not only that, but you'd have to run through the entire point cloud several times for each model shape, which is not optimal.

Euclidean clustering creates clusters by grouping data points that are within some threshold distance from the nearest neighbor in the data. The first step of this algorithm is to construct a k-d tree from the cloud_objects point cloud. For this, we require point cloud data with only spatial information, so first we convert from color/space to just space using the XYZRGB_to_XYZ() helper function. Next we construct the k-d tree using the make_kdtree() function, and we also create a cluster extraction object using make_EuclideanClusterExtraction() function.

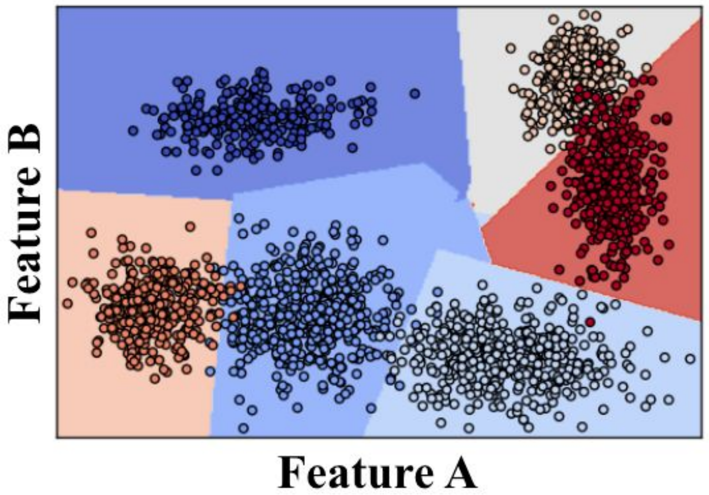
Next, we need to set the cluster tolerance, the minimum cluster size, and the maximum cluster size. The cluster tolerance is the max distance that a point cloud can be to its nearest neighbor while still remaining a part of the cluster. The tolerance I set was 0.05, which was large enough that points within each object are part of the cluster but small enough that the distance between this object and the next object is not within this tolerance (so two objects are not considered as one object). The minimum cluster size is the min number of points needed to create a cluster. We need to be aware of noise, so we should not create a min cluster size large enough that noise in the data is considered part of the cluster. Lastly, the max cluster size is the max number of points that create a cluster. This needs to be large enough that we don't exclude any useful point cloud data.



The final point cloud called cluster_cloud contains points for each of the segmented objects, with each set of points having a unique color. I knew that the clustering values were incorrect if there were any objects that didn't have it's own unique color. If that happened, two or more objects were considered one object and would be misclassified.

Exercise 3

At this point, we have removed the tabletop, and have individually identified each of the objects. The next step is to apply a supervised machine learning algorithm called Support Vector Machine (SVM) that allows us to create discrete classes of the dataset. With this, we will be able to classify objects. The SVM applies an iterative method to a training dataset where each item in the training set is characterized by a feature vector and a label. A feature is something we use to describe the item, and once we have these features we can train an object classifier to identify certain objects based on these features. In our case, the training dataset consists of the objects we have in our 3 different test worlds. In the below image, there are two features A and B. The color of each point corresponds to its label (the object in the scene it represents).



The training set is split by decision boundaries (colored polygons), so when we have a new object that has no label yet, we can determine which class the object belongs to based on the features and SVM training.

For this project, we used the color and surface normal as feature vectors. As mentioned before, each point cloud contains color information and shape information, so when training the object classifier, it makes sense to identify features based on color and shape. The color information contained in a point cloud is based on RGB colors, however, RGB is not the best color representation for perception tasks because objects can have a very different color based on the different lighting conditions. So we convert this color data to HSV which is less sensitive to lighting changes. Hue is the color in the pixel, saturation is the intensity of that color, and value is the overall brightness. So if you darken an RGB image, the HSV image will still remain bright, so it’s easier for us to obtain features.

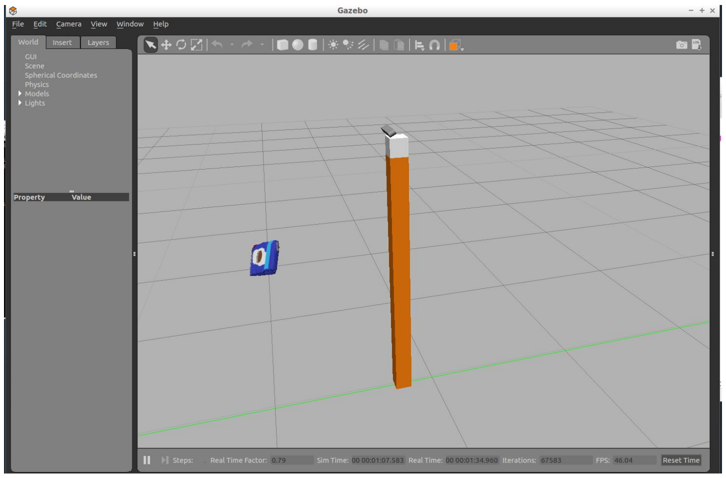
To create a color histogram, we split up the data values into bins from 0 to 255, then count up how many values fall into each bin. Then when we compare the color histogram of a known object to a test object, locations with a similar color distribution will provide a good match. To create a surface normal histogram, you generate normal vectors across the object and then take the distribution of these normal vectors, which tells us the shape of the object. Again, we compare this data with a test object to determine how much they match.

In order to train the SVM, we loop through different views of each object, and for each view the SVM determines the features and then gives a label based on which boundaries it falls between.

Project

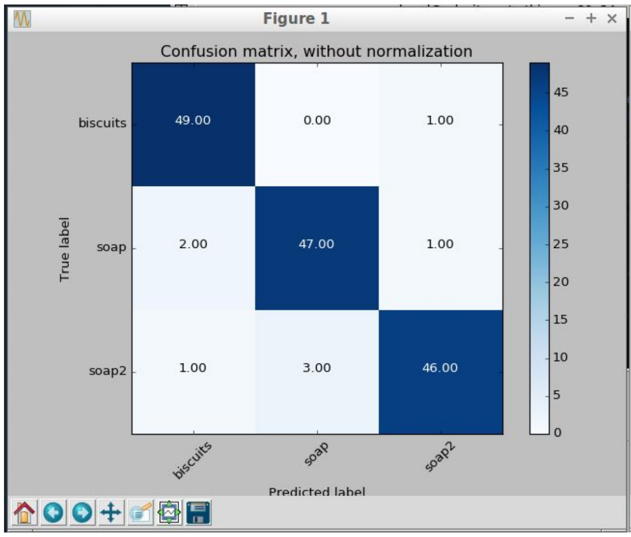
The above 3 exercises were used to build the project, however, the project introduced noise in the environment so a statistical outlier filter was used to filter out the noise. The camera used in this project displays noise due to external factors (ex dust, light sources, etc). The statistical outlier filter computes the distance for each point cloud to all of its neighbors, and then calculates a mean distance. By assuming a Gaussian distribution, all points whose mean distances fall outside of an interval defined by the mean distance plus some standard deviation are considered to be outliers and removed from the point cloud.

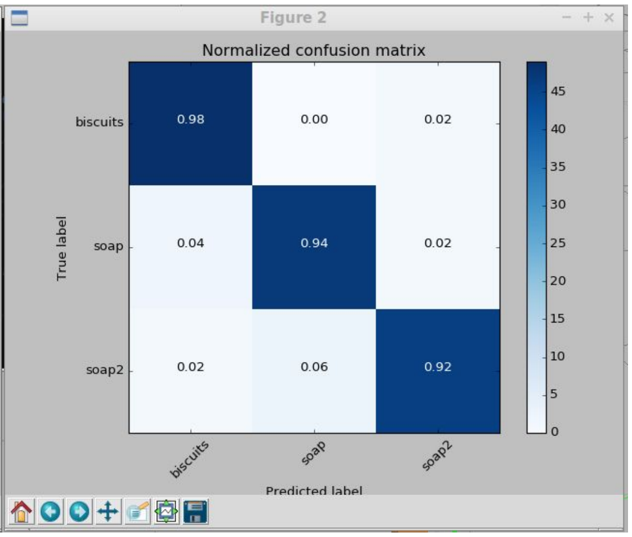
Once all code was completed for the project, I went about training the SVM. I edited the capture_features.py script used in exercise 3. This script contains the different models that appear in the environment. The first test environment, “test1”, contains a biscuit package, and two different types of soap. The script will loop through each item and will extract the features of that item; I initially set the loop to run through 50 times. The more times we loop through each item, the more accurate feature information we will obtain.



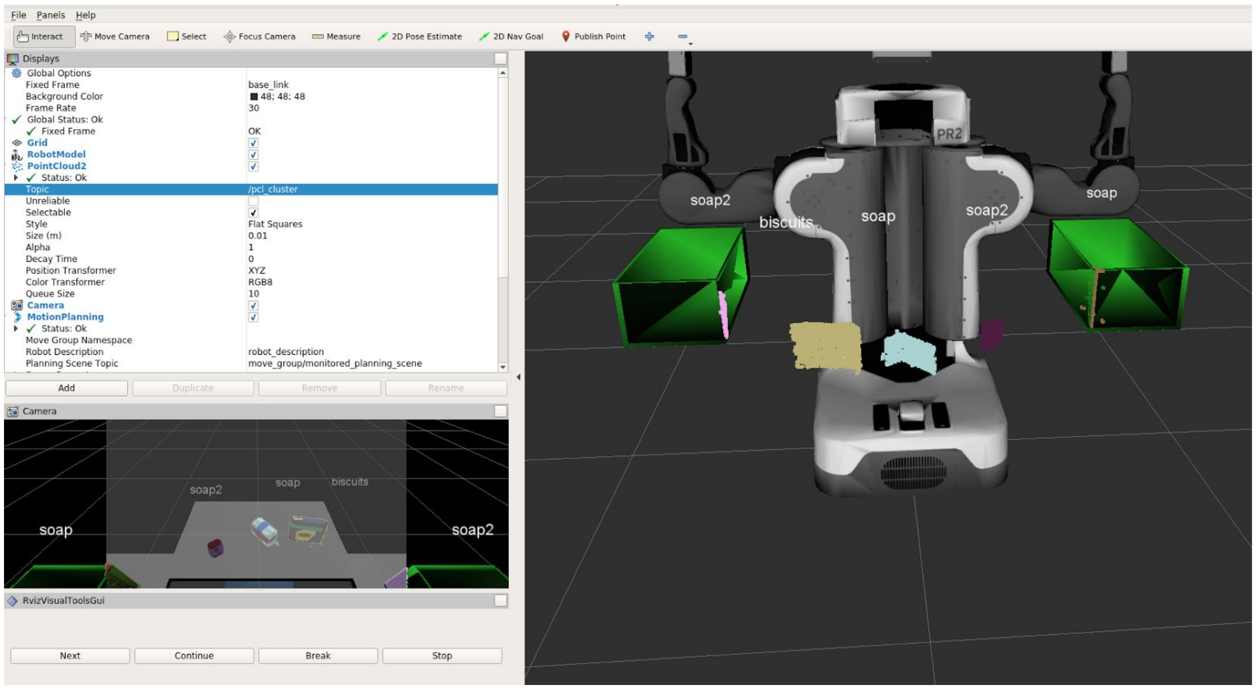
The output of the training is a training_set.sav file. Once this is generated for all 3 test environment, we can generate two different plots for each world. The first plot shows the number of times each item was classified correctly and incorrectly. The second plot shows the normalized accuracy plot as a percentage of the total. The biscuits were correct 98% of the time, the first soap was correct 94% of the time and the second soap was correct 92% of the time. The number of features in this training set was 150, and overall average was 95%.

Test1 Confusion Matrix



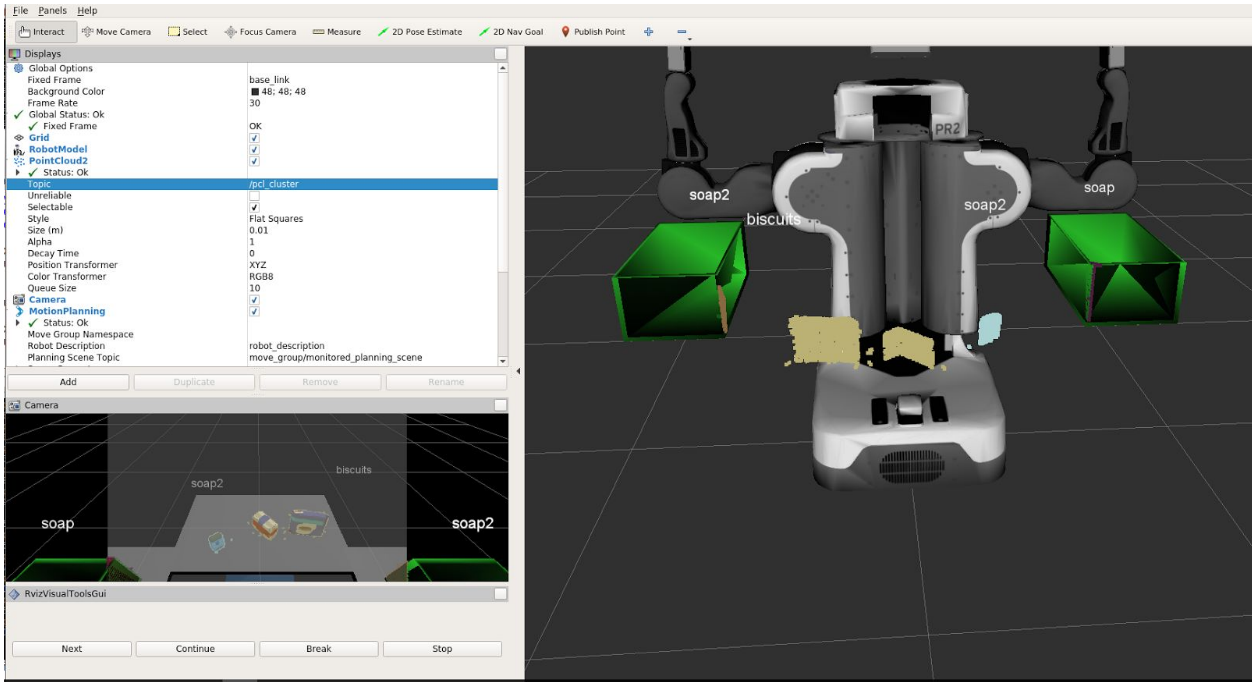


Since for Test1 world I had a 94% accuracy, I was able to correctly label the soap, soap2, and biscuits most of the time.



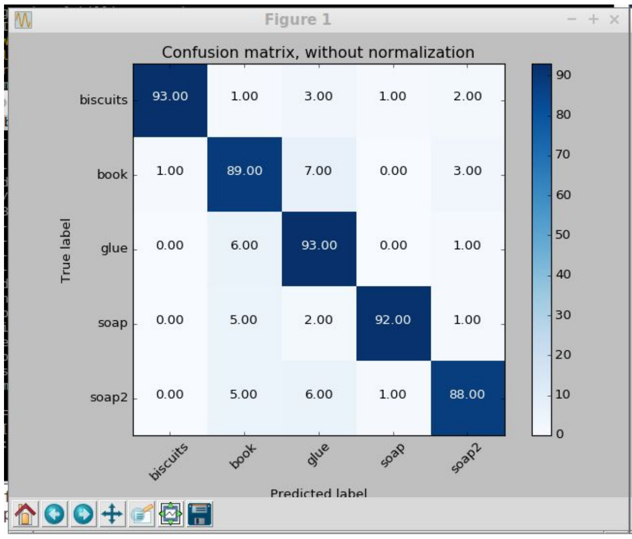
Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
PointCloud2	✓
✓ Status: Ok	
Topic	/pcl_cluster
Unreliable	
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
Queue Size	10
Camera	✓
MotionPlanning	✓
✓ Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitored_planning_scene

However, every once in awhile, I had an incorrect labeling where the biscuits and soap would be considered one item, and only the biscuit and soap2 label would appear.

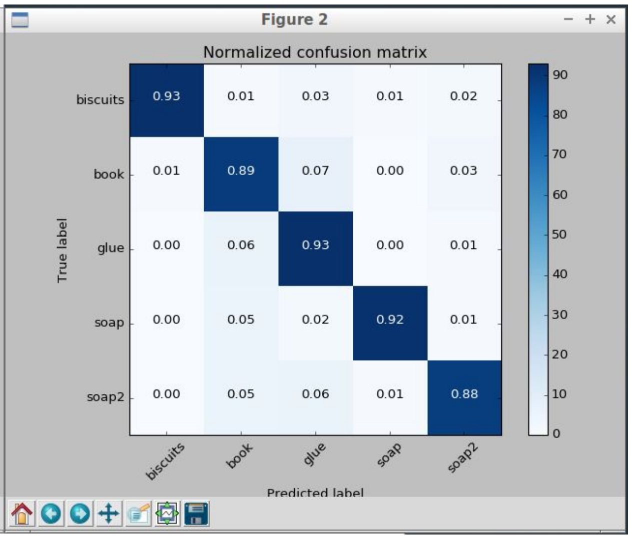


Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
PointCloud2	✓
✓ Status: Ok	
Topic	/pcl_cluster
Unreliable	
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
Queue Size	10
Camera	✓
MotionPlanning	✓
✓ Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitored_planning_scene

Test2 Confusion Matrix

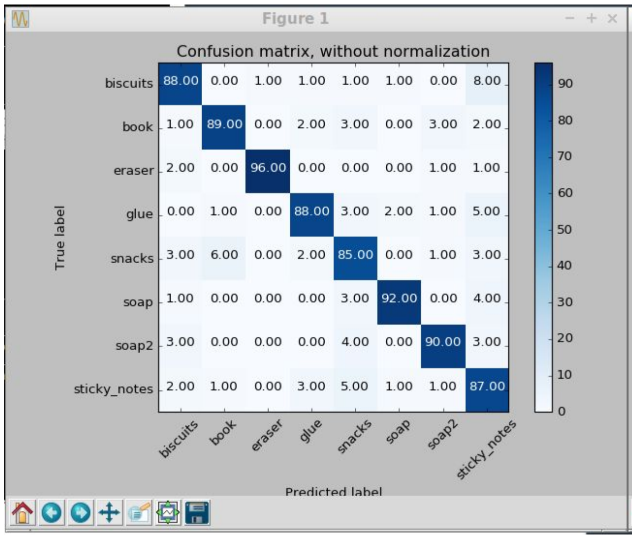


	biscuits	book	glue	soap	soap2
biscuits	93.00	1.00	3.00	1.00	2.00
book	1.00	89.00	7.00	0.00	3.00
glue	0.00	6.00	93.00	0.00	1.00
soap	0.00	5.00	2.00	92.00	1.00
soap2	0.00	5.00	6.00	1.00	88.00

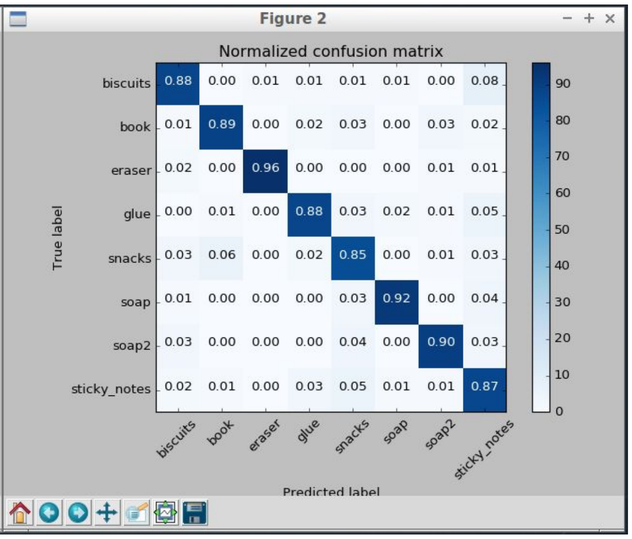


	biscuits	book	glue	soap	soap2
biscuits	0.93	0.01	0.03	0.01	0.02
book	0.01	0.89	0.07	0.00	0.03
glue	0.00	0.06	0.93	0.00	0.01
soap	0.00	0.05	0.02	0.92	0.01
soap2	0.00	0.05	0.06	0.01	0.88

Test3 Confusion Matrix



	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	88.00	0.00	1.00	1.00	1.00	1.00	0.00	8.00
book	1.00	89.00	0.00	2.00	3.00	0.00	3.00	2.00
eraser	2.00	0.00	96.00	0.00	0.00	0.00	1.00	1.00
glue	0.00	1.00	0.00	88.00	3.00	2.00	1.00	5.00
snacks	3.00	6.00	0.00	2.00	85.00	0.00	1.00	3.00
soap	1.00	0.00	0.00	0.00	3.00	92.00	0.00	4.00
soap2	3.00	0.00	0.00	0.00	4.00	0.00	90.00	3.00
sticky_notes	2.00	1.00	0.00	3.00	5.00	1.00	1.00	87.00



	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	0.88	0.00	0.01	0.01	0.01	0.01	0.00	0.08
book	0.01	0.89	0.00	0.02	0.03	0.00	0.03	0.02
eraser	0.02	0.00	0.96	0.00	0.00	0.00	0.01	0.01
glue	0.00	0.01	0.00	0.88	0.03	0.02	0.01	0.05
snacks	0.03	0.06	0.00	0.02	0.85	0.00	0.01	0.03
soap	0.01	0.00	0.00	0.00	0.03	0.92	0.00	0.04
soap2	0.03	0.00	0.00	0.00	0.04	0.00	0.90	0.03
sticky_notes	0.02	0.01	0.00	0.03	0.05	0.01	0.01	0.87

The last portion of the project that was required was to output the information to yaml files. We are provided a pick list of objects for each of the 3 test environments, and these pick lists exist as yaml files. This file contains an ordered object list that consists of the name of each item and the color of the box the item belongs to.

In the pr2_mover function, I first initialize variables for each of the ROS messages, as well as a few other variables. The message types can be found in the pr2_robot/srv directory in the PickPlace.srv file. We have 5 messages to initialize:

- test_scene_num: the number of the test scene we want to load in the environment
- object_name: the name of the object which is obtained from the pick list
- arm_name: the name of the arm (left or right)

- pick_pose: the calculated pose of the object’s centroid
- place_pose: the objects placement pose

Next, I need to read in the parameters from the parameter server. I get the object list as well as the bins in the environment. Then I create one large for loop that loops through the objects found in the environment. For each object found, we have to determine the corresponding label, it’s position, which arm will pick the object, and which bin the object will be placed in. After all these are determined, I create a yaml dictionary, and send the corresponding information to output yaml files.

Future Improvements

I was unable to complete the challenges. I’d like to be able to at least have the robot pick the objects and place them in the correct bins, however, I was unable to get this to work. Another improvement that can be made is in regards to training the SVM. If we run through more iterations while training the SVM (maybe up to 1000), we can get more accurate results. This takes time to do, however, I’d like to see how accurate I can make this. Lastly, my object recognition would recognize the two bins as objects since the y direction plane was not correctly clipped. I attempted to clip the plane, however, couldn’t successfully do this. I’d like to still improve on this.