Perception Project

Liev Birman

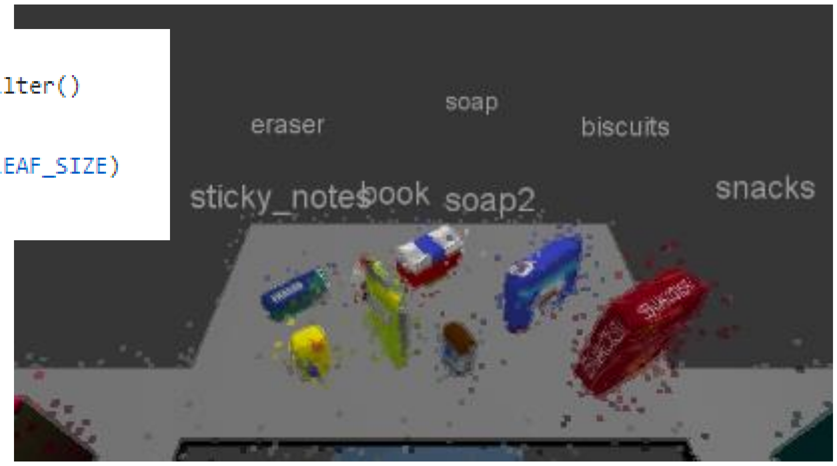
Summary

In this project we filtered and segmented RGBD point cloud data. Then we used models to train a Support Vector Machine to identify the objects in our field-of-view. Finally, we fed the position of these objects into a .yaml file which can be used by the pr2 robot to perform an ordered pick and place operation.

Pipeline Steps

1. **Downsampling**. This is the process of averaging data under the assumption that the proper sample size will still contain enough information to accomplish the perception task. In this task, 1cm was an appropriate scale unit, although I suspect the task could've been accomplished with a slightly larger voxel size.

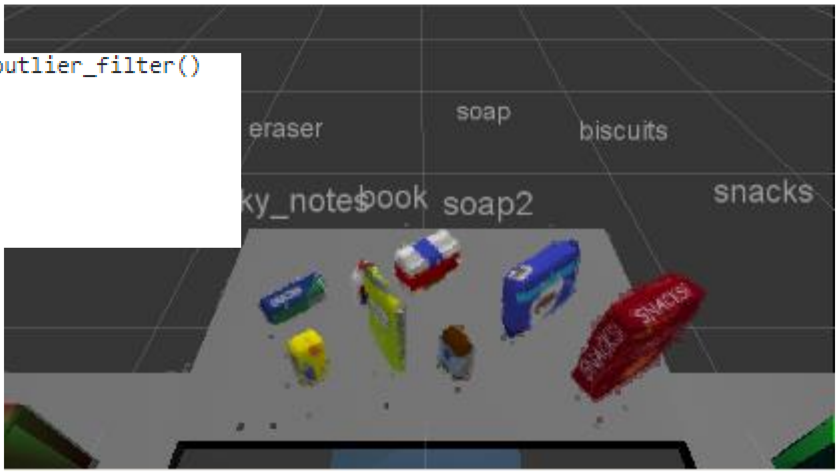
```
# TODO: Voxel Grid Downsampling
vox = cloud_filtered.make_voxel_grid_filter()
LEAF_SIZE = 0.01
vox.set_leaf_size(LEAF_SIZE,LEAF_SIZE,LEAF_SIZE)
cloud_filtered = vox.filter()
```



You can see here that the geometry of our items are recapitulated, however, the glue hidden behind the book is not identified. This likely an issue of clustering tolerance.

2. **Outlier Filtering** is the process of averaging voxels to determine whether any outliers exist within some range. Here we chose that range to be 10cm which seems to have done the trick.

```
outlier_filter = cloud.make_statistical_outlier_filter()
outlier_filter.set_mean_k(50)
x = 0.1
outlier_filter.set_std_dev_mul_thresh(x)
cloud_filtered = outlier_filter.filter()
```

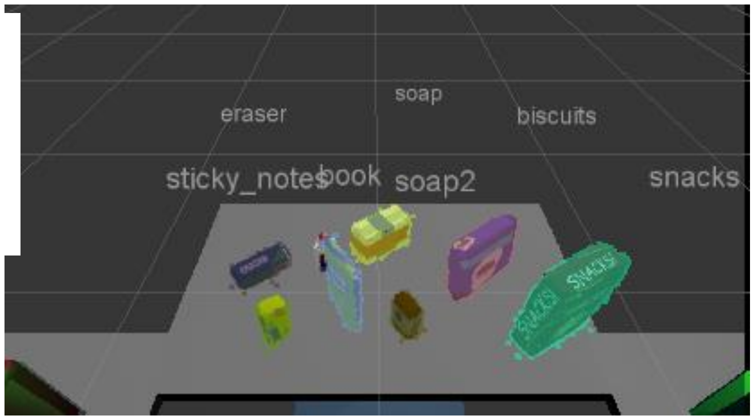


3. **Passthrough Filtering** is simply taking all data point above and below some distance threshold and cutting them out. We removed points along the z axis to remove the table's stand as well as along the x axis to remove the bins in the lower portion of the robot's field-of-view. Had to cut it really close with the table as the overhand was being recognized as an object in the later segmentation step. In the case where information about the environment is not known beforehand, it might be very difficult to apply this filter.

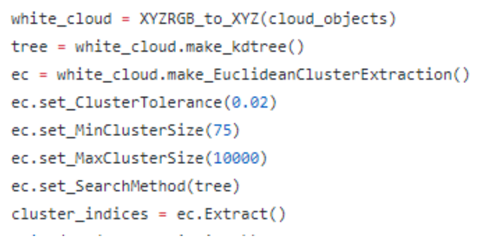
```
passthrough = cloud_filtered.make_passthrough_filter
filter_axis = 'z'
passthrough.set_filter_field_name(filter_axis)
axis_min=0.605
axis_max=1.1
passthrough.set_filter_limits(axis_min,axis_max)
cloud_filtered = passthrough.filter()

passthrough = cloud_filtered.make_passthrough_filter
filter_axis = 'x'
passthrough.set_filter_field_name(filter_axis)
axis_min=0.35
axis_max=1.1
passthrough.set_filter_limits(axis_min,axis_max)
cloud_filtered = passthrough.filter()
```

4. **RANSAC** plane segmentation is the process of fitting a model to a data set in order to identify data points which fit said model. These points can then be removed from the processing task, thus further increasing our efficiency. A tolerance needs to be set, in this case, that's in the form of a distance. Applying this algorithm successfully removed the table as is evident by the lack of extra recognized objects in the upcoming results. This could be very useful as a base for segmentation when you're looking for objects with very similar features, such as a single type of leaf on a tree.

```
seg = cloud_filtered.make_segmenter()
seg.set_model_type(pcl.SACMODEL_PLANE)
seg.set_method_type(pcl.SAC_RANSAC)
max_distance=0.01
seg.set_distance_threshold(max_distance)
inliers,coefficients = seg.segment()
```

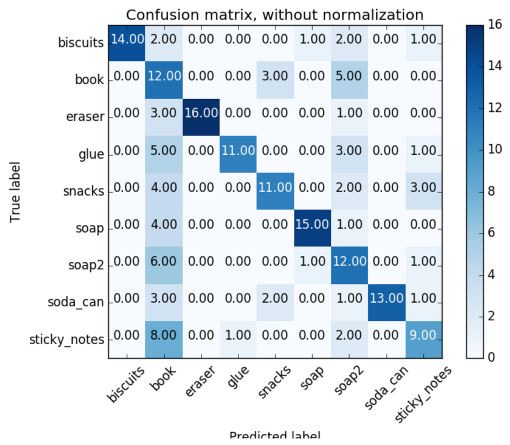


5. **Euclidean clustering** is the process of clustering data together based on the Euclidean distance of one data point to the other. It's very useful when the data is distinct within its n-dimensional parameter space. In this case, a good choice for a distance metric works because none of our objects are touching. If it so happened that two objects were side-by-side and within 2 cm of each other, my particular settings wouldn't work and identification becomes considerably more difficult as distance decreases and more resolution is needed.

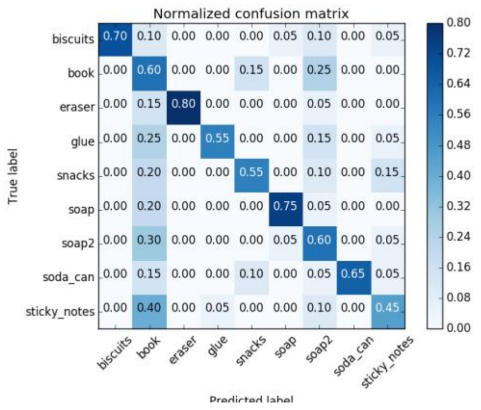


```
white_cloud = XYZRGB_to_XYZ(cloud_objects)
tree = white_cloud.make_kdtree()
ec = white_cloud.make_EuclideanClusterExtraction()
ec.set_ClusterTolerance(0.02)
ec.set_MinClusterSize(75)
ec.set_MaxClusterSize(10000)
ec.set_SearchMethod(tree)
cluster_indices = ec.Extract()
```

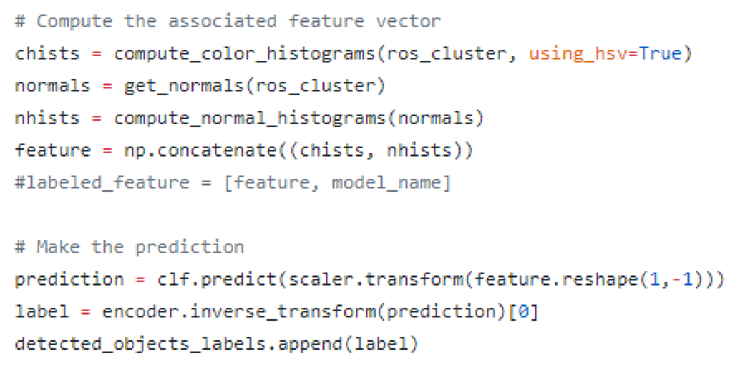
6. **SVM based object recognition** is an algorithm that creates a gap between clusters of distinct features within a data set. It is a type of supervised learning algorithm, meaning you need to feed it known examples, and it will return an inferred function based on this initial data which can be used to map new examples. In this case I fed the model 20 instances per object, using HSV coloring, and an rbf kernel.



	biscuits	book	eraser	glue	snacks	soap	soap2	soda_can	sticky_notes
biscuits	14.00	2.00	0.00	0.00	0.00	1.00	2.00	0.00	1.00
book	0.00	12.00	0.00	0.00	3.00	0.00	5.00	0.00	0.00
eraser	0.00	3.00	16.00	0.00	0.00	0.00	1.00	0.00	0.00
glue	0.00	5.00	0.00	11.00	0.00	0.00	3.00	0.00	1.00
snacks	0.00	4.00	0.00	0.00	11.00	0.00	2.00	0.00	3.00
soap	0.00	4.00	0.00	0.00	0.00	15.00	1.00	0.00	0.00
soap2	0.00	6.00	0.00	0.00	0.00	1.00	12.00	0.00	1.00
soda_can	0.00	3.00	0.00	0.00	2.00	0.00	1.00	13.00	1.00
sticky_notes	0.00	8.00	0.00	1.00	0.00	0.00	2.00	0.00	9.00



	biscuits	book	eraser	glue	snacks	soap	soap2	soda_can	sticky_notes
biscuits	0.70	0.10	0.00	0.00	0.00	0.05	0.10	0.00	0.05
book	0.00	0.60	0.00	0.00	0.15	0.00	0.25	0.00	0.00
eraser	0.00	0.15	0.80	0.00	0.00	0.00	0.05	0.00	0.00
glue	0.00	0.25	0.00	0.55	0.00	0.00	0.15	0.00	0.05
snacks	0.00	0.20	0.00	0.00	0.55	0.00	0.10	0.00	0.15
soap	0.00	0.20	0.00	0.00	0.00	0.75	0.05	0.00	0.00
soap2	0.00	0.30	0.00	0.00	0.00	0.05	0.60	0.00	0.05
soda_can	0.00	0.15	0.00	0.00	0.10	0.00	0.05	0.65	0.05
sticky_notes	0.00	0.40	0.00	0.05	0.00	0.00	0.10	0.00	0.45

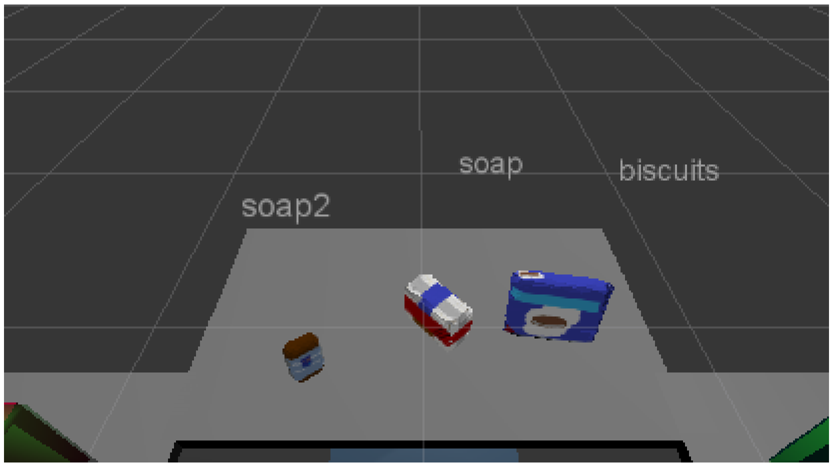


```
# Compute the associated feature vector
chists = compute_color_histograms(ros_cluster, using_hsv=True)
normals = get_normals(ros_cluster)
nhists = compute_normal_histograms(normals)
feature = np.concatenate((chists, nhists))
#labeled_feature = [feature, model_name]

# Make the prediction
prediction = clf.predict(scaler.transform(feature.reshape(1,-1)))
label = encoder.inverse_transform(prediction)[0]
detected_objects_labels.append(label)
```

This SVM has relatively poor performance as the best classification rate is only 75%. It performed better than 75% in our trials but this is just due to the convenient orientation. If this robot is to be of commercial use it has a long way to go in terms of optimization. Training with more examples and choosing finer clustering parameters could be a step towards this.

Results

Pick List 1

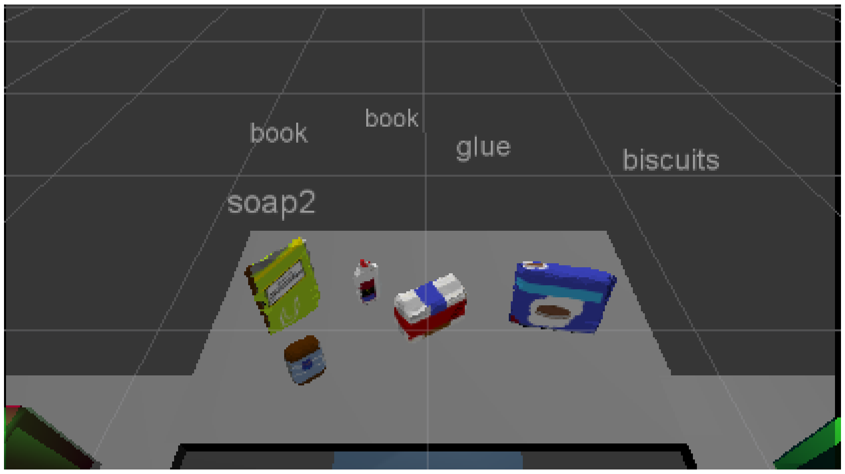


Soap2

Soap

Biscuits

Pick List 2



Biscuits

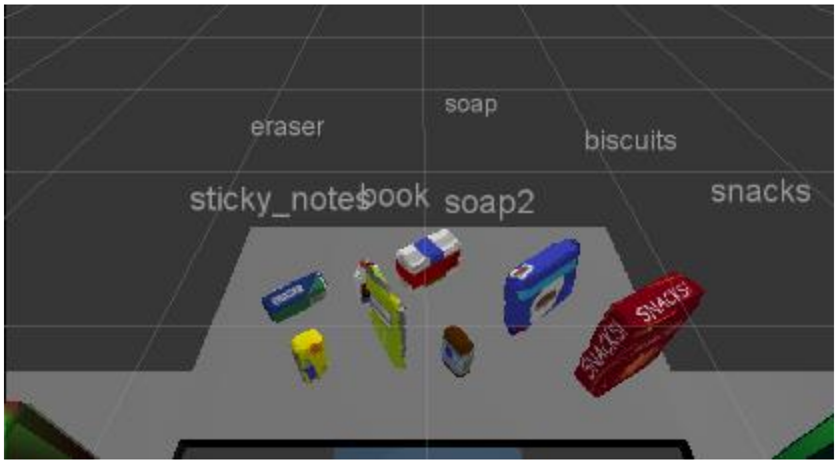
Soap

Book

Soap2

Glue

Pick List 3



Sticky Notes

Book

Snacks

Biscuits

Eraser

Soap2

Soap

Glue