Project 3D Perception

1. Writeup

Criteria	Meets Specifications	Result
Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. The project repository contains a template writeup for this project that you can use as a guide and a starting point.	The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled. The writeup should include a discussion of what worked, what didn't and how the project implementation could be improved going forward.	Each rubric item addressed with code, relevant explanation and output. Object recognition was implemented successfully to meet criteria 3. However, I did not understand the mechanism of calling pr2_mover. A call was made from the pcl_callback function to obtain the output_n.yml files. Pick and place of the objects using the PR2 robot couldn't be completed on time.

2. Exercise 1, 2 and 3 pipeline implemented

2.1. Criteria 2.1

Criteria	Meets Specifications	Result
Complete Exercise 1 steps. Pipeline for filtering and RANSAC plane fitting implemented.	the pcl_callback() function within within the template Python script has been filled out to include filtering and RANSAC plane fitting. Not required, but to help your reviewer consider adding screenshots of output at different steps in your writeup with brief explanations.	RANSAC plane fitting implemented in pcl_callback function.

Pipeline for filtering:

1. Outlier removal

The outlier filter helps to remove the noise in the RGBD sensor. The outlier filter performs a statistical analysis in the neighborhood of each point, and removes those points which do not meet a certain criteria. PCL's StatisticalOutlierRemoval filter below is 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. The optimum value for scale factor X was determined through experimentation to be 0.1

```
#-------------------------------------------------------------------------------------------
# Outlier removal filter
#-------------------------------------------------------------------------------------------
    # TODO: Convert ROS msg to PCL data
    cloudinc = ros_to_pcl(pcl_msg)

    # Outlier removal filter

    # Much like the previous filters, we start by creating a filter object:
    outlier_filter = cloudinc.make_statistical_outlier_filter()

    # Set the number of neighboring points to analyze for any given point
    outlier_filter.set_mean_k(10) #10 - cluster was successful but lot of noise

    # Set threshold scale factor
    x = 0.1

    # Any point with a mean distance larger than global (mean distance+x*std_dev) will be considered outlier
    outlier_filter.set_std_dev_mul_thresh(x)

    # Finally call the filter function for magic
    cloud = outlier_filter.filter()
```

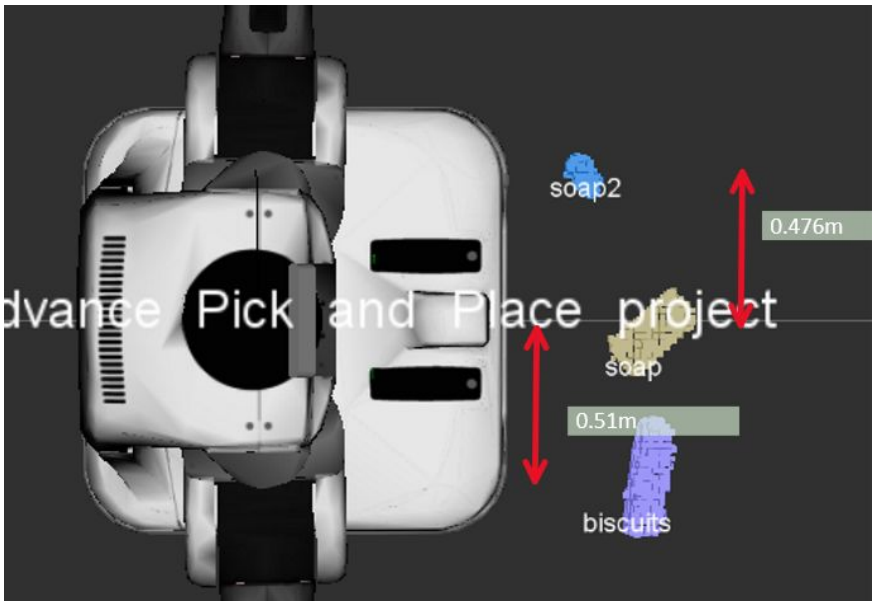
2. Voxel grid downsampling

A Voxeld grid downsampling filter helps to reduce the number of data points. This filter helps to reduce the computation load while still maintaining the features of the target object. The optimum value for LEAF_SIZE was determined through experimentation. The optimum value was LEAF_SIZE = 0.01

```
#-------------------------------------------------------------------------------------------
    # TODO: Voxel Grid Downsampling

    # Voxel Grid filter
    # Create a VoxelGrid filter object for our input point cloud
    vox = cloud.make_voxel_grid_filter()

    # Choose a voxel (also known as leaf) size
    # Note: this (1) is a poor choice of leaf size
    # Experiment and find the appropriate size!
    #After experimentation, 0.01 was determined to be the best value
    LEAF_SIZE = 0.01

    # Set the voxel (or leaf) size
    vox.set_leaf_size(LEAF_SIZE, LEAF_SIZE, LEAF_SIZE)

    # Call the filter function to obtain the resultant downsampled point cloud
    cloud_filtered = vox.filter()
#-------------------------------------------------------------------------------------------
```

3. Passthrough filter

- Two filters are required to remove the table and the boxes.
- The Z axis filter is required to remove the table while the y axis filter is required to remove the boxes on the side of the robot.
- The y axis value was determined using the measure feature in rviz



Code segment:

```
#----------------------------------------------------------------------------------------------
    # TODO: PassThrough Filter

    # PassThrough filter
    # Create a PassThrough filter object.
    passthrough_z = cloud_filtered.make_passthrough_filter()

    # Assign axis and range to the passthrough filter object.
    filter_axis = 'z'
    passthrough_z.set_filter_field_name (filter_axis)
    axis_min = 0.6
    axis_max = 1.1
    passthrough_z.set_filter_limits (axis_min, axis_max)

    # Finally use the filter function to obtain the resultant point cloud.
    cloud_filtered = passthrough_z.filter()

    #Adding another pass through filter to filter out the boxes on the y axis
    passthrough_y = cloud_filtered.make_passthrough_filter()

    # Assign axis and range to the passthrough filter object.
    filter_axis = 'y'
    passthrough_y.set_filter_field_name (filter_axis)
    axis_min = -0.5 # The distance between the camera center and the box edge is ~0.8 in rviz
    axis_max = 0.5

    passthrough_y.set_filter_limits (axis_min, axis_max)

    # Finally use the filter function to obtain the resultant point cloud.
    cloud_filtered = passthrough_y.filter()
#----------------------------------------------------------------------------------------------
```

4. RANSAC plane segmentation

```
#-------------------------------------------------------------------------------
    # TODO: RANSAC Plane Segmentation

    # Create the segmentation 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)

    # Max distance for a point to be considered fitting the model
    # Experiment with different values for max_distance
    # for segmenting the table
    max_distance = 0.01
    seg.set_distance_threshold(max_distance)

    # Call the segment function to obtain set of inlier indices and model coefficients
    inliers, coefficients = seg.segment()

#-------------------------------------------------------------------------------
    # TODO: Extract inliers and outliers

    cloud_table = cloud_filtered.extract(inliers, negative=False)   #extracted inliners
    cloud_objects = cloud_filtered.extract(inliers, negative=True)  #extracted outliers
```

Max_distance value was tweaked until only the cloud points from the objects were visible in the cloud_objects topic in rviz. The optimum value of max_distance was determined to be 0.01

2.2. Criteria 2.2

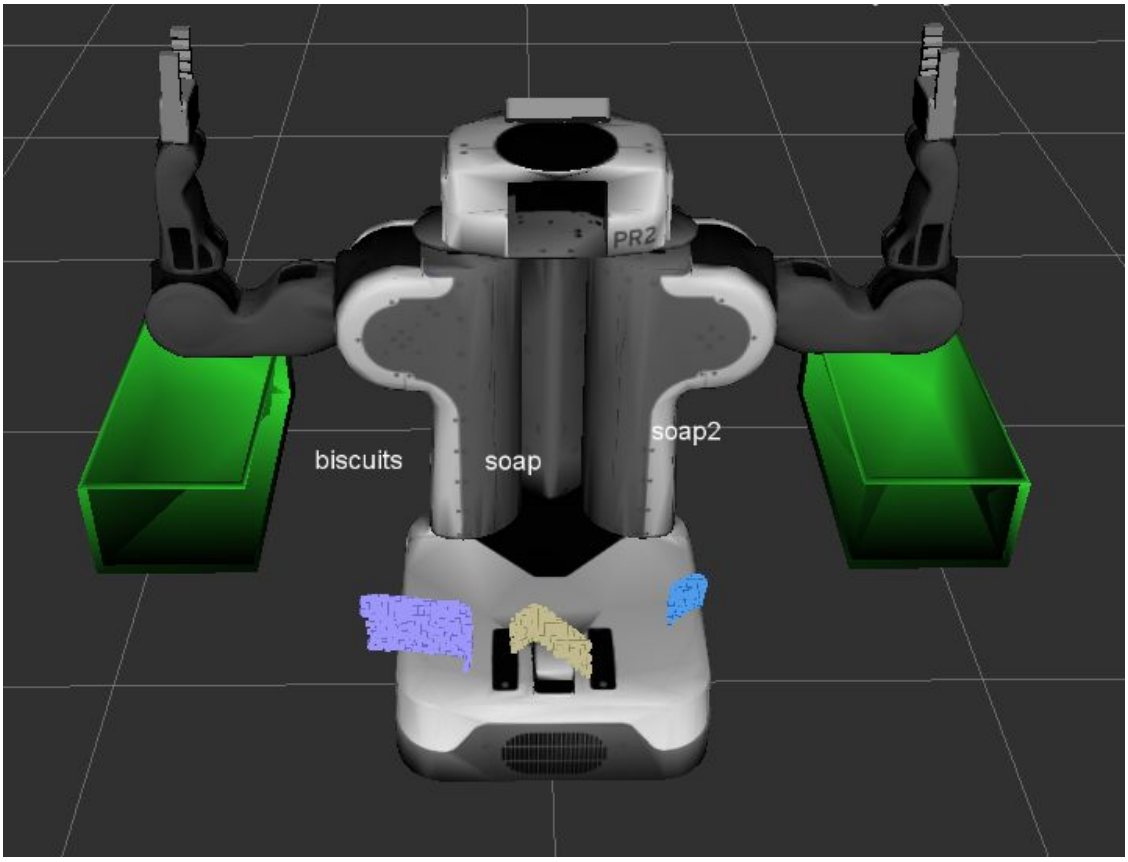
Criteria	Meets Specifications	Result
Complete Exercise 2 steps: Pipeline including clustering for segmentation implemented.	Steps for cluster segmentation have been added to the pcl_callback() function in the template Python script. Not required, but to help your reviewer consider adding screenshots of output at different steps in your writeup with brief explanations	pcl_callback() function implemented with euclidean cluster segmentation

Euclidean clustering

```
#-------------------------------------------------------------------------------
    # TODO: Euclidean Clustering
    white_cloud = XYZRGB_to_XYZ(cloud_objects) # Apply function to convert XYZRGB to XYZ
    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)
    # NOTE: Default values provided were  poor choices of clustering parameters
    # Your task is to experiment and find values that work for segmenting objects.
    ec.set_ClusterTolerance(0.05) # The values here provided the best perfromance
    ec.set_MinClusterSize(1)
    ec.set_MaxClusterSize(5000)
    # Search the k-d tree for clusters
    ec.set_SearchMethod(tree)
    # Extract indices for each of the discovered clusters
    cluster_indices = ec.Extract()
#-------------------------------------------------------------------------------
    # TODO: Create Cluster-Mask Point Cloud to visualize each cluster separately
    #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)
#-------------------------------------------------------------------------------
    # TODO: Convert PCL data to ROS messages
    ros_cloud_objects =  pcl_to_ros(cloud_objects)
    ros_cloud_table = pcl_to_ros(cloud_table)
    ros_cluster_cloud = pcl_to_ros(cluster_cloud)
#-------------------------------------------------------------------------------
    # TODO: Publish ROS messages
    pcl_objects_pub.publish(ros_cloud_objects)
    pcl_table_pub.publish(ros_cloud_table)
    pcl_cluster_cloud_pub.publish(ros_cluster_cloud) #check: commented temp
#-------------------------------------------------------------------------------
```

The best values for the tuning parameters are provided below. The values were varied until there were unique colors getting assigned for each object.
cluster tolerance = 0.05
Minimum cluster size = 1
Maximum cluster size = 5000

Output of cluster_cloud topic



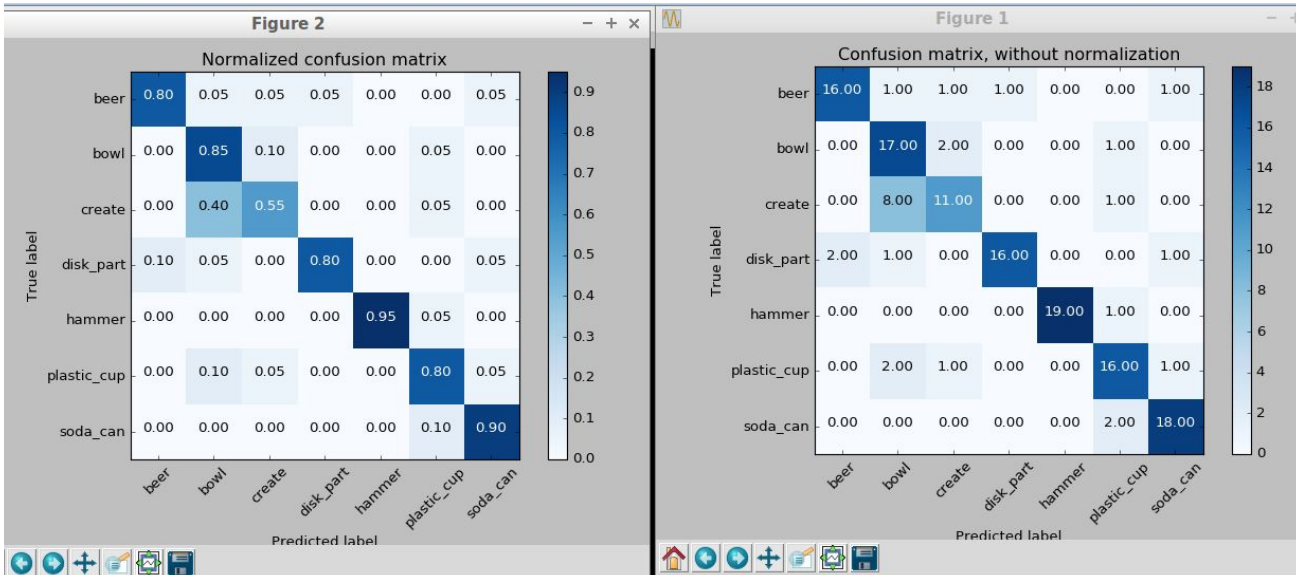
2.3. Criteria 2.3

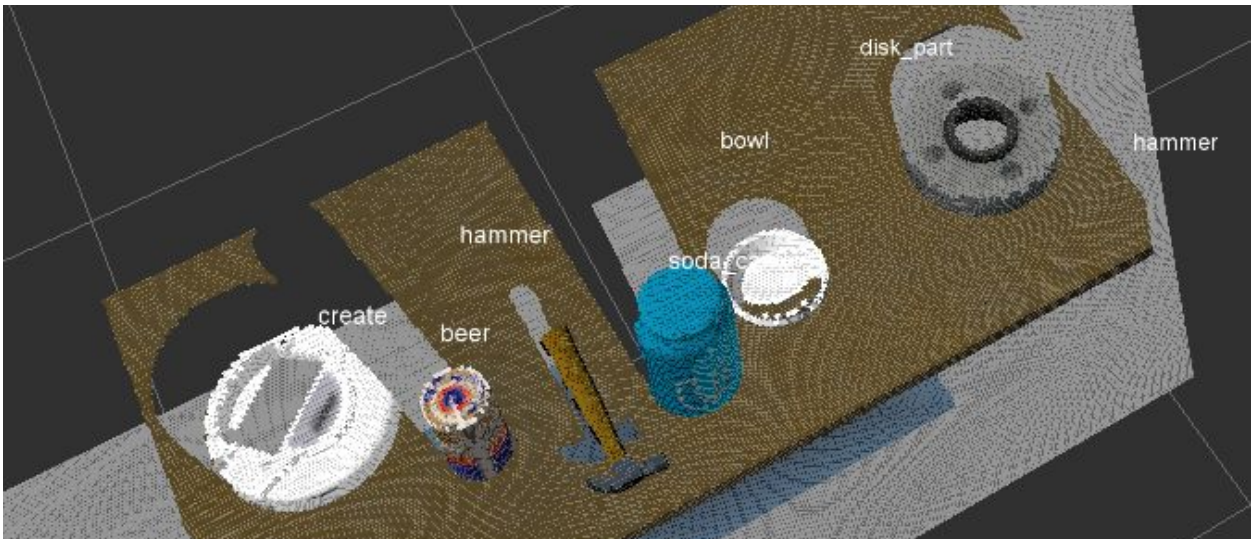
Criteria	Meets Specifications	Result
Complete Exercise 3 Steps. Features extracted and SVM trained. Object recognition implemented.	compute_color_histograms() and compute_normal_histograms() functions have been filled out and SVM has been trained using train_svm.py. Please provide a snapshot of your normalized confusion matrix (output from train_svm.py in your writeup / README. Object recognition steps have been implemented in the pcl_callback() function within template Python script. Not required, but to help your reviewer consider adding screenshots of output at different steps in your writeup with brief explanations.	Compute_color_histograms and compute_normal_histograms calculation completed and the samples were obtained using capture_features.py and training.launch. SVM was trained and the normalized confusion matrix is provided below. Object recognition steps implementation in pcl_callback() completed

Code segment:

```
    # Classify the clusters! (loop through each detected cluster one at a time)
    detected_objects_labels = []
    detected_objects = []

    for index, pts_list in enumerate(cluster_indices):
#---------------------------------------------------------------------------------------------
        # Grab the points for the cluster
        #pcl_cluster = cluster_cloud.extract(pts_list) #check: changing cloud_out to cluster_cloud
        pcl_cluster = cloud_objects.extract(pts_list)
#---------------------------------------------------------------------------------------------
        # Compute the associated feature vector

        # Convert the cluster from pcl to ROS using helper function
        ros_cluster = pcl_to_ros(pcl_cluster)

        # Extract histogram features
        chists = compute_color_histograms(ros_cluster, using_hsv=True) #check: Improve performance by using_hsv = true
        normals = get_normals(ros_cluster)
        nhists = compute_normal_histograms(normals)
        feature = np.concatenate((chists, nhists))
#---------------------------------------------------------------------------------------------
        # Make the prediction
        # Make the prediction, retrieve the label for the result
        # and add it to detected_objects_labels list
        #check: Doubt: how does th code cycle through the color and normal histogorams?

        prediction = clf.predict(scaler.transform(feature.reshape(1,-1)))
        label = encoder.inverse_transform(prediction)[0]
        detected_objects_labels.append(label)
#---------------------------------------------------------------------------------------------
        # Publish a label into RViz

        label_pos = list(white_cloud[pts_list[0]])
        label_pos[2] += .4
        object_markers_pub.publish(make_label(label,label_pos,index))
#---------------------------------------------------------------------------------------------
        # Add th+ee detected object to the list of detected objects.
        do = DetectedObject()
        do.label = label
        do.cloud = ros_cluster
        detected_objects.append(do)

        print"Detected objects <start>:","detected_objects_len:",len(detected_objects)
        for x in range(len(detected_objects)):
            print detected_objects[x].label
```

Normalized confusion matrix





Steps taken to tune the SVM:

The parameters used to fine tune the SVM were: Nbins, Kernel, color, Range

Nbins : ~catkin_ws/src/sensor_stick/src/sensor_stick/features.py

```
39      # Compute the histogram of the HSV channels separately
40      nbins=32
41      bins_range=(0, 256)
```

Kernel: ~catkin_ws/src/sensor_stick/scripts/train_svm.py

```
64  # Create classifier
65  clf = svm.SVC(kernel='linear')
66  #clf = svm.SVC(kernel='rbf')
```

Color: ~catkin_ws/src/sensor_stick/scripts/capture_features.py
Call chists function by setting the using_hsv value to true to use the hsv color space instead of RGB

```
74              # Extract histogram features
75              chists = compute_color_histograms(sample_cloud, using_hsv=True)
76              normals = get_normals(sample_cloud)
```

Range: ~catkin_ws/src/sensor_stick/scripts/capture_features.py

```
59          for i in range(50): #check: Improve from 5 to 10 attempts
60              # make five attempts to get a valid a point cloud then give up
61              sample_was_good = False
62              try_count = 0
63              while not sample_was_good and try_count < 5:
64                  sample_cloud = capture_sample()
65                  sample_cloud_arr = ros_to_pcl(sample_cloud).to_array()
```

Working combination : Nbins = 32, Kernel = linear, Color = hsv, Range = 15

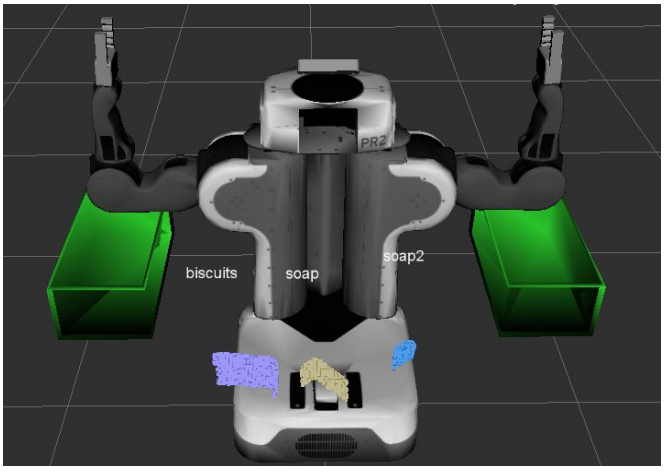
- Nbins experimented with 32 and 28. After reading performance reported by fellow students on slack, the value was retained as 32.
- Kernel - Experimented with RBF kernel. Tried tuning the SVM by altering the C and gamma values. After checking results achieved by fellow students, I kept the kernel as linear with default values.
- Range - Experimented with value of 5,10 and 15. During initial test runs, I changed both Nbin value and Range together. In later runs, I fixed Nbins as 32 and experimented only with Range. For test3_world, the range was increased to 50 to get an optimum confusion matrix.

3. Pick and Place Setup

Criteria	**Meets Specifications**	**Result**
For all three tabletop setups (test*.world), perform object recognition, then read in respective pick list (pick_list_*.yaml). Next construct the messages that would comprise a valid PickPlace request output them to .yaml format.	You can add this functionality to your already existing ros node or create a new node that communicates with your perception pipeline to perform sequential object recognition. Save your PickPlace requests into output_1.yaml, output_2.yaml, and output_3.yaml for each scene respectively. Add screenshots in your writeup of output showing label markers in RViz to demonstrate your object recognition success rate in each of the three scenarios. **Note: for a passing submission, your pipeline must correctly identify 100% of objects in test1.world, 80% (4/5) in test2.world and 75% (6/8) in test3.world.**	**Object identification:** Test1.world: 100% of objects identified correctly Test2.world: 80% of objects identified 5/5 times. Book identified as soap because it was not part of the training set Test3.world: 100% of objects identified 8/8 times **Output_n.yaml:** Yaml files attached to zip folder

3.1. test1.world

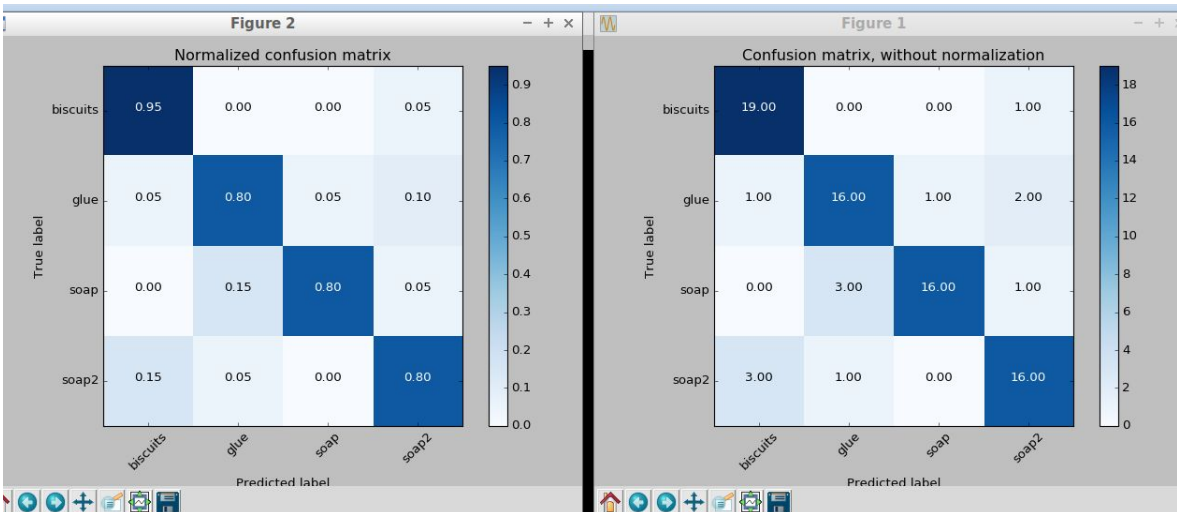
Requirement: 100% recognition
Result: 100% objects were detected.

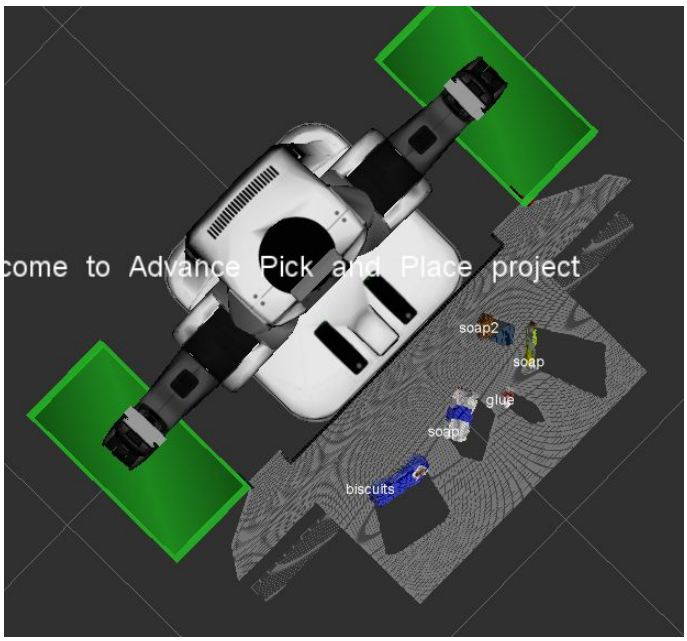


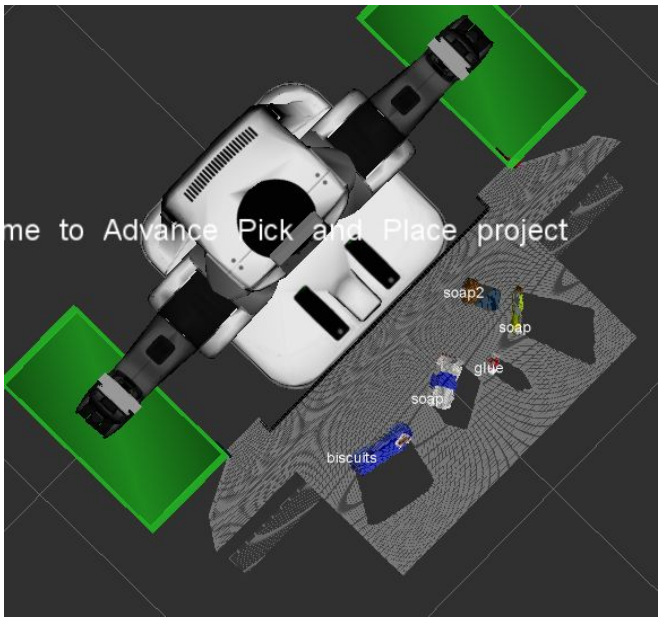
3.2. test2.world

Requirement: 80% recognition (4/5 times)
Result: 80% (5/5 times)

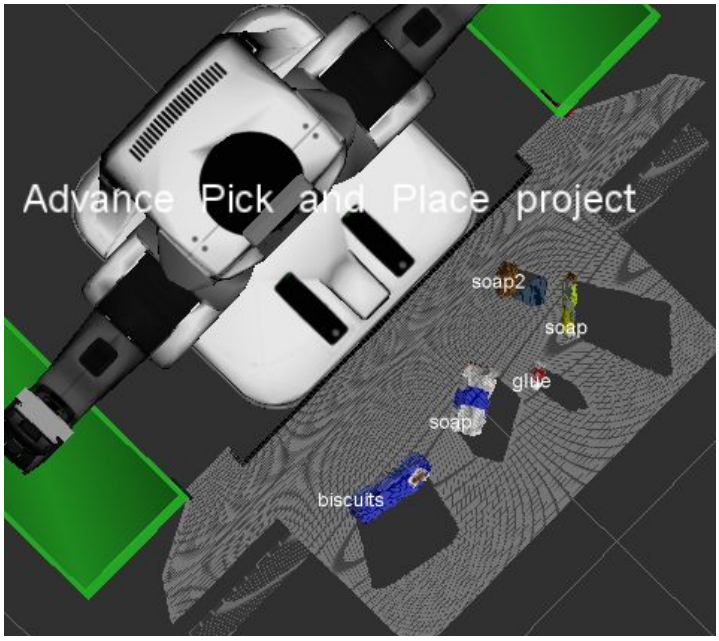
SVM training result:






me to Advance Pick and Place project
soap2
soap
glue
soap
biscuits

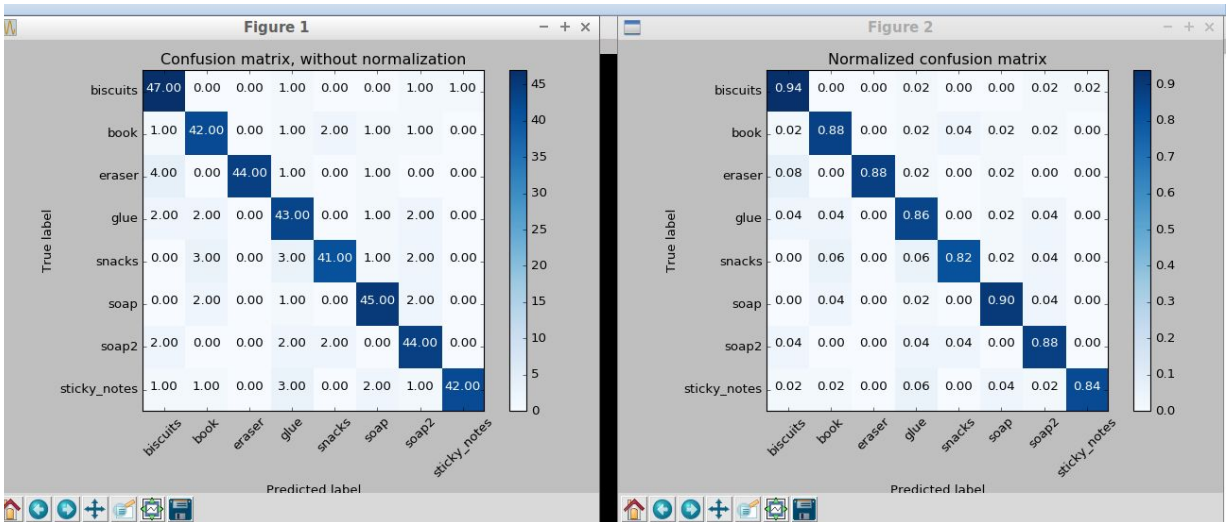

e to Advance Pick and Place project
soap2
soap
glue
soap
biscuits

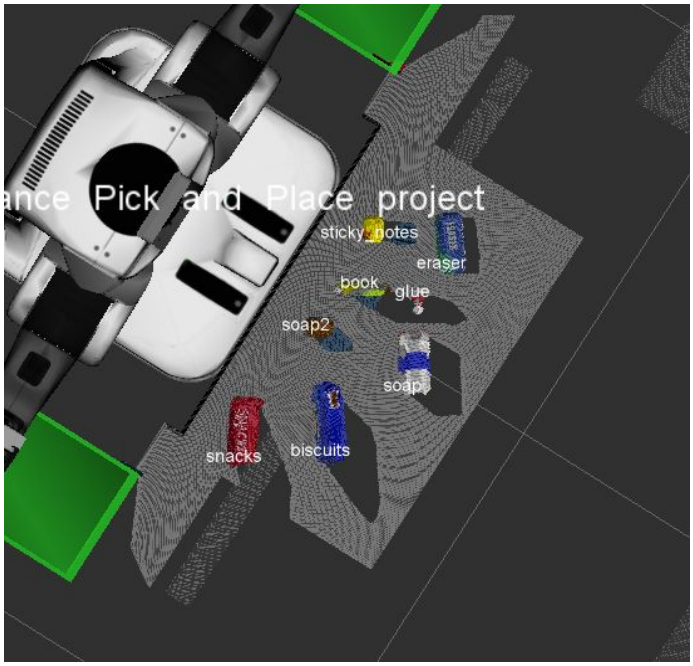

Advance Pick and Place project
soap2
soap
glue
soap
biscuits

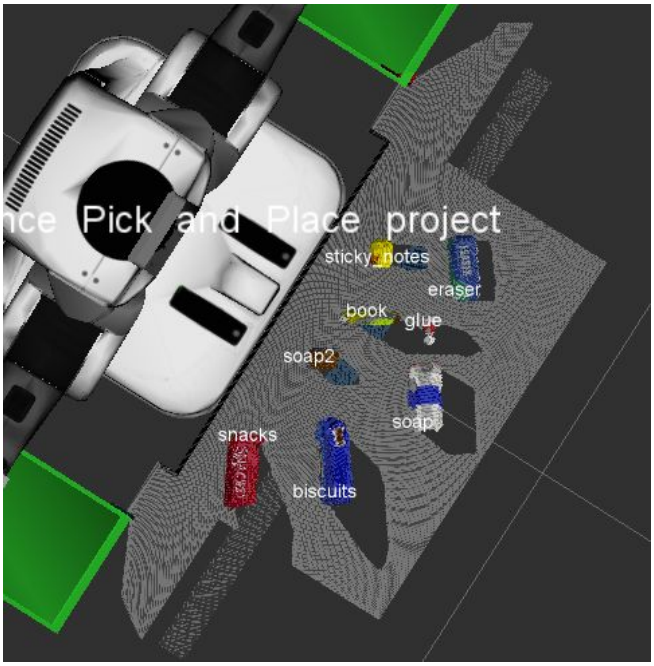
3.3. test3.world

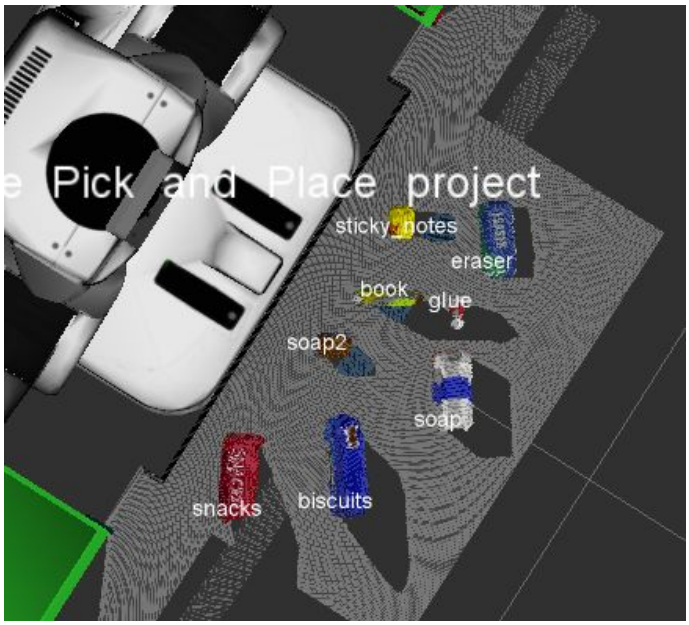
Requirement: 75% recognition (6/8 times)
Result: 100% 8/8 times

SVM training result:

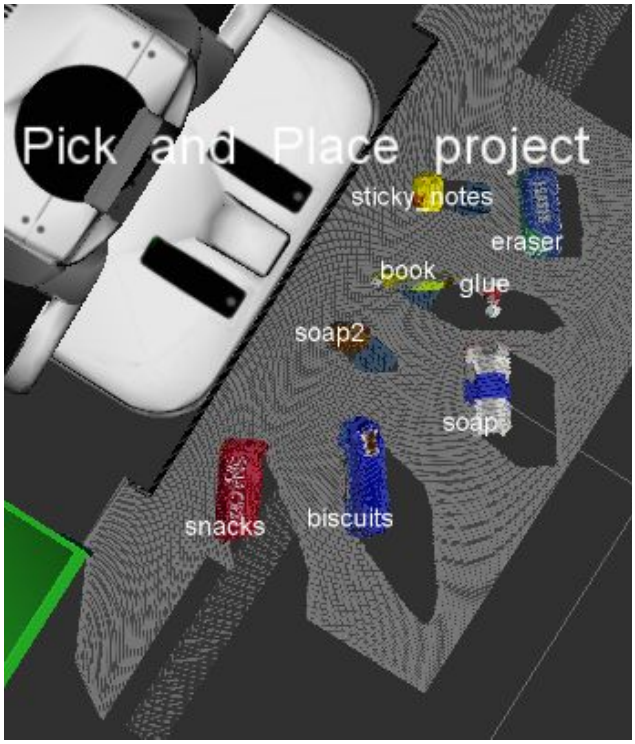


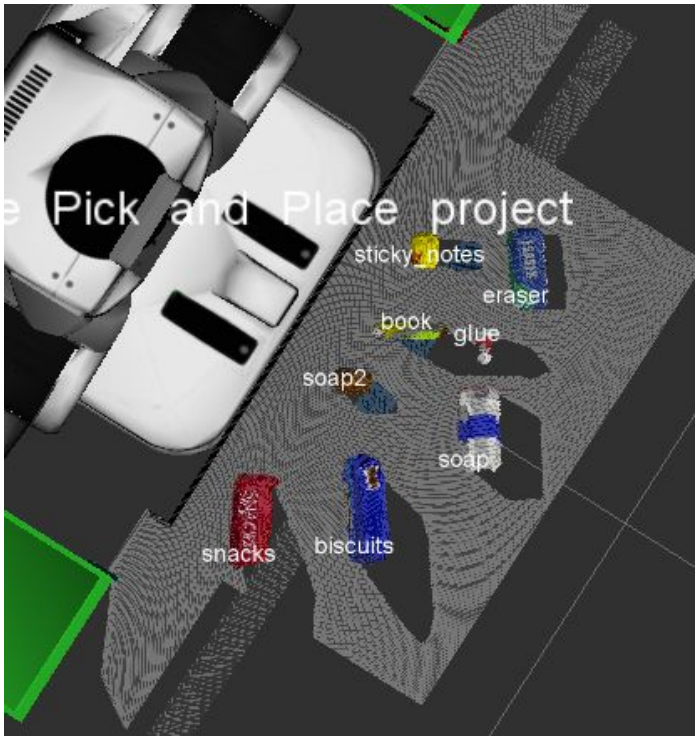


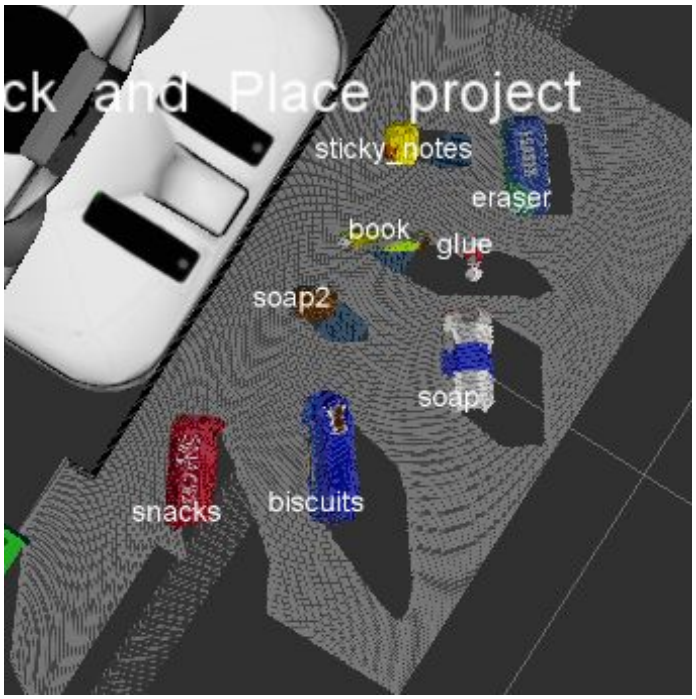


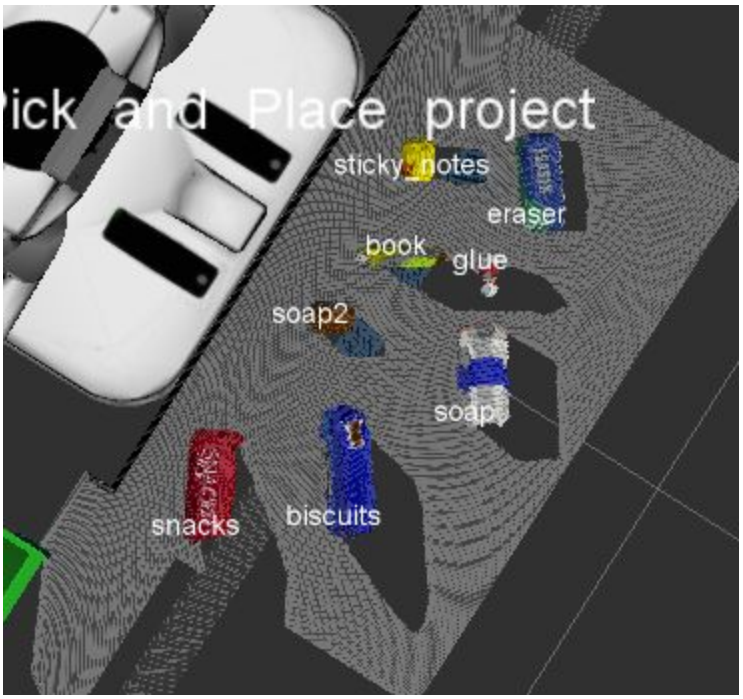

Pick and Place project
sticky_notes
eraser
book
glue
soap2
soap
snacks
biscuits

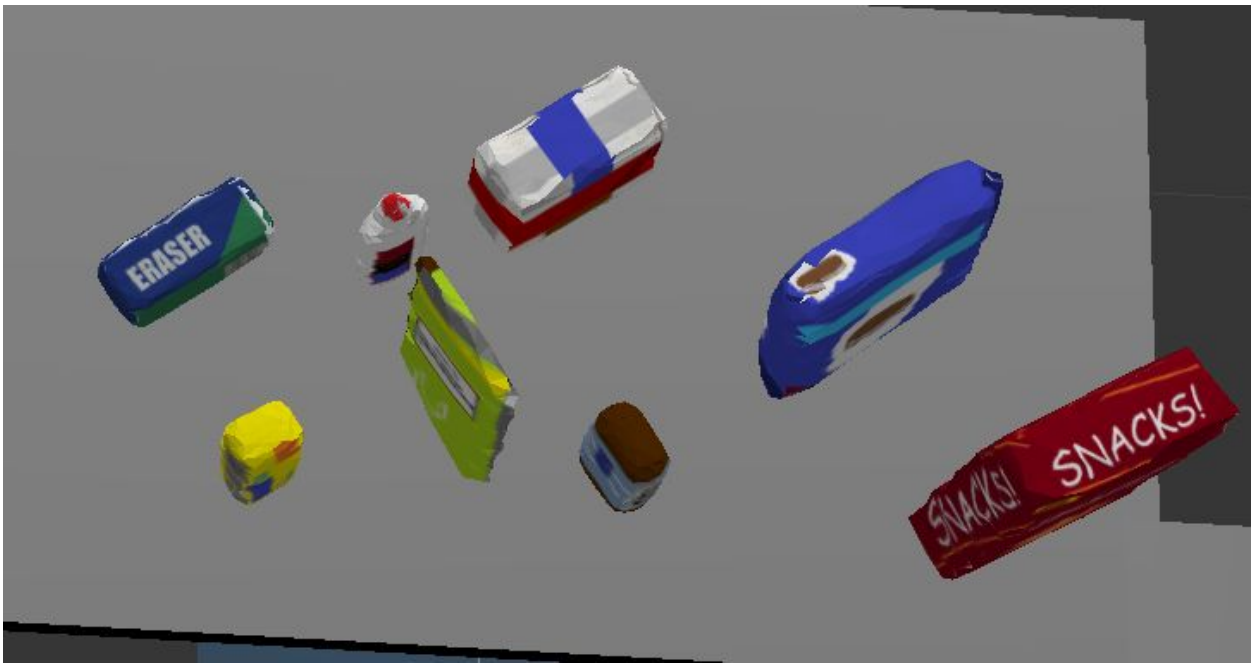

ick and Place project
sticky_notes
eraser
book
biscuits
soap2
soap
snacks
biscuits


Pick and Place project
sticky_notes
eraser
book
glue
soap2
soap
snacks
biscuits


Pick and Place project
sticky_notes
eraser
book
glue
soap2
soap
snacks
biscuits


ck and Place project
sticky_notes
eraser
book
glue
soap2
soap
snacks
biscuits


ick and Place project
sticky_notes
eraser
book
glue
soap2
soap
snacks
biscuits


ERASER
SNACKS!
SNACKS!