Perception Project

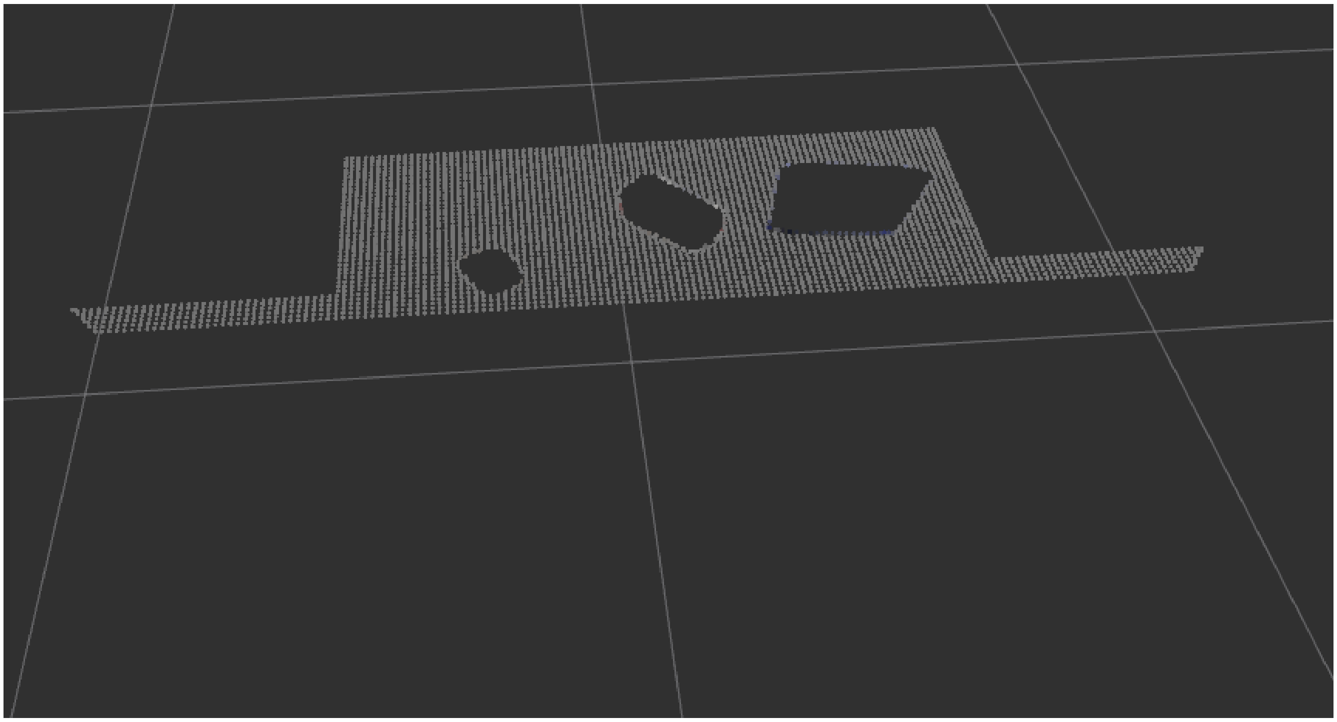
Rubric Points

1. Complete Exercise 1 steps. Pipeline for filtering and RANSAC plane fitting implemented.

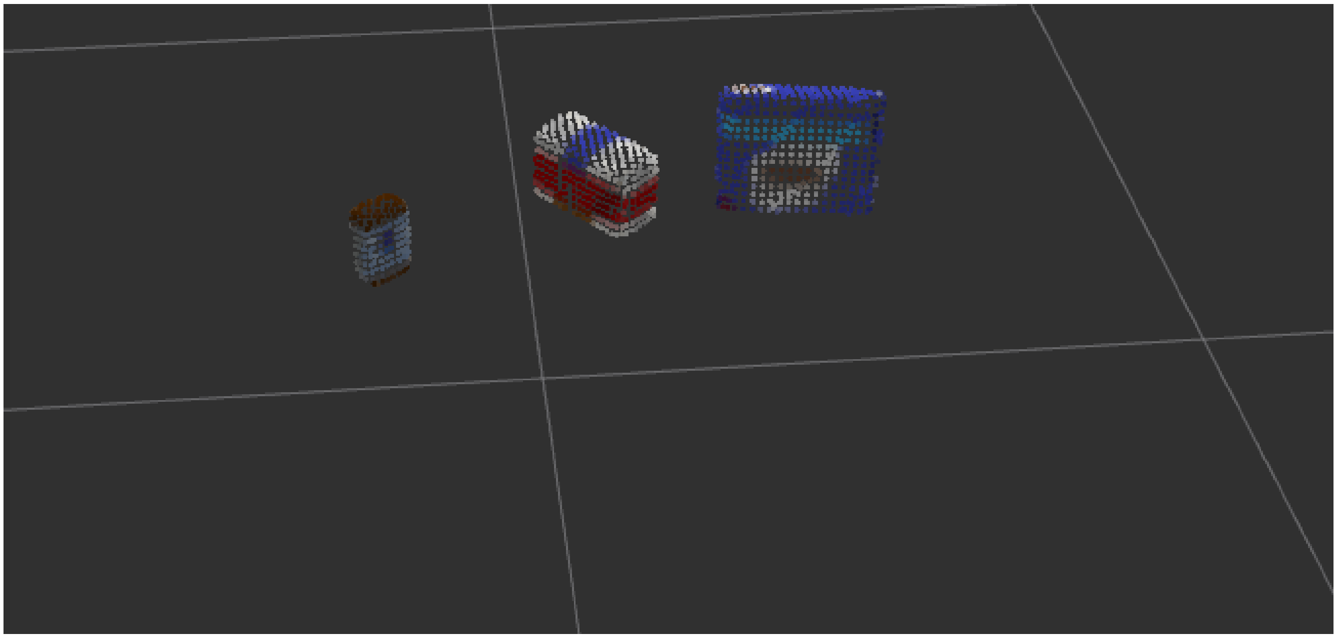
```
# Callback function for your Point Cloud Subscriber
def pcl_callback(pcl_msg):

# Exercise-2 TODOs:

    # TODO: Convert ROS msg to PCL data
    pcl_cloud=ros_to_pcl(pcl_msg)

    # TODO: Statistical Outlier Filtering

    stat=pcl_cloud.make_statistical_outlier_filter()
    stat.set_mean_k(10)
    stat.set_std_dev_mul_thresh(0.1)
    stat_cloud=stat.filter()

    # TODO: Voxel Grid Downsampling
    vox = stat_cloud.make_voxel_grid_filter()
    leafSize = 0.01
    vox.set_leaf_size(leafSize,leafSize,leafSize)
    cloud_vox = vox.filter()

    # TODO: PassThrough Filter
    passthrough = cloud_vox.make_passthrough_filter()
    filter_axis = 'z'
    passthrough.set_filter_field_name(filter_axis)
    axis_min = 0.2
    axis_max = 10
    passthrough.set_filter_limits(axis_min,axis_max)
    temp_passthrough=passthrough.filter()
    passthrough2 = temp_passthrough.make_passthrough_filter()
    filter_axis = 'x'
    passthrough2.set_filter_field_name(filter_axis)
    axis_min = 0.4
    axis_max = 0.8
    passthrough2.set_filter_limits(axis_min,axis_max)
    cloud_passthrough = passthrough2.filter()
    #cloud_passthrough = passthrough.filter()

    # TODO: RANSAC Plane Segmentation
    segmented = cloud_passthrough.make_segmenter()
    segmented.set_model_type(pcl.SACMODEL_PLANE)
    segmented.set_method_type(pcl.SAC_RANSAC)
    threshold = 0.01
    segmented.set_distance_threshold(threshold)
    inliers, coefficients = segmented.segment()

    # TODO: Extract inliers and outliers
    cloud_table = cloud_passthrough.extract(inliers,negative = False)
    cloud_objects = cloud_passthrough.extract(inliers,negative = True)
```

I implemented a statistical outlier filter to remove the noise. After tuning, values of mean 10 and standard deviation of 0.1 seemed to work well. This was followed by voxel downsampling with grids of 0.1 along all axes. Further, I implemented a passthrough filter along two axes. One along x to separate out the table, and one along z to get rid of the shadows on the ground. Finally, RANSAC plane filtering was applied to separate out the table, to give a table cloud and an objects cloud.

TABLE:



OBJECTS:

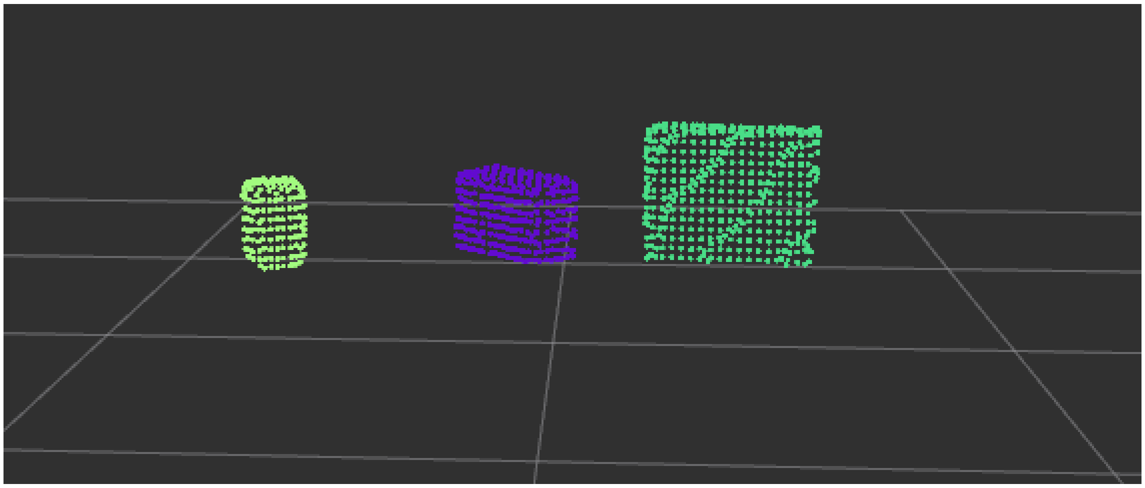


2. Complete Exercise 2 steps: Pipeline including clustering for segmentation implemented.

```
# TODO: Extract inliers and outliers
cloud_table = cloud_passthrough.extract(inliers,negative = False)
cloud_objects = cloud_passthrough.extract(inliers,negative = True)

# TODO: Euclidean Clustering
white_cloud = XYZRGB_to_XYZ(cloud_objects)
tree = white_cloud.make_kdtree()

# TODO: Create Cluster-Mask Point Cloud to visualize each cluster separately
ec = white_cloud.make_EuclideanClusterExtraction()
ec.set_ClusterTolerance(0.05)
ec.set_MinClusterSize(50)
ec.set_MaxClusterSize(500)

ec.set_SearchMethod(tree)

cluster_indices = ec.Extract()

# Create colored cluster masks
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_table = pcl_to_ros(cloud_table)
ros_cloud_objects = pcl_to_ros(cloud_objects)
ros_cluster_cloud = pcl_to_ros(cluster_cloud)
```

Clustering was implemented for the objects cloud by first converting them to white cloud, and then selecting nearby points based on Euclidean distance, with a tolerance of 0.05, minimum size of 50 and maximum size of 500.

CLUSTER:



3. Complete Exercise 3 Steps. Features extracted and SVM trained. Object recognition implemented.

```python
# Exercise-3 TODOs:

    # 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 = cloud_objects.extract(pts_list)
        ros_cluster = pcl_to_ros(pcl_cluster)

        # 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))

        # Make the prediction

        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 the detected object to the list of detected objects.

        do = DetectedObject()
        do.label = label
        do.cloud = ros_cluster
        detected_objects.append(do)

    # Publish the list of detected objects
    rospy.loginfo('Detected {} objects: {}'.format(len(detected_objects_labels), detected_objects_labels))
    detected_objects_pub.publish(detected_objects)
```

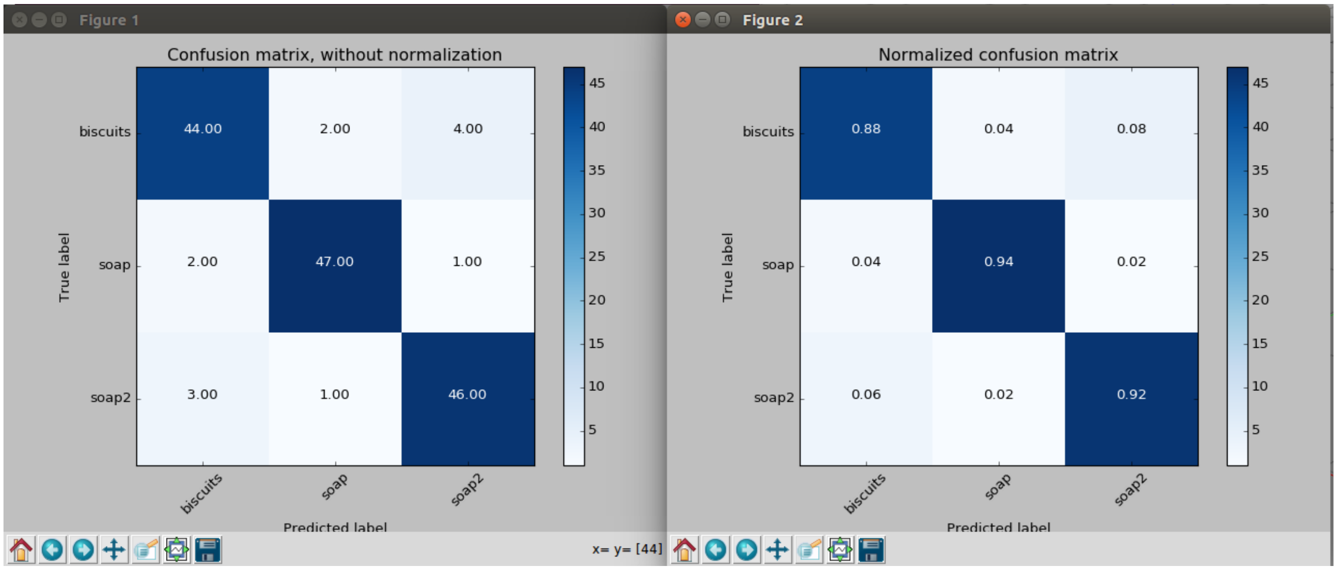
Here is a snippet of both the capture features code.

capture_features.py

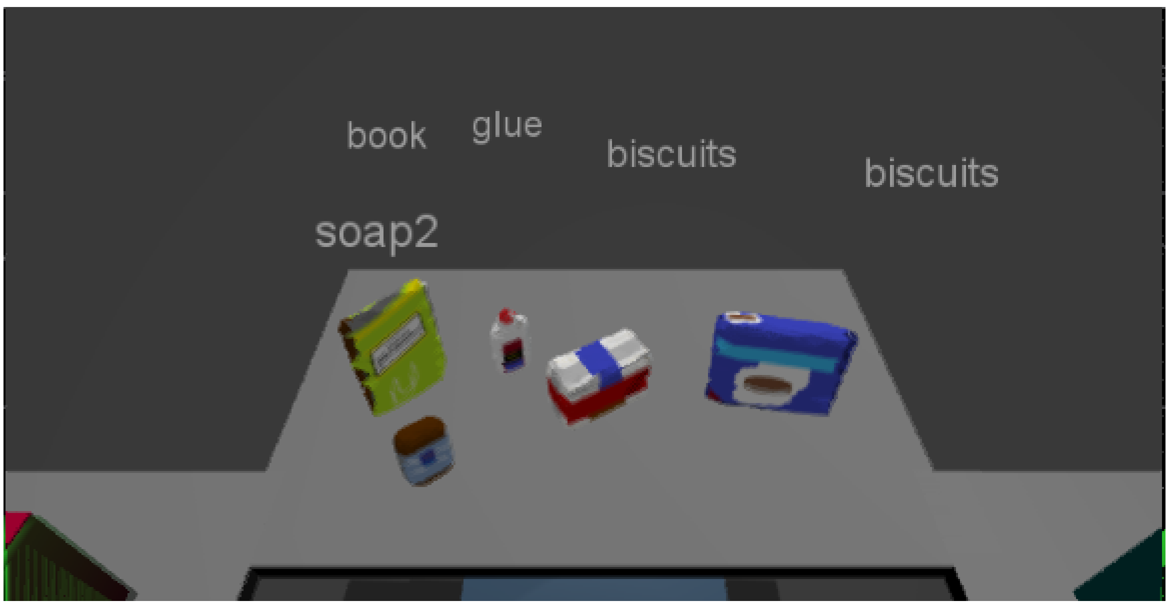
```
        'glue']
'''
#'''
# Models for test3 world of Perception Project
models = [\
    'sticky_notes',
    'book',
    'snacks',
    'biscuits',
    'eraser',
    'soap2',
    'soap',
    'glue']
#'''


# Disable gravity and delete the ground plane
initial_setup()
labeled_features = []

for model_name in models:
    spawn_model(model_name)

    for i in range(50):
        # make five attempts to get a valid a point cloud then give up
        sample_was_good = False
        try_count = 0
        while not sample_was_good and try_count < 5:
            sample_cloud = capture_sample()
            sample_cloud_arr = ros_to_pcl(sample_cloud).to_array()

            # Check for invalid clouds.
            if sample_cloud_arr.shape[0] == 0:
                print('Invalid cloud detected')
                try_count += 1
            else:
                sample_was_good = True

        # Extract histogram features
        chists = compute_color_histograms(sample_cloud, using_hsv=True)
        normals = get_normals(sample_cloud)
        nhists = compute_normal_histograms(normals)
        feature = np.concatenate((chists, nhists))
        labeled_features.append([feature, model_name])

    delete_model()
```

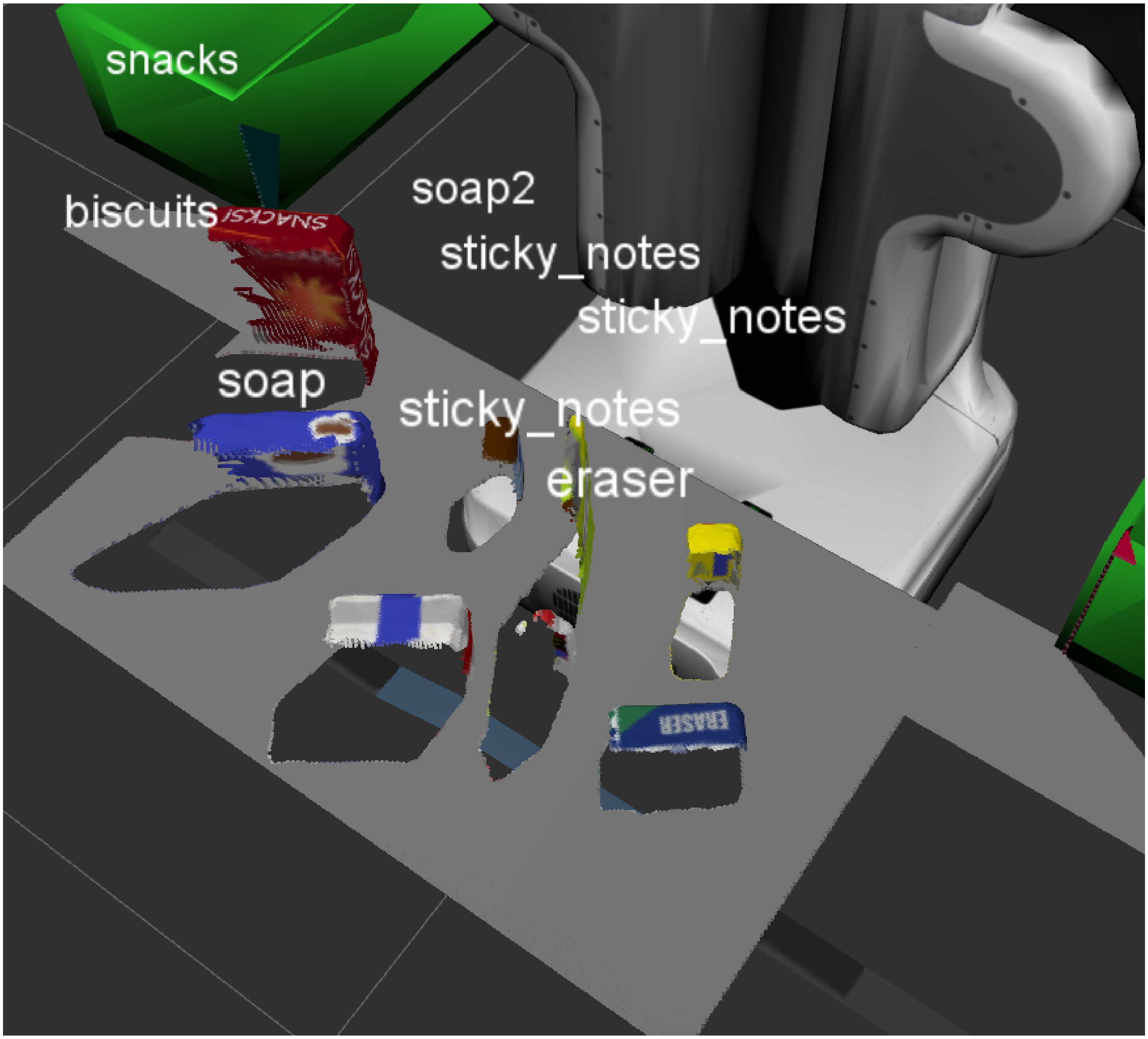
The SVM was trained with a linear option.

Results of train_svm.py on test3.world



4. 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.





There are some times when all objects seem to be recognized correctly in the last world, but the screenshot I have attached shows the worst case. In the worst case, it is only able to detect 6 out of the 8 objects.

Conclusion

The object recognition can be improved further. Currently I get 100% recognition some times for the last world, but a lot of times it recognizes only 6 objects. I tried tweaking the features by increasing the number of orientations captured, and also changing the number of bins in the histograms. While I wa able to improve the values in the confusion matrix after that, the results were still pretty similar.

Another problem I encounter is that the gripper does not lift the objects some times. As a result, the arm starts colliding with those objects in the subsequent pick and place operations. I plan on increasing the friction coefficient of the objects to see if the gripper can hold on to the object.