3D Perception Project

The goal is to correctly detect objects on a table and pick & place them by a pr2 robot.

1. Exercise 1-3

Statistical outlier filter

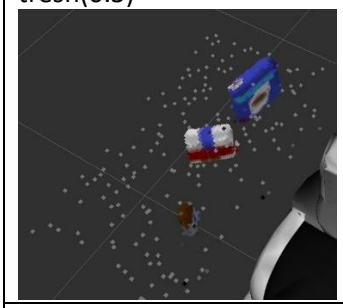
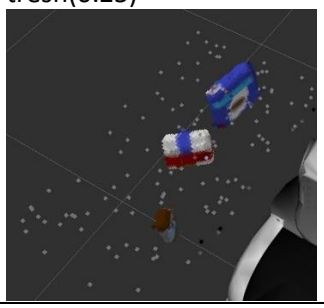
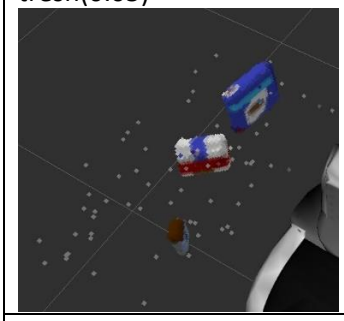
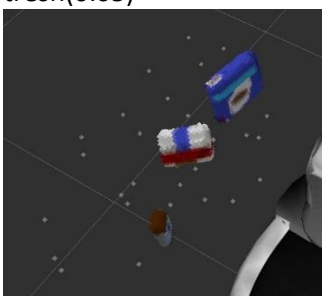
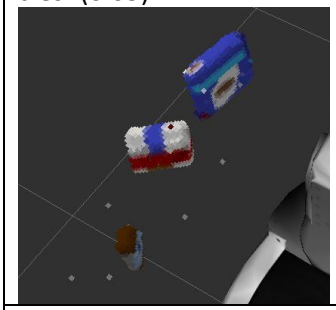
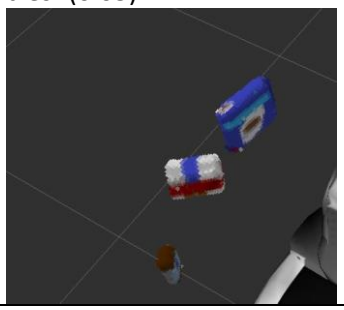
The ROS node pr2/world/points contains noisy point cloud. The noise can be filtered out by a statistical outlier filter.

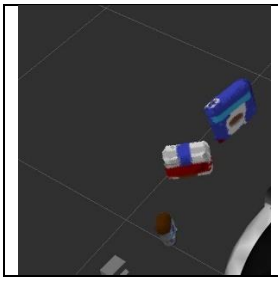
```
if __name__ == '__main__':

    # TODO: ROS node initialization
    # rospy.init_node('object_recognition', anonymous=True)
    rospy.init_node('feature_extractor', anonymous=True)

    # TODO: Create Subscribers
    pcl_sub = rospy.Subscriber("/pr2/world/points", pc2.PointCloud2, pcl_callback, queue_size=1)
```

```
# TODO: Statistical Outlier Filtering
outlierfilter = cloud.make_statistical_outlier_filter()
outlierfilter.set_mean_k(10)
outlierfilter.set_std_dev_mul_thresh(0.2)
outlierfilter_cloud = outlierfilter.filter()
```

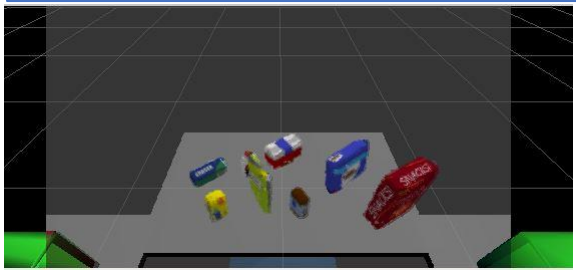
Set_mean_k(50) tresh(0.5) 	Set_mean_k(50) tresh(0.25) 
Set_mean_k(40) tresh(0.05) 	Set_mean_k(30) tresh(0.05) 
Set_mean_k(20) tresh(0.05) 	Set_mean_k(10) tresh(0.05) 
Set_mean_k(10) tresh(0.2)	



It showed that the filter setting of mean_k(10) and tresh(0.2) were filter out most of the noise.

Voxel / Passthrough

```
# TODO: Voxel Grid Downsampling
vox = outlierfilter_cloud.make_voxel_grid_filter()
LEAF_SIZE = 0.005
vox.set_leaf_size(LEAF_SIZE, LEAF_SIZE, LEAF_SIZE)
cloud_vox_filtered = vox.filter()
# TODO: PassThrough Filter
passthrough_z = cloud_vox_filtered.make_passthrough_filter()
passthrough_z.set_filter_field_name ('z')
axis_min = 0.6 # all under axis_min [m] is erased
axis_max = 1.2  # all over axis_max [m] is erased, ev. 1.3
passthrough_z.set_filter_limits (axis_min, axis_max)
cloud_filtered_passthrough = passthrough_z.filter()
passthrough_y = cloud_filtered_passthrough.make_passthrough_filter()
passthrough_y.set_filter_field_name ('y')
axis_min = -0.46 # all under axis_min [m] is erased
axis_max = 0.46  # all over axis_max [m] is erased
passthrough_y.set_filter_limits (axis_min, axis_max)
cloud_filtered = passthrough_y.filter()
```

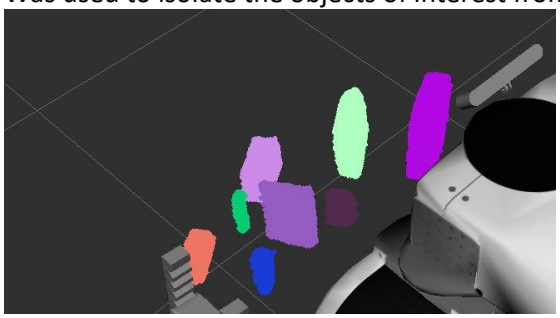


As passthrough also the y was considered to make sure the dropboxes were not considered as objects.

RANSAC plane fitting:

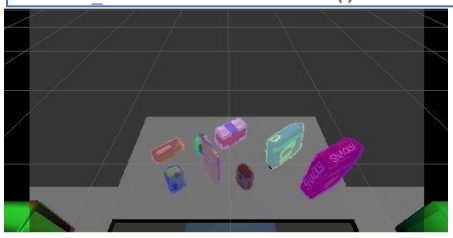
```
# TODO: RANSAC Plane Segmentation
seg = cloud_filtered.make_segmenter()
seg.set_model_type(pcl.SACMODEL_PLANE)
seg.set_method_type(pcl.SAC_RANSAC)
max_distance = 0.015 # [m] 0.01 max dist of point to be consid
seg.set_distance_threshold(max_distance)
inliers, coefficients = seg.segment()
# TODO: Extract inliers and outliers
# how close a point must be to the model in order to be consid
# Inliner
cloud_table = cloud_filtered.extract(inliers, negative=False)
# Outliner
cloud_objects = cloud_filtered.extract(inliers, negative=True)
```

Was used to isolate the objects of interest from the rest of the scene.



Euclidean clustering:

```
# Construct k-d tree (cloud with only spatial (raeumlich) information, colorless cloud)
white_cloud = XYZRGB_to_XYZ(cloud_objects)
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: These are poor choices of clustering parameters
# Your task is to experiment and find values that work for segmenting objects.
ec.set_ClusterTolerance(0.025) # [m]     0.05 to 0.006 is possible!
ec.set_MinClusterSize(30)   #20-50
ec.set_MaxClusterSize(2500) #2500-3000
# Search the k-d tree for clusters
ec.set_SearchMethod(tree)
# Extract indices for each of the discovered clusters
cluster_indices = ec.Extract()
```



To create separate clusters for each item.

Object recognition:

Next I first captured the features of the different object with the sensor_stick by just coping the models to the sensor_stick and capture them. I made 500 captures for each object. I used 32bins for the histogram calculations:

The color histogram function:

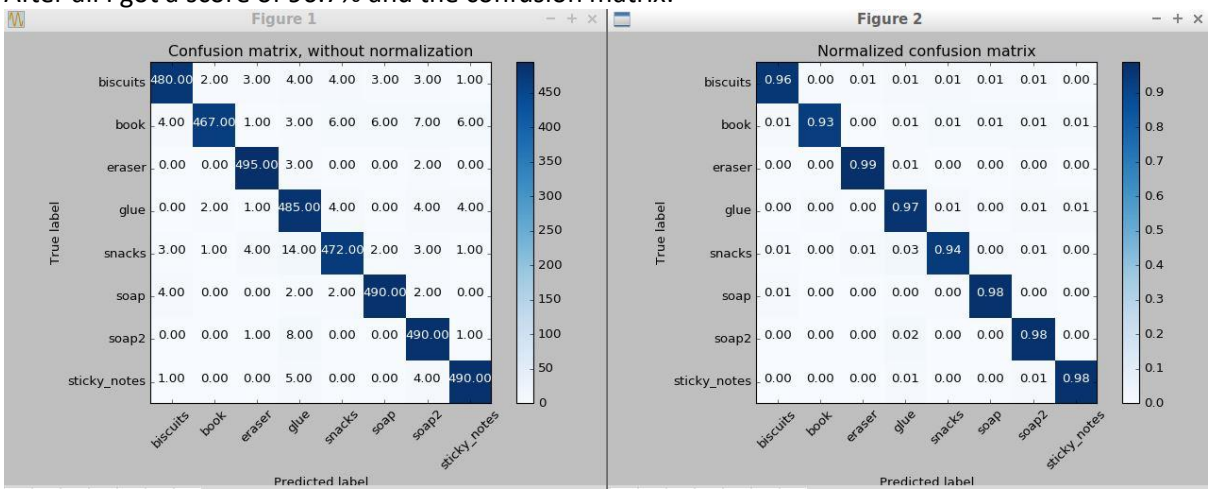
```
# Compute histograms
channel_1_hist = np.histogram(channel_1_vals, bins=32, range=(0,256))
channel_2_hist = np.histogram(channel_2_vals, bins=32, range=(0,256))
channel_3_hist = np.histogram(channel_3_vals, bins=32, range=(0,256))
```

The normal histogram function:

```
norm_x_hist = np.histogram(norm_x_vals, bins=32, range=(-1, 1))
norm_y_hist = np.histogram(norm_y_vals, bins=32, range=(-1, 1))
norm_z_hist = np.histogram(norm_z_vals, bins=32, range=(-1, 1))
```

Finally to train the model I used a LinearSVC:

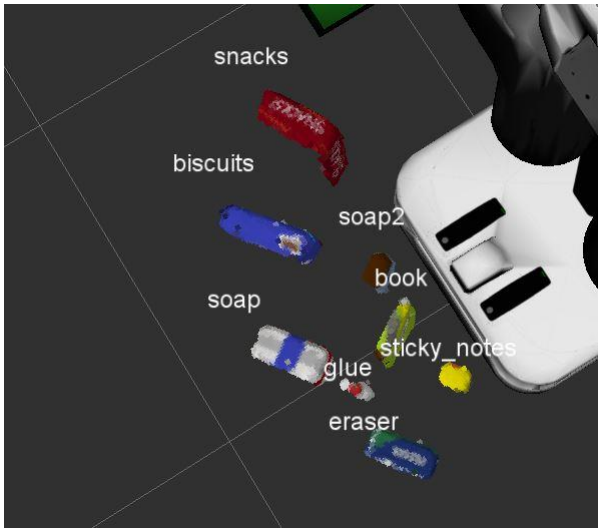
After all I got a score of 96.7% and the confusion matrix:



World 1: (3/3)

World 2: (5/5)

World 3: (8/8)



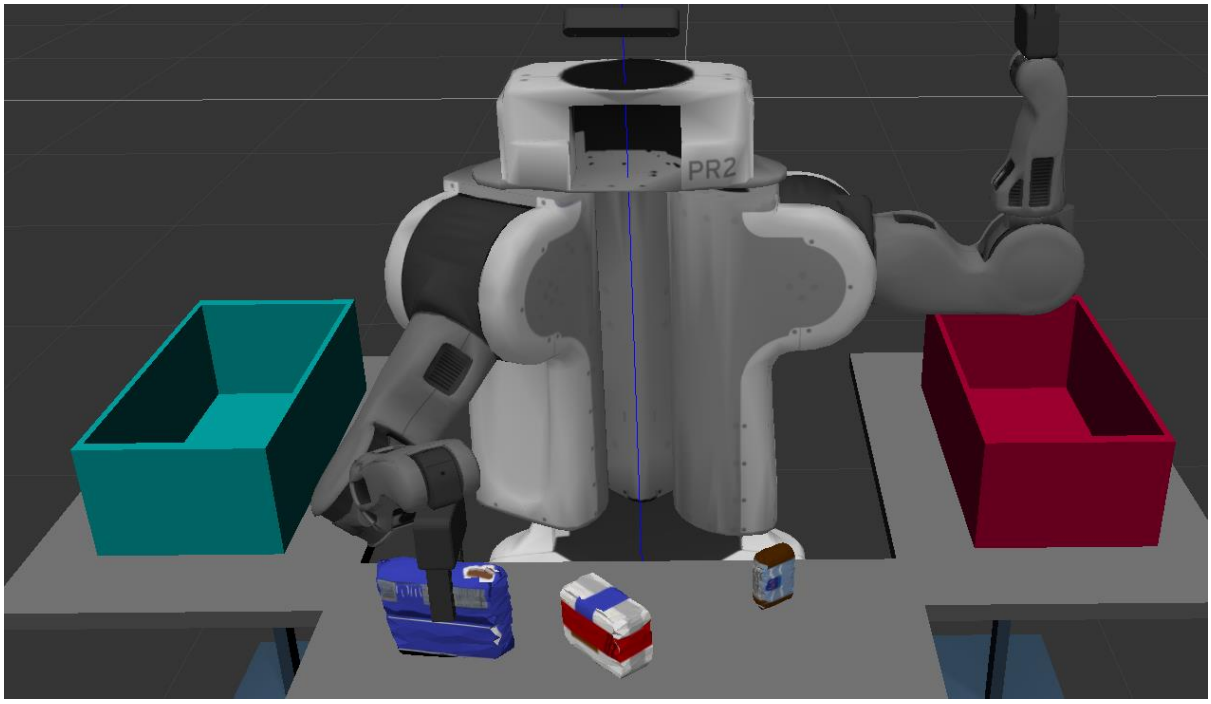
2. Pick and Place

Calculate the centroid

```
# Get the PointCloud for a given object and obtain it's centroid
points_arr = ros_to_pcl(found_object.cloud).to_array()
centroids = np.mean(points_arr, axis=0)[:3]
#print("centroids item 0: ",centroids.item(0))

pick_pose.position.x =  np.asscalar(centroids[0])
pick_pose.position.y =  np.asscalar(centroids[1])
pick_pose.position.z =  np.asscalar(centroids[2])
```

With the code above the centre of each object was calculated to be able to grab the item.
The rest of the code you can be found in the project_template.py



3. Improvement

The objects from the perception exercise could be used as well, I could do a world 4 with all objects. The robot could be improved how it crabs the stuff. The biscuit somehow had no gravity and a weird motion in the box after droping. This would need to be checked. Try out if other svm train kernels work as well and which one works best.