Project: 3D Perception - Thomas Hatting

Exercise 1, 2 and 3 pipeline implemented

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

The template has been filled out including voxel grid filtering, pass-through filtering and RANSAC plane segmentation. The original image is shown figure 1. The segmented table image is shown in figure 2 (extracted inliers), and furthermore, the segmented objects are shown in figure 3 (extracted outliers).

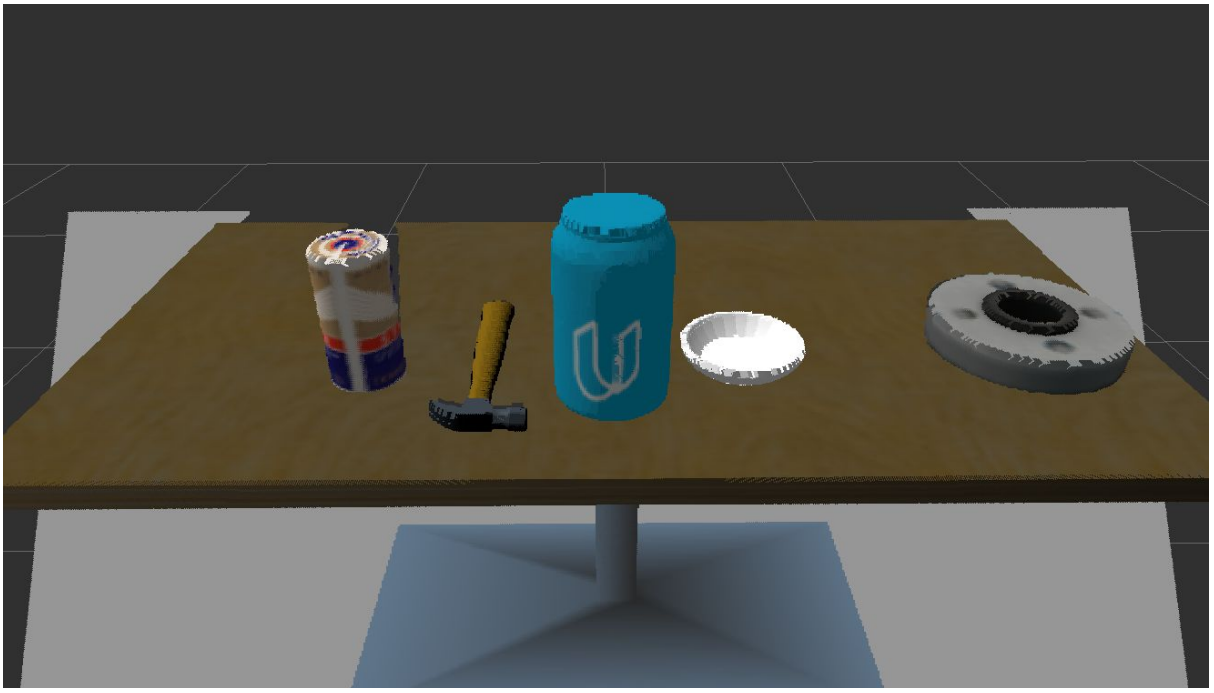

Figure 1: The original image

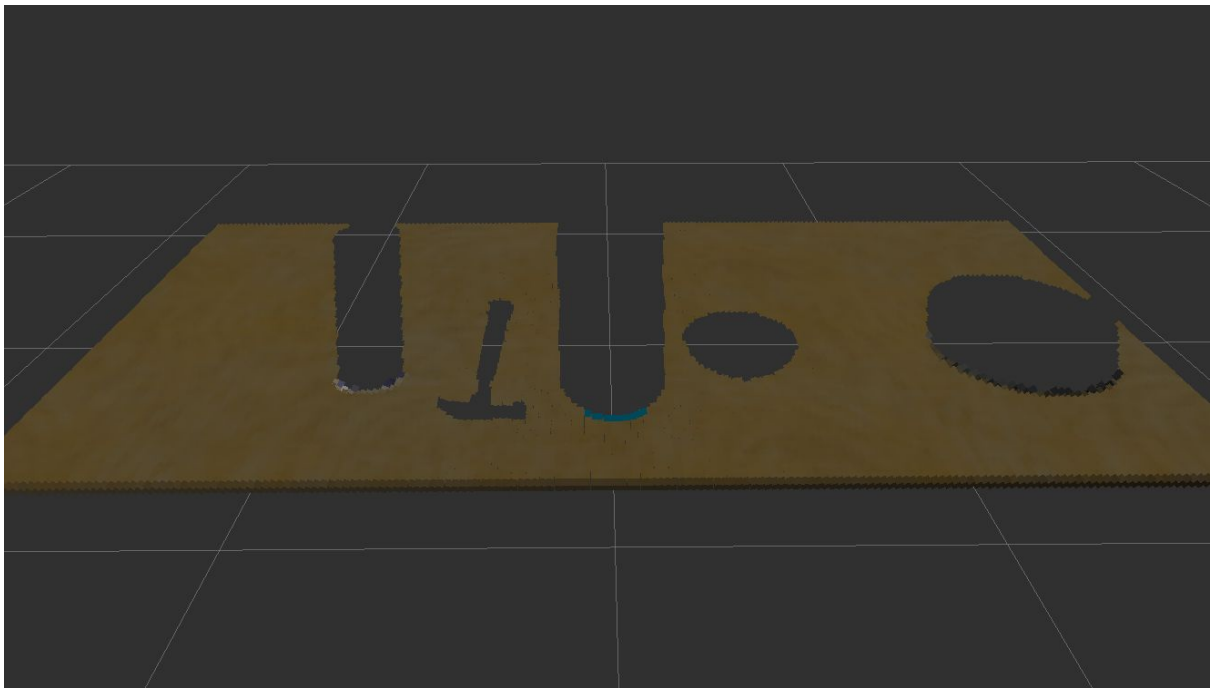

Figure 2: The segmented table after voxel grid downsampling, pass-through filtering and RANSAC plane segmentation (extracted inliers)

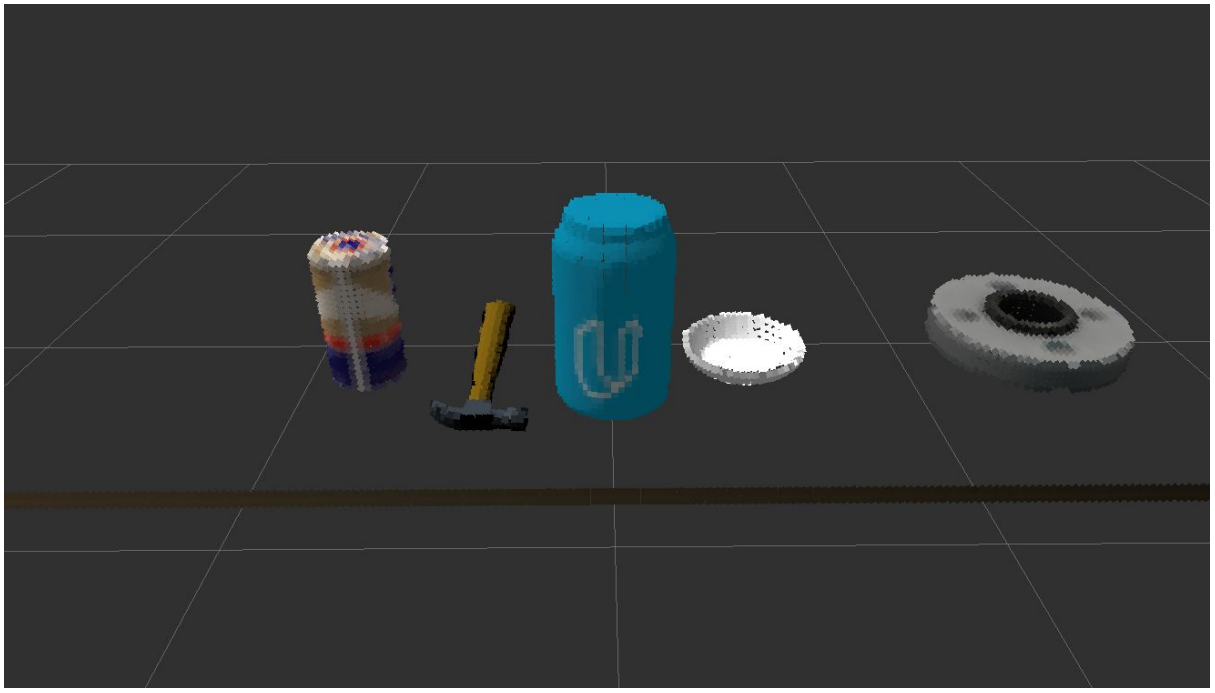

Figure 3: The objects after voxel grid downsampling, pass-through filtering and RANSAC plane segmentation (extracted outliers)

The Python code for Exercise 1 is shown in the appendix at the end of the report.

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

The steps for cluster segmentation have been added to the pcl_callback() function in the Python script. The segmentation method used is Euclidean Clustering (DBSCAN algorithm). The results are shown in figure 4. As can be seen, the objects are separated and colored individually (a ledge of the table can be seen in light green at the bottom of the image; this can be due to the fact that a bit of the table slipped through the filtering process in the earlier stages).

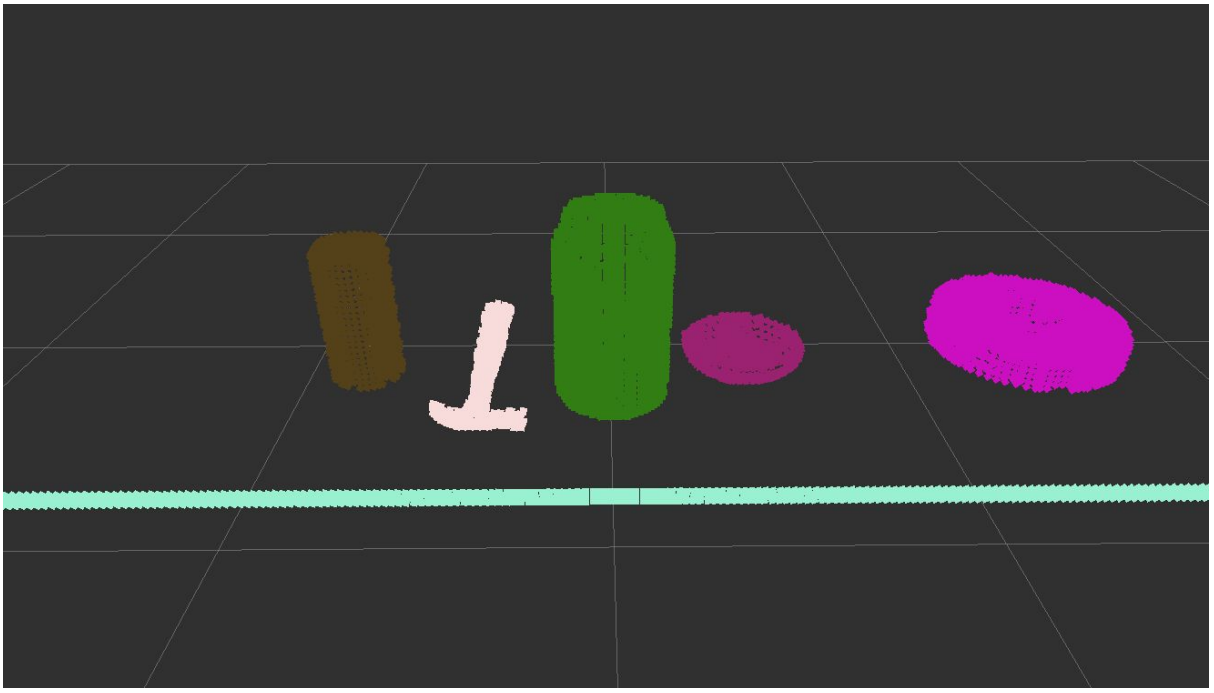

Figure 4: The objects after clustering segmentation (Euclidean Clustering)

The Python code for Exercise 2 is shown in the appendix at the end of the report.

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

The support vector machine (SVM) has been trained using scripts “capture_features.py” and “train_svm.py”. The accuracy of the resulting classifier has been improved by following two steps:

a) In the script “capture_features.py”, the for-loop *for i in range(5):* has been changed to *for i in range(10):* (ie.10 iterations instead of 5).

b) Furthermore, in script “capture_features.py”, the HSV color model has been used instead of the RGB model by setting parameter “using_hsv” to True.

```
chists = compute_color_histograms(ros_cluster, using_hsv=True)
```

In figure 5, the resulting normalized confusion matrix is shown. It shows that the SVM classifier is fairly accurate for most objects (accuracy is shown by the numbers in the diagonal).

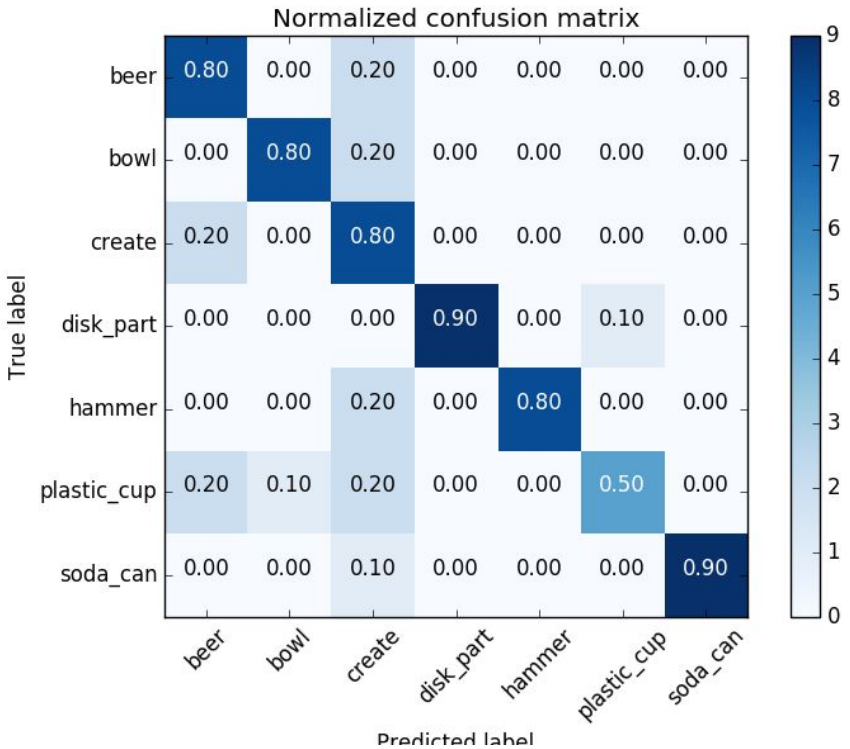

Figure 5: Normalized confusion matrix in Exercise 3

The object recognition steps have been implemented in the pcl_callback() function within the Python template. The resulting object recognition is shown in figure 6 and 7:

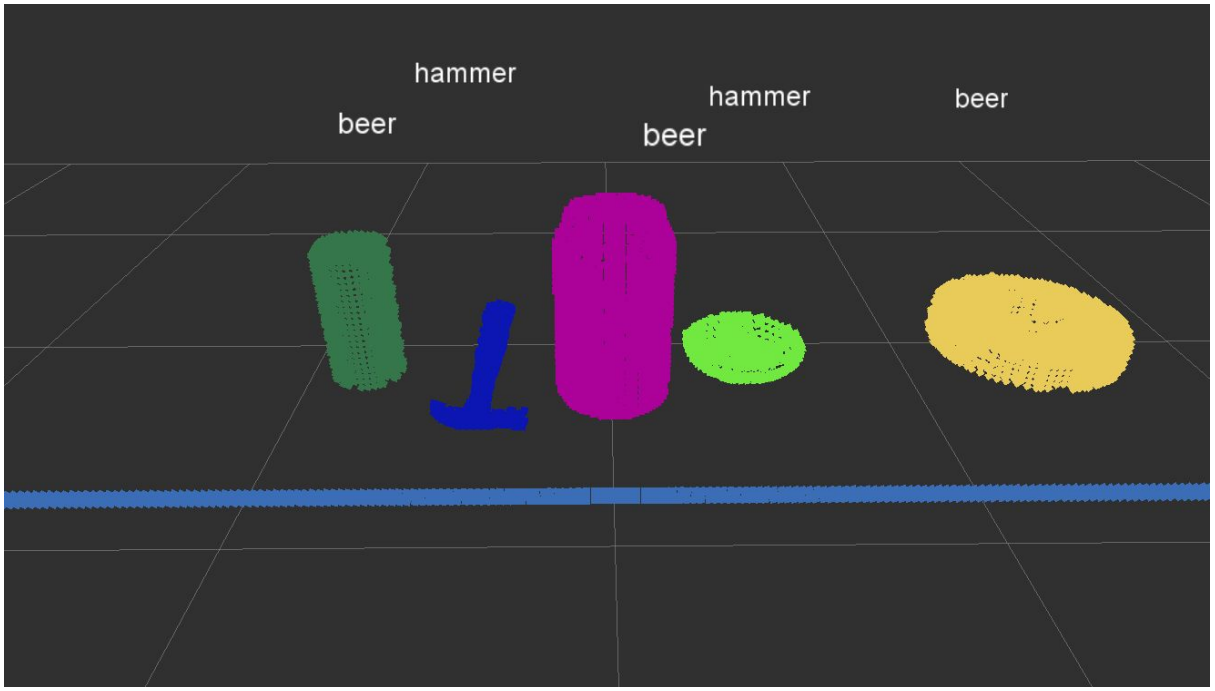

Figure 6: Object recognition

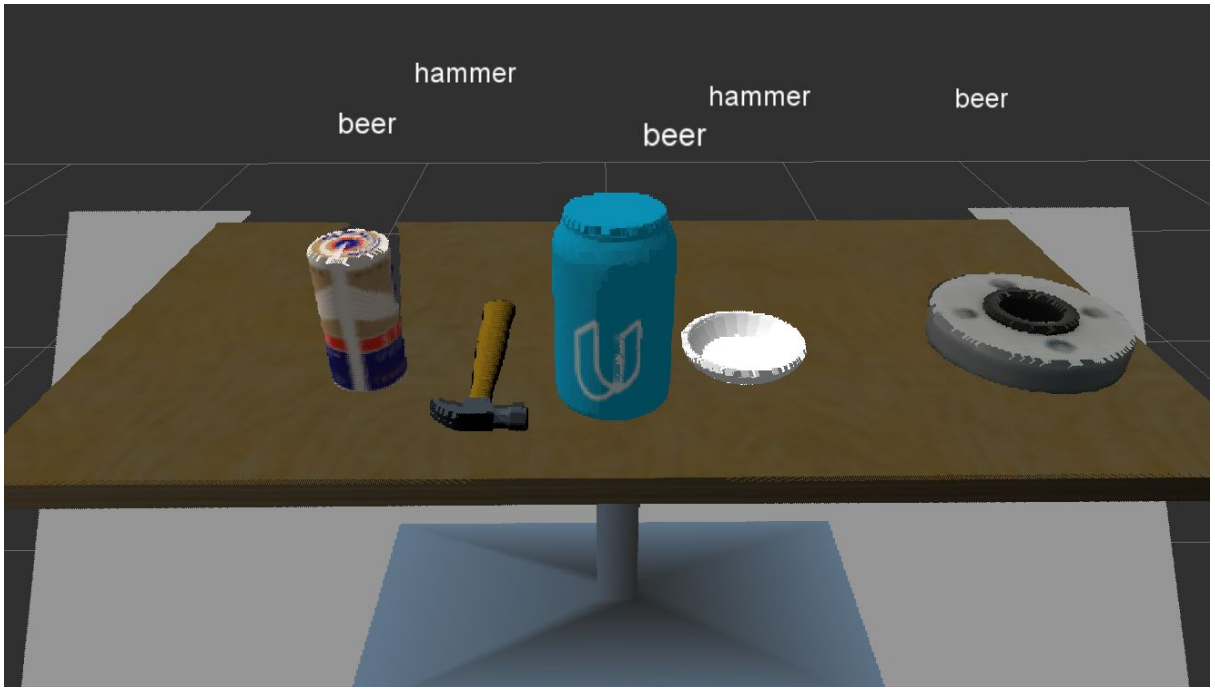

Figure 7: The original image with identified objects

As can be seen in the figures, the object recognition did not work that well. The hammer and beer-can are recognized but neither the bowl nor disc-part. This was due to an error in the object recognition part of the Python code (line: “pcl_cluster = cluster_cloud.extract(pts_list)” should be changed to “pcl_cluster = cloud_objects.extract(pts_list)).

Furthermore, the SVM training could have been improved by increasing the number of iterations in the above-mentioned for-loop and, also, by changing the SVM kernel from ‘linear’ to ‘rbf’ to produce more complex decision boundaries (see documentation for scikit learn: http://scikit-learn.org/stable/modules/svm.html). These improvements have been implemented in the next section.

The Python code for Exercise 3 is shown in the appendix at the end of the report.

Pick and Place Setup

1. 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 Pick & Place request output them to .yaml format.

1.1 Feature capturing and SVM training

Following improvements were made during feature capturing and SVM training:

a) In the script “capture_features.py”, the for-loop *for i in range(5):* has been changed to *for i in range(25):* (ie. 25 iterations instead of 5).

b) As before, in the script “capture_features.py”, the HSV color model has been used instead of the RGB model by setting parameter “using_hsv” to True.

```
chists = compute_color_histograms(ros_cluster, using_hsv=True)
```

c) To improve the accuracy of the SVM classifier, the kernel has been changed from ‘linear’ to ‘rbf’ in file “train_svm.py”. The resulting confusion matrix is shown in figure 8.

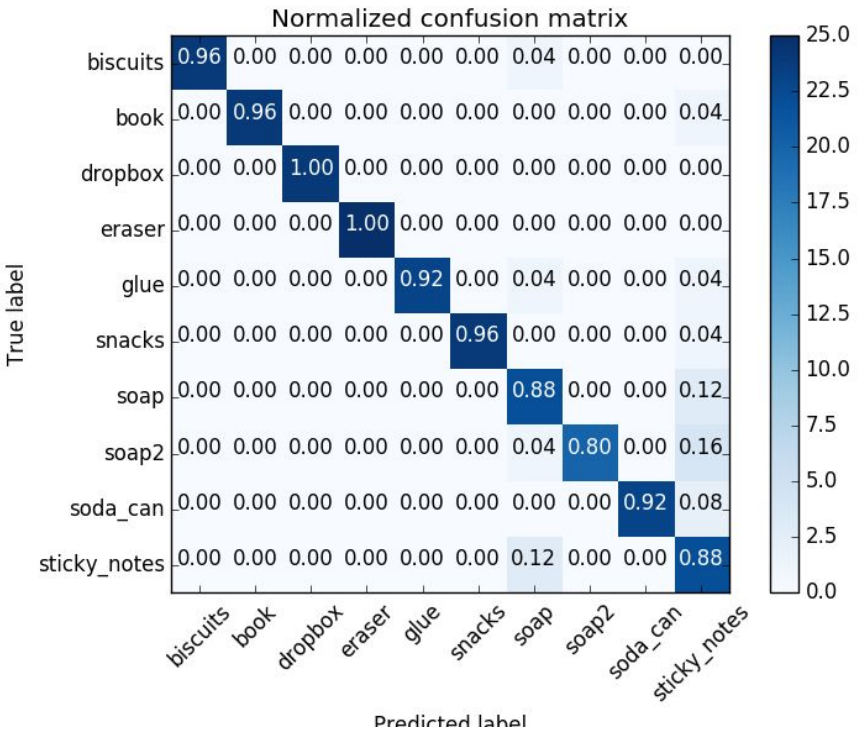

Figure 8: Confusion matrix (normalized)

The model has an accuracy of 93% according to output in the terminal:

Features in Training Set: 250
Invalid Features in Training set: 1
Scores: [0.9 0.92 0.92 0.98 0.91836735]
Accuracy: 0.93 (+/- 0.05)
accuracy score: 0.927710843373

1.2 Simulation of the three test configurations

The code for the pick and place simulation is contained in python script: **3d_perception.py** (see attached file). This script includes the code from Exercises 1, 2 and 3 for filtering, clustering and object identification.

Following parameters have been used in the simulation:

	Filtering/Clustering/Recognition	**Parameters**	**Comment**
1.	Statistical Outlier Filter	outlier_filter.set_mean_k(50) x = 1.0	The number of neighboring points to analyze for any given point Threshold scale factor
2.	Voxel Grid Downsampling	LEAF_SIZE = 0.01	Leaf size for downsampling
3.	Pass-through Filtering	filter_axis = 'z' axis_min = 0.6 axis_max = 1.0	The axis and range to the pass-through filter object (in meters)
4.	RANSAC Plane Segmentation	max_distance = 0.005	Max distance for a point to be considered fitting the model (in meters)
5.	Euclidean Clustering	ec.set_ClusterTolerance(0.02) ec.set_MinClusterSize(50) ec.set_MaxClusterSize(20000)	Tolerances for distance threshold (in meters) as well as minimum and maximum cluster size (in points)
6.	Object Recognition	using_hsv=True	HSV color coding

Figure 9: Parameters used in the simulation

Test Scene 1:

The terminal output for test scene 1 is as follows:

robond@udacity:~/catkin_ws$ rosrun pr2_robot 3d_perception.py
[INFO] [1502641168.481918, 998.649000]: Detected 6 objects: ['biscuits', 'soap', 'soap2', 'dropbox', 'dropbox', 'dropbox']

The dropbox was included in feature capturing hence the dropbox appears in the terminal output, as part of the detected objects. The detected objects are shown in figure 10:

Figure 10: Clustering segmentation and object identification for test1.world

As can be seen, the PR2 robot recognizes 3/3 objects (100%) in test scene 1. The output has been stored in file “output_1.yaml”.

Test Scene 2:

The terminal output for test scene 2 is as follows:

robond@udacity:~/catkin_ws$ rosrun pr2_robot 3d_perception.py
[INFO] [1502640162.971885, 1116.652000]: Detected 8 objects: ['biscuits', 'book', 'soap', 'soap2', 'dropbox', 'dropbox', 'dropbox', 'dropbox']

The dropbox was included in feature capturing hence the dropbox appears in the terminal output, as part of the detected objects. The detected objects are shown in figure 11:

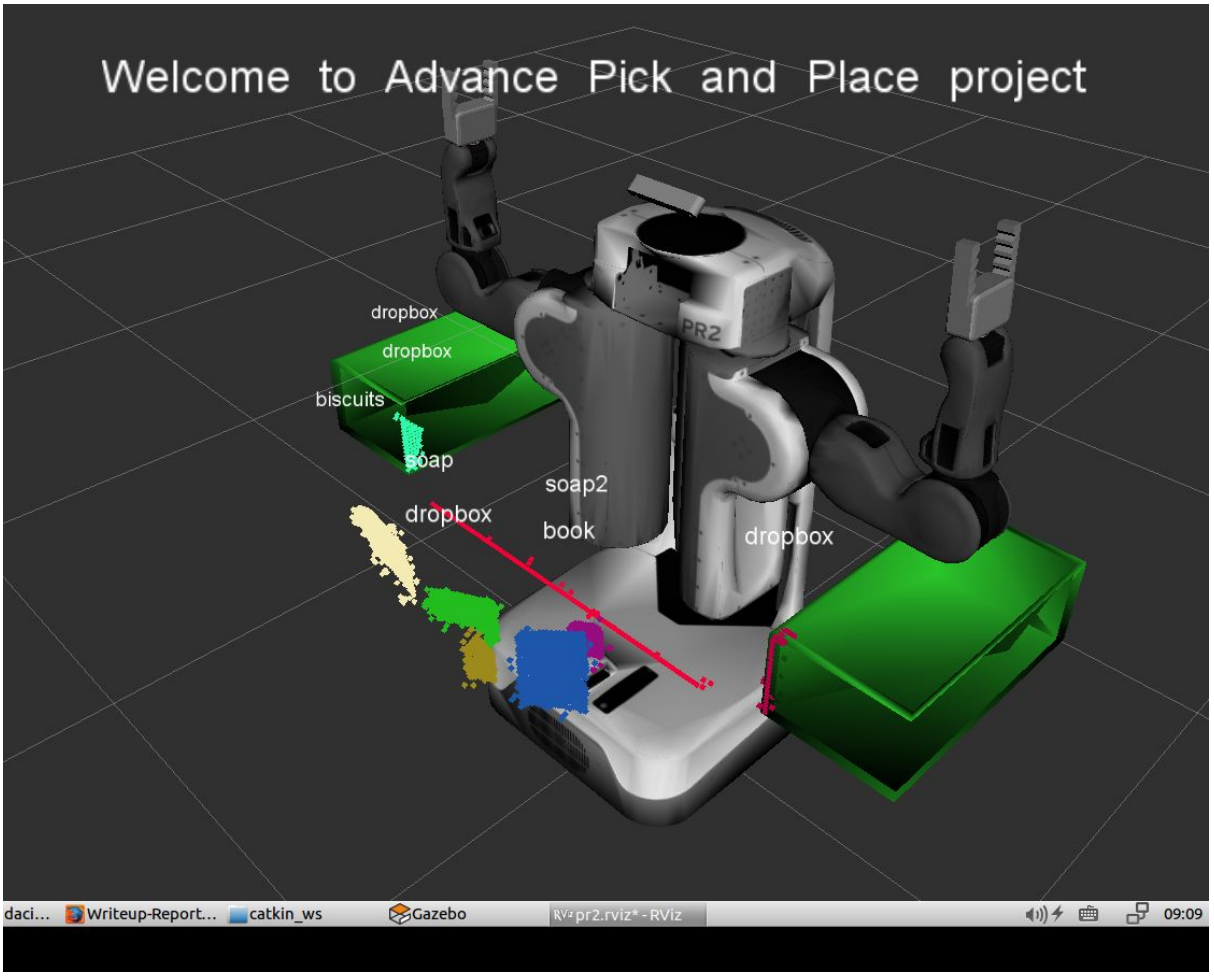

Figure 11: Clustering segmentation and object identification for test2.world

As can be seen, the PR2 robot recognizes 4/5 objects (80%) in test scene 2. The output has been stored in file “output_2.yaml”.

Test Scene 3

The terminal output for test scene 3 is as follows:

robond@udacity:~/catkin_ws$ rosrun pr2_robot 3d_perception.py
[INFO] [1502639218.918010, 1332.562000]: Detected 11 objects: ['snacks', 'biscuits', 'book', 'soap', 'eraser', 'soap2', 'sticky_notes', 'dropbox', 'dropbox', 'dropbox', 'dropbox']

The dropbox was included in feature capturing hence the dropbox appears in the terminal output, as part of the detected objects. The detected objects are shown in figure 12:

Figure 12: Clustering segmentation and object identification for test3.world

As can be seen, the PR2 robot recognizes 7/8 objects (88%) in test scene 3. The output has been stored in file “output_3.yaml”.

1.3 Future improvements

The accuracy of the SVM classifier is good (93%) , but it could be improved even more by modifying the parameters in the Support Vector Machine (SVM). The main code (script “3d_perception.py”) can also be expanded to include collision avoidance and robot motion.

References

SVM 2017, Documentation on Support Vector Machines. Available online: :http://scikit-learn.org/stable/modules/svm.html

Appendix 1

Python code for Exercise 1:

```
# Import PCL module
import pcl

# Load Point Cloud file
cloud = pcl.load_XYZRGB('tabletop.pcd')

# ******************
# 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!
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()
filename = 'voxel_downsampled.pcd'
pcl.save(cloud_filtered, filename)


# ********************
# PassThrough filter
# ********************

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

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

# Finally use the filter function to obtain the resultant point cloud.
cloud_filtered = passthrough.filter()
filename = 'pass_through_filtered.pcd'
pcl.save(cloud_filtered, filename)
```

```
# **********************************
# 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()


# ****************
# Extract inliers
# ****************

# Extract inliers
extracted_inliers = cloud_filtered.extract(inliers, negative=False)
filename = 'extracted_inliers.pcd'
pcl.save(extracted_inliers, filename)


# ******************
# Extract outliers
# ******************

extracted_outliers = cloud_filtered.extract(inliers, negative=True)
filename = 'extracted_outliers.pcd'
pcl.save(extracted_outliers, filename)
```

Python code for Exercise 2:

```
#!/usr/bin/env python

# Import modules
from pcl_helper import *

# Define functions as required

# Callback function for your Point Cloud Subscriber
def pcl_callback(PointCloud2_msg):

    # Convert ROS msg to PCL data

    cloud = ros_to_pcl(PointCloud2_msg)

    # Voxel Grid Downsampling

    # 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!
    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()
    filename = 'voxel_downsampled.pcd'
    pcl.save(cloud_filtered, filename)

    # PassThrough Filter

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

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

    # Finally use the filter function to obtain the resultant point cloud.
    cloud_filtered = passthrough.filter()
    filename = 'pass_through_filtered.pcd'
    pcl.save(cloud_filtered, filename)
```

```
# 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()


# Extract inliers and outliers

# Extract inliers
extracted_inliers = cloud_filtered.extract(inliers, negative=False)
filename = 'extracted_inliers.pcd'
pcl.save(extracted_inliers, filename)

cloud_table = extracted_inliers   #Assign inliers to cloud_table

extracted_outliers = cloud_filtered.extract(inliers, negative=True)
filename = 'extracted_outliers.pcd'
pcl.save(extracted_outliers, filename)

cloud_objects = extracted_outliers    #Assign outliers to cloud_objects



# Euclidean Clustering

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(0.02)
ec.set_MinClusterSize(50)
ec.set_MaxClusterSize(20000)
```

```
    # Search the k-d tree for clusters
    ec.set_SearchMethod(tree)
    # Extract indices for each of the discovered clusters
    cluster_indices = ec.Extract()

    # 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)

    # 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)

    # Publish ROS messages
    pcl_objects_pub.publish(ros_cloud_objects)
    pcl_table_pub.publish(ros_cloud_table)
    pcl_cluster_pub.publish(ros_cluster_cloud)


if __name__ == '__main__':

    # ROS node initialization

    rospy.init_node('clustering', anonymous=True)

    # Create Subscribers

    pcl_sub = rospy.Subscriber("/sensor_stick/point_cloud", pc2.PointCloud2, pcl_callback,
    queue_size=1)

    # Create Publishers

    pcl_objects_pub = rospy.Publisher("/pcl_objects", PointCloud2, queue_size=1)
    pcl_table_pub = rospy.Publisher("/pcl_table", PointCloud2, queue_size=1)
    pcl_cluster_pub = rospy.Publisher("/pcl_cluster", PointCloud2, queue_size=1)
```

```
    # Initialize color_list
    get_color_list.color_list = []


    # TODO: Spin while node is not shutdown

    while not rospy.is_shutdown():
        rospy.spin()
```

Python code for Exercise 3:

```
#!/usr/bin/env python

import numpy as np
import sklearn
from sklearn.preprocessing import LabelEncoder

import pickle

from sensor_stick.srv import GetNormals
from sensor_stick.features import compute_color_histograms
from sensor_stick.features import compute_normal_histograms
from visualization_msgs.msg import Marker

from sensor_stick.marker_tools import *
from sensor_stick.msg import DetectedObjectsArray
from sensor_stick.msg import DetectedObject
from sensor_stick.pcl_helper import *

def get_normals(cloud):
    get_normals_prox = rospy.ServiceProxy('/feature_extractor/get_normals', GetNormals)
    return get_normals_prox(cloud).cluster

# Callback function for your Point Cloud Subscriber
def pcl_callback(PointCloud2_msg):

# Exercise-2:

    # Convert ROS msg to PCL data
    cloud = ros_to_pcl(PointCloud2_msg)

    # Voxel Grid Downsampling

    # 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!
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()
filename = 'voxel_downsampled.pcd'
pcl.save(cloud_filtered, filename)


# PassThrough Filter

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

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

# Finally use the filter function to obtain the resultant point cloud.
cloud_filtered = passthrough.filter()
filename = 'pass_through_filtered.pcd'
pcl.save(cloud_filtered, filename)


# 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()
```

```
# Extract inliers and outliers

extracted_inliers = cloud_filtered.extract(inliers, negative=False)
filename = 'extracted_inliers.pcd'
pcl.save(extracted_inliers, filename)

cloud_table = extracted_inliers   #Assign inliers to cloud_table

extracted_outliers = cloud_filtered.extract(inliers, negative=True)
filename = 'extracted_outliers.pcd'
pcl.save(extracted_outliers, filename)

cloud_objects = extracted_outliers    #Assign outliers to cloud_objects


# Euclidean Clustering

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(0.02)
ec.set_MinClusterSize(50)
ec.set_MaxClusterSize(20000)
# Search the k-d tree for clusters
ec.set_SearchMethod(tree)
# Extract indices for each of the discovered clusters
cluster_indices = ec.Extract()

# 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)


    # 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)


    # Publish ROS messages
    pcl_objects_pub.publish(ros_cloud_objects)
    pcl_table_pub.publish(ros_cloud_table)
    pcl_cluster_pub.publish(ros_cluster_cloud)


# Exercise-3:

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

        # Compute the associated feature vector
        # Extract histogram features
        # Complete this step just
        # as you did before in capture_features.py
        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
        # retrieve the label for the result
        # and add it to detected_objects_labels list
        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)

        rospy.loginfo('Detected {} objects: {}'.format(len(detected_objects_labels),
        detected_objects_labels))

        # Publish the list of detected objects
        # This is the output you'll need to complete the upcoming project!
        detected_objects_pub.publish(detected_objects)


    if __name__ == '__main__':

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

        # Create Subscribers
        pcl_sub = rospy.Subscriber("/sensor_stick/point_cloud", pc2.PointCloud2, pcl_callback,
        queue_size=1)

        # Create Publishers
        # Here you need to create two publishers
        # Call them object_markers_pub and detected_objects_pub
        # Have them publish to "/object_markers" and "/detected_objects" with
        # Message Types "Marker" and "DetectedObjectsArray" , respectively

        pcl_objects_pub = rospy.Publisher("/pcl_objects", PointCloud2, queue_size=1)
        pcl_table_pub = rospy.Publisher("/pcl_table", PointCloud2, queue_size=1)
        pcl_cluster_pub = rospy.Publisher("/pcl_cluster", PointCloud2, queue_size=1)

        object_markers_pub   = rospy.Publisher("/object_markers", Marker, queue_size=1)
        detected_objects_pub = rospy.Publisher("/detected_objects", DetectedObjectsArray,
        queue_size=1)

        # Load Model From disk
        model = pickle.load(open('model.sav', 'rb'))
        clf = model['classifier']
        encoder = LabelEncoder()
        encoder.classes_ = model['classes']
        scaler = model['scaler']

        # Initialize color_list
        get_color_list.color_list = []

        # Spin while node is not shutdown
        while not rospy.is_shutdown():
            rospy.spin()
```