DECEMBER 6, 2018

RoboND-Perception-Project

GOALS OF THE PROJECT:

- The goal of this project is to apply learned perception techniques to pr2 pick & place robot.
- Pr2 robot is mounted with RGB-D camera that gives us required data to form 3D point cloud.
- By applying filters we reduce noise from camera data.
- By applying segmentation techniques we get inliers & outliers
- Finally every object is detected by object recognition techniques

PROJECT STEPS:

1- Set up your ROS Workspace
2- Complete Perception Exercises 1, 2 and 3, which comprise the project perception pipeline
3- Downloaded the project repository into the src directory of your ROS Workspace
4- assimilate your work from previous exercises to successfully complete a tabletop pick and place operation using PR2
5- implement a perception pipeline
6- output yaml files that has the data of objects in each scenarios
7- extra challenges

RUBRIC POINTS:

APPLYING TECHNIQUES FROM EXERCISE 1

1ST Outlier Removal Filter

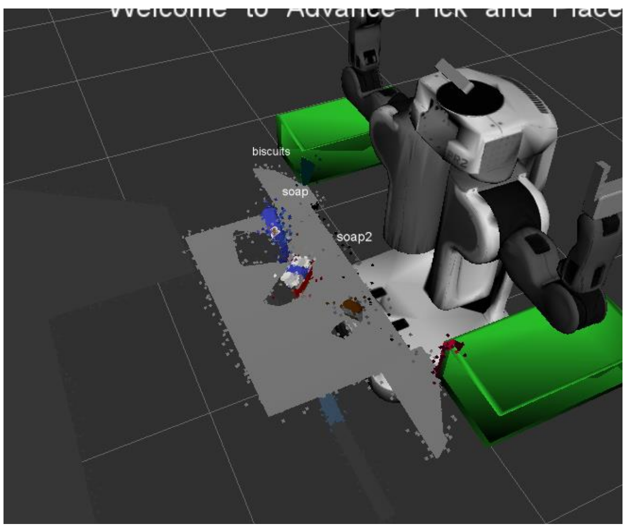
1) FUNCTION

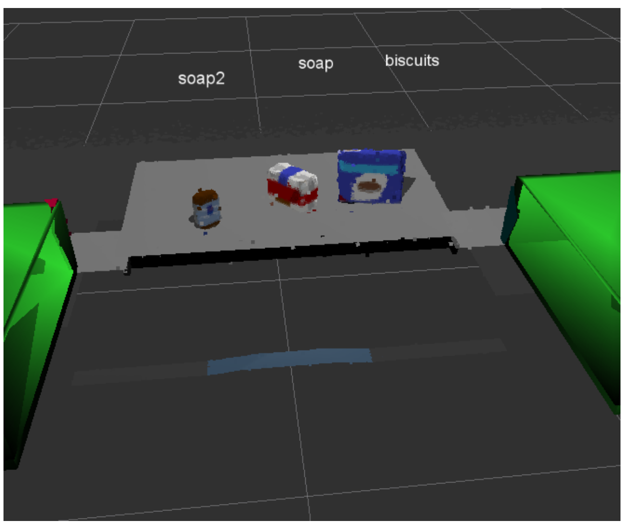
Because camera row data has some noise and distortion this filter is applied to remove this noise, And to make point cloud more efficient for the upcoming processes

2) CODE

```
# TODO: Statistical Outlier Filtering
outlier_filter = cloud.make_statistical_outlier_filter()
# Set the number of neighboring points to analyze for any given point
outlier_filter.set_mean_k(20)
# Set threshold scale factor
x = .2
# 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_filtered = outlier_filter.filter()
```

3) SCREENSHOTS (BEFORE ,AFTER)





2nd Voxel Grid

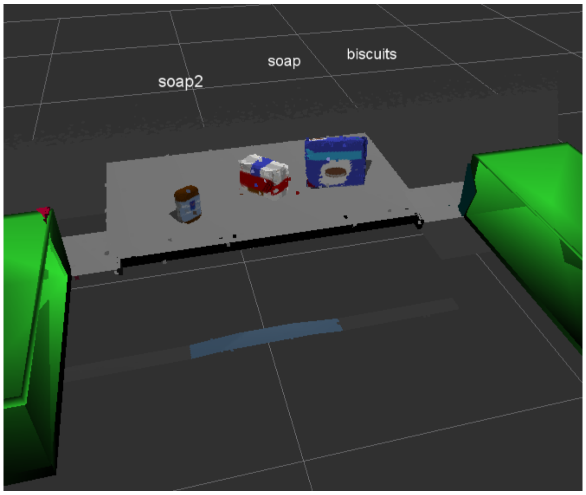
1) FUNCTION

- Because cameras obtained data varies in respect of its quality ,High quality of the camera gives us data with high size.
- When data size increases it will be a waste of processing power to process all this data ,and can make our project slow
- Thus, Voxel Grid Down sampling is applied to define the size of each point in the cloud
- Here the most suitable case was when leaf size=.005

2) Code

```
# TODO: Voxel Grid Downsampling
    vox = cloud_filtered.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 = .005
    # 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 = vox.filter()
```

3) SREENSHOTS



3rd Pass-Through Filter

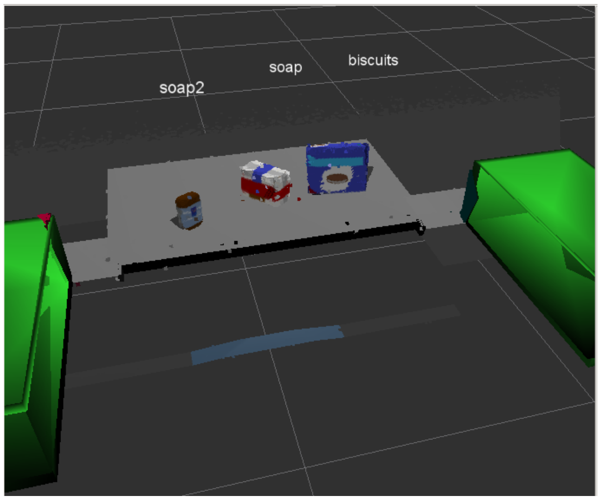
1) FUNCTION

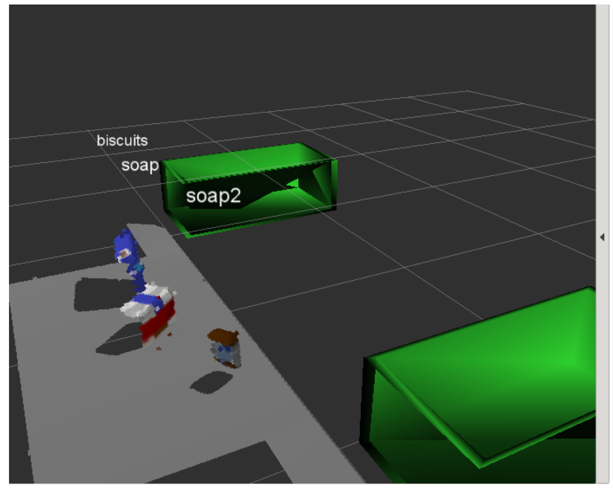
- This technique is applied to limit the data obtained from RGB-D camera to make it only view only needed area
- In this case the only needed view was the table and objects on it, So we applied Pass Through Filter to x and z axis

2) CODE

```
# TODO: PassThrough Filter
    passthrough = cloud_filtered_vox.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
    passthrough.set_filter_limits(axis_min, axis_max)
    cloud_filtered_pass = passthrough.filter()
    #filter x
    passthrough = cloud_filtered_pass.make_passthrough_filter()
    filter_axis = 'x'
    passthrough.set_filter_field_name(filter_axis)
    axis_min = .35
    axis_max = 1
    passthrough.set_filter_limits(axis_min, axis_max)
    cloud_filtered_pass = passthrough.filter()
```

3) SCREENSHOTS (Before,After)





4th RANSAC Plane

1) Function

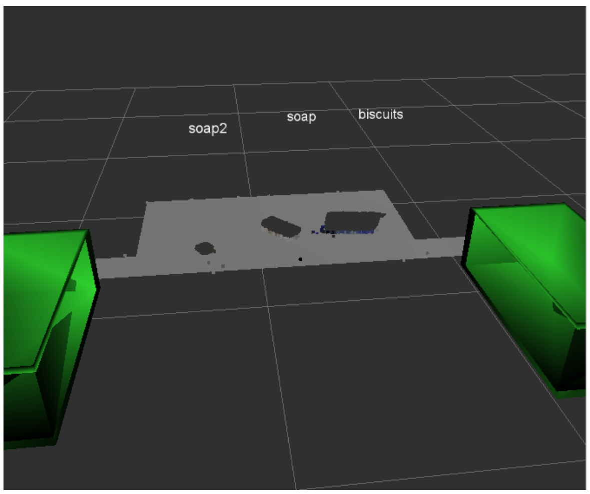
- To separate the table and object
- We used plane model
- At the end we gets two point clouds one contains the objects and the other one contains the table

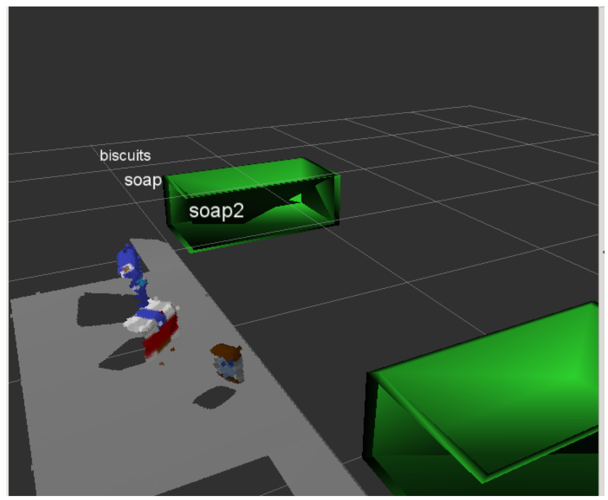
2) Code

```
# TODO: RANSAC Plane Segmentation
    seg = cloud_filtered_pass.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 = .012
    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
extracted_inliers = cloud_filtered_pass.extract(inliers, negative=False)
extracted_outliers = cloud_filtered_pass.extract(inliers, negative=True)
```

3) Screenshots





APPLYING TECHNIQUES FROM EXERCISE 2

1st Euclidean Clustering

1) Function

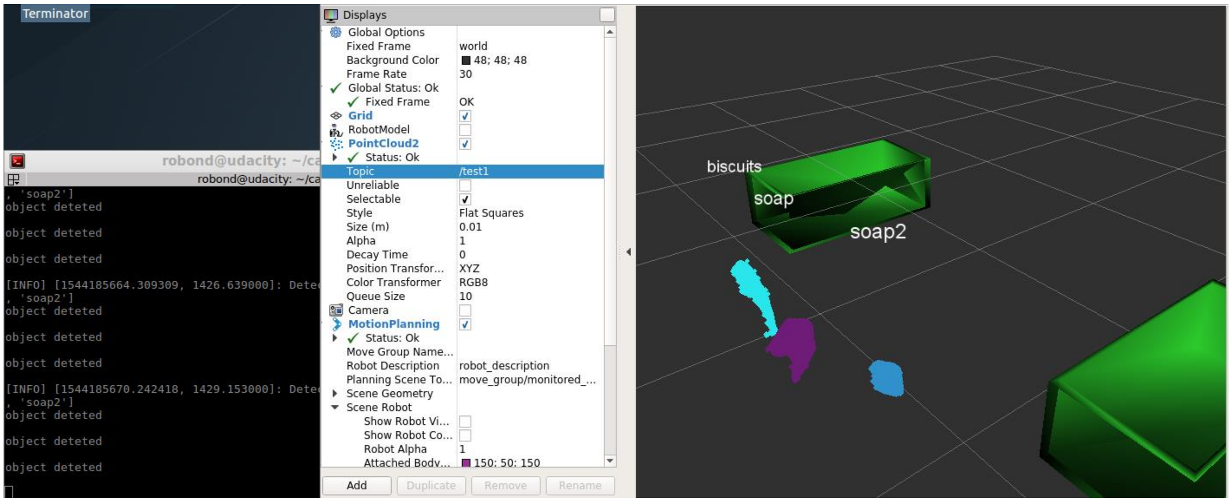
- This code is applied to separate each object from objects point cloud
- This code seperates objects by identifying the density of point coulds in a particular space
- Parameters needed to be modified is:
    - minimum distance between two points to form segment
    - minimum points in one segment
    - maximum points in one segment

2) Code

```
# TODO: Euclidean Clustering
    cloud_objects_white=XYZRGB_to_XYZ(extracted_outliers)
    tree = cloud_objects_white.make_kdtree()

    # TODO: Create Cluster-Mask Point Cloud to visualize each cluster separately
    ec = cloud_objects_white.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(.03)
    ec.set_MinClusterSize(50)
    ec.set_MaxClusterSize(3000)
    # Search the k-d tree for clusters
    ec.set_SearchMethod(tree)
    # Extract indices for each of the discovered clusters
    cluster_indices = ec.Extract()
    #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([cloud_objects_white[indice][0],
                                            cloud_objects_white[indice][1],
                                            cloud_objects_white[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)
```

3) Screenshots



APPLYING TECHNIQUES FROM EXERCISE 3

1st Calculating Color Histograms, Surface Normals

1) Function

- To Calculate a special pattern based on color histograms and surface normals features
- This enables us to find special pattern for each segment in our project
- To make our pattern more optimized we took each feature more tan one time to make the code accurate
- Here we took 32 bins for color histogram ,20 different orientation for surface normals

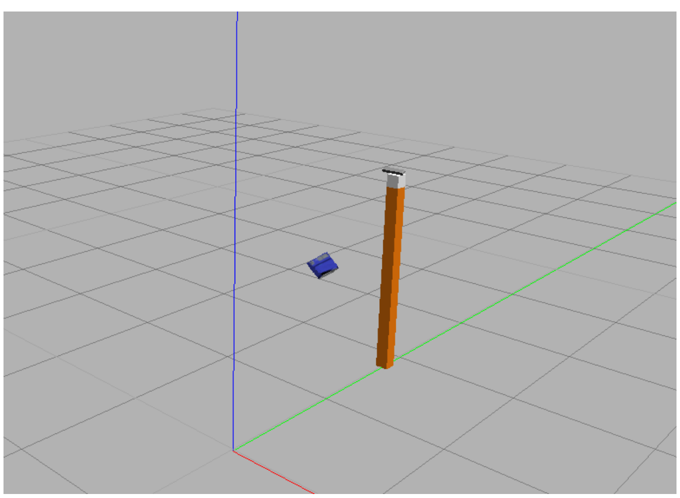
2) Code

```
def compute_color_histograms(cloud,nbins=32, using_hsv=True):
    # Compute histograms for the clusters
    point_colors_list = []
    # Step through each point in the point cloud
    for point in pc2.read_points(cloud, skip_nans=True):
        rgb_list = float_to_rgb(point[3])
        if using_hsv:
```

```
        point_colors_list.append(rgb_to_hsv(rgb_list) * 255)
      else:
        point_colors_list.append(rgb_list)

    # Populate lists with color values
    channel_1_vals = []
    channel_2_vals = []
    channel_3_vals = []

    for color in point_colors_list:
      channel_1_vals.append(color[0])
      channel_2_vals.append(color[1])
      channel_3_vals.append(color[2])

    # TODO: Compute histograms
    ch1_hist =np.histogram(channel_1_vals,nbins, (0, 256))
    ch2_hist = np.histogram(channel_2_vals,nbins, (0, 256))
    ch3_hist = np.histogram(channel_3_vals,nbins, (0, 256))
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((ch1_hist[0], ch2_hist[0], ch3_hist[0])).astype(np.float64)
    # Normalize the result
    norm_features = hist_features / np.sum(hist_features)
    # TODO: Concatenate and normalize the histograms

    # Generate random features for demo mode.
    # Replace normed_features with your feature vector
    normed_features = norm_features
    return normed_features

def compute_normal_histograms(normal_cloud):
    norm_x_vals = []
    norm_y_vals = []
    norm_z_vals = []
```

```
for norm_component in pc2.read_points(normal_cloud,
                                      field_names = ('normal_x', 'normal_y', 'normal_z'),
                                      skip_nans=True):
    norm_x_vals.append(norm_component[0])
    norm_y_vals.append(norm_component[1])
    norm_z_vals.append(norm_component[2])
# TODO: Compute histograms of normal values (just like with color)
x_hist =np.histogram(norm_x_vals,32, (0, 256))
y_hist = np.histogram(norm_y_vals,32, (0, 256))
z_hist = np.histogram(norm_z_vals,32, (0, 256))
# Concatenate the histograms into a single feature vector
hist_features = np.concatenate((x_hist[0], y_hist[0], z_hist[0])).astype(np.float64)
# Normalize the result
norm_features = hist_features / np.sum(hist_features)
# TODO: Concatenate and normalize the histograms
# Replace normed_features with your feature vector
normed_features = norm_features
return normed_features
```

3) Screenshots



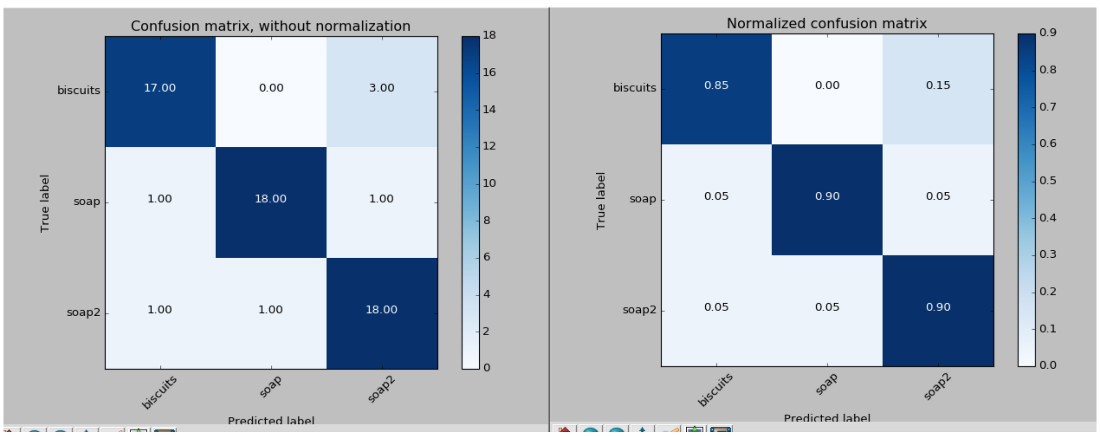
2nd Training our model with SVM

1) Function

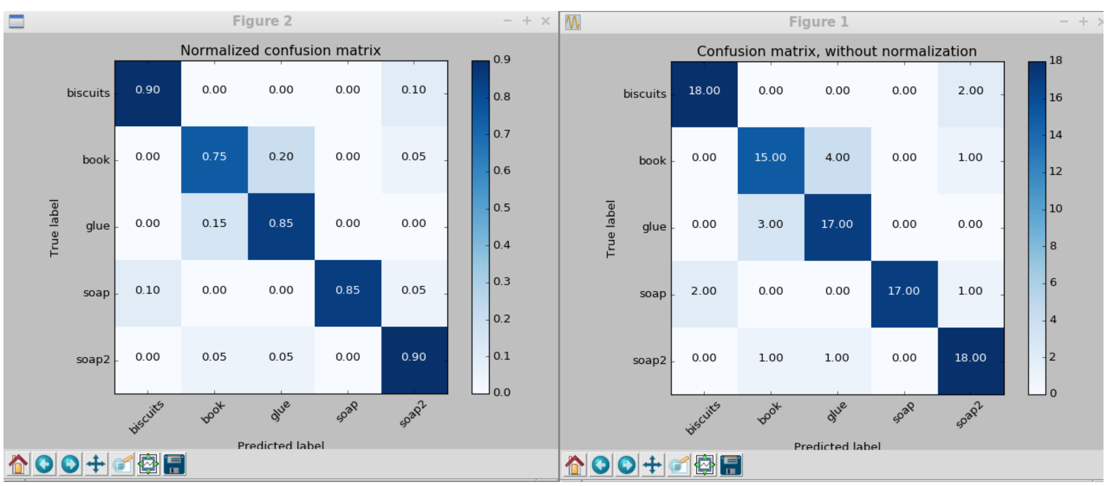
- SVM is a machine learning technique is used train our captured_features for each element In the scene to form a model that recognizes each element in our segments
- Model accuracy for each object is obtained from confusion matrix

2) Confusion Matrix for each world

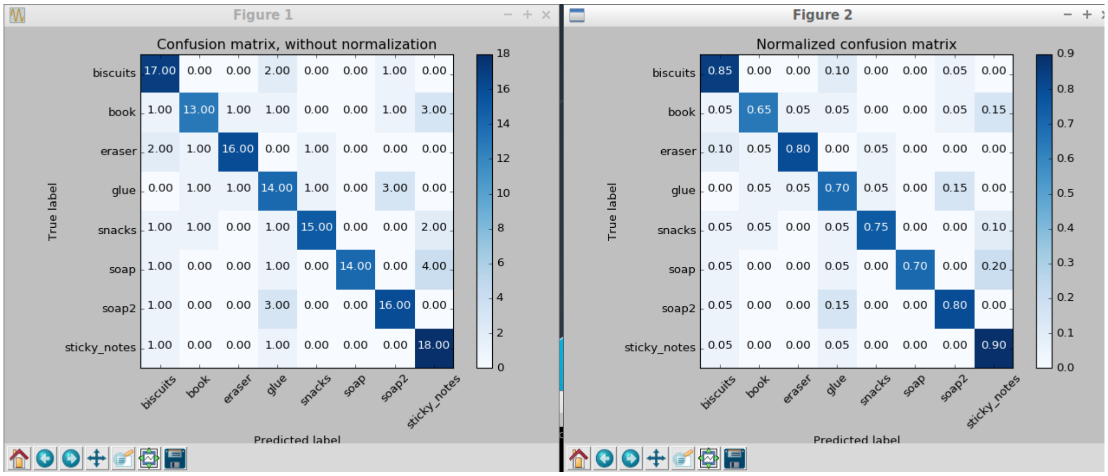
- World 1



- World2



- World 3



PUBLISHING MESSAGES NEEDED & OUTPUT DATA

1st Publish Detected Objects, Call pr2_mover function

1) Code

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


    # Suggested location for where to invoke your pr2_mover() function within pcl_callback()
    # Could add some logic to determine whether or not your object detections are robust
    # before calling pr2_mover()
    try:
        pr2_mover(detected_objects)
    except rospy.ROSInterruptException:
        pass
```

2nd Implement pr2_mover function

1) Function

- To check if elements in object list is in the deteted objects
- Define pick place and place pose of each detected objected
- Define arm name that will grab each object
- Define list name of objects
- Outputs all detected objects data to yml file

2) Code

```
def pr2_mover(object_list):
    # TODO: Initialize variables
    #declare
    test_scene_num = Int32()
    object_name = String()
    arm_name = String()
    pick_pose = Pose()
    place_pose = Pose()
    #initialize
    test_scene_num.data = 1
    object_list_param = rospy.get_param('/object_list')
    dropbox_list_param = rospy.get_param('/dropbox')
    # TODO: Parse parameters into individual variables
    labels = []
    centroids = [] # to be list of tuples (x, y, z)
    dict_list = []
    centroid =[]
    for object in object_list:
        labels.append(object.label)
        points_arr = ros_to_pcl(object.cloud).to_array()
        centroid=np.mean(points_arr, axis=0)[:3]
        centroid = [np.asscalar(centroid[0]),np.asscalar(centroid[1]),np.asscalar(centroid[2])]
        centroids.append(centroid)


    # TODO: Create 'place_pose' for the object
```

```
    for i in range(0, len(object_list_param)):
        object_name_l = object_list_param[i]['name']
        object_group = object_list_param[i]['group']
        for i in range(0,len(labels)):
                detected_object_name=labels[i]
                if(object_name_l==detected_object_name):
                    print('object deteted \n')
                    object_name.data=object_name_l
                    pick_pose.position.x=centroids[i][0]
                    pick_pose.position.y=centroids[i][1]
                    pick_pose.position.z=centroids[i][2]
                    for i in range(0,len(dropbox_list_param)):
                        dropbox_name=dropbox_list_param[i]['name']
                        dropbox_group=dropbox_list_param[i]['group']
                        dropbox_pos=dropbox_list_param[i]['position']
                        if(object_group==dropbox_group):
                            arm_name.data=dropbox_name
                            place_pose.position.x=dropbox_pos[0]
                            place_pose.position.y=dropbox_pos[1]
                            place_pose.position.z=dropbox_pos[2]
                    yaml_dict = make_yaml_dict(test_scene_num, arm_name, object_name, pick_pose,
place_pose)
                    dict_list.append(yaml_dict)
    send_to_yaml('output1.yml', dict_list)
                         # Wait for 'pick_place_routine' service to come up
                #rospy.wait_for_service('pick_place_routine')
                         #try:
                             #pick_place_routine = rospy.ServiceProxy('pick_place_routine', PickPlace)
                             # TODO: Insert your message variables to be sent as a service request
                             #resp = pick_place_routine(test_scene_num, object_name, arm_name,
pick_pose, place_pose)
                             #print ("Response: ",resp.success)
```

```
                #except rospy.ServiceException, e:
                  #print "Service call failed: %s"%e
```

APPLY OBJECT RECOGNITION MODEL IN EACH WORLD

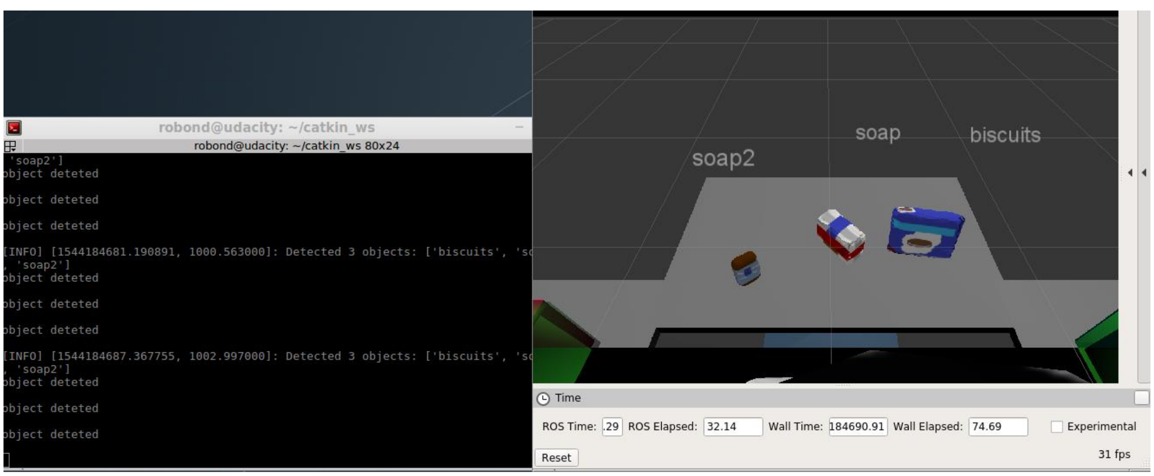
Code

```
detected_objects_labels = []
  detected_objects = []

  # Classify the clusters! (loop through each detected cluster one at a time)
  for index, pts_list in enumerate(cluster_indices):

    # Grab the points for the cluster from the extracted outliers (cloud_objects)
    pcl_cluster_object = extracted_outliers.extract(pts_list)



    # TODO: convert the cluster from pcl to ROS using helper function
    cluster_ros=pcl_to_ros(pcl_cluster_object)



    # Extract histogram features
    chists = compute_color_histograms(cluster_ros,nbins=32 ,using_hsv=True)
    normals = get_normals(cluster_ros)
    nhists = compute_normal_histograms(normals)
    feature = np.concatenate((chists, nhists))
    #labeled_features.append([feature, model_name])
    # TODO: complete this step just as is covered in capture_features.py

    # 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)))
```

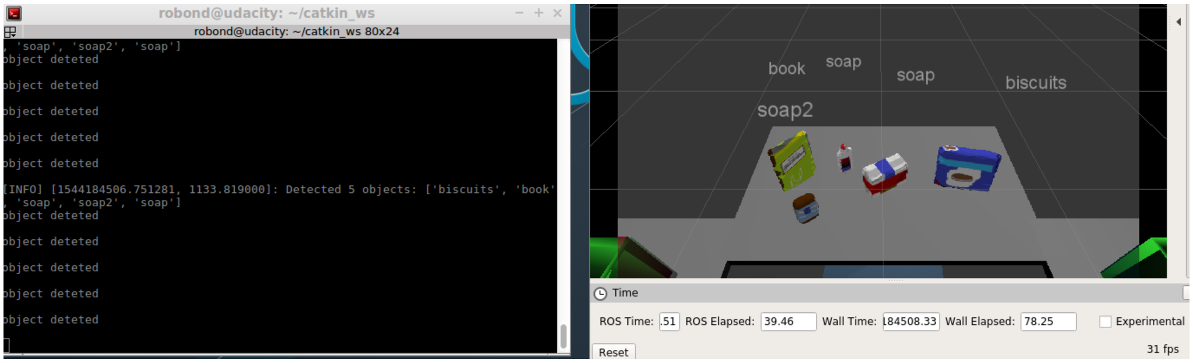
```
        label = encoder.inverse_transform(prediction)[0]
        detected_objects_labels.append(label)

        # Publish a label into RViz
        label_pos = list(cloud_objects_white[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 = cluster_ros
        detected_objects.append(do)
```

Results

- **World 1**



- **World 2**


robond@udacity: ~/catkin_ws
robond@udacity: ~/catkin_ws 80x24
, 'soap', 'soap2', 'soap']
object deteted
object deteted
object deteted
object deteted
object deteted
[INFO] [1544184506.751281, 1133.819000]: Detected 5 objects: ['biscuits', 'book'
, 'soap', 'soap2', 'soap']
object deteted
object deteted
object deteted
object deteted
object deteted
book
soap
soap
biscuits
soap2
Time
ROS Time: .51 ROS Elapsed: 39.46 Wall Time: 184508.33 Wall Elapsed: 78.25
Experimental
Reset
31 fps

- **World 3**


object deteted
object deteted
[INFO] [1544184313.686625, 1455.381000]: Detected 8 objects: ['s
s', 'book', 'soap', 'eraser', 'soap2', 'sticky_notes', 'glue']
object deteted
object deteted
object deteted
object deteted
object deteted
object deteted
object deteted
eraser
glue
soap
biscuits
sticky_notes
book
soap2
snacks