Writeup / README

<u>Perception Project Objective:</u>

In the PR2 Pick and Place simulator, there are three different tabletop configurations. The goal was to be succeeded in recognizing:
100% (3/3) objects in test1.world
80% (4/5) objects in test2.world
75% (6/8) objects in test3.world
A successful pick and place operation involves passing correct request parameters to the 'pick_place_server'. Hence, for each test scene, correct values must be output to '.yaml' format for following parameters:

- Object Name (Obtained from the pick list)
- Arm Name (Based on the group of an object)
- Pick Pose (Centroid of the recognized object)
- Place Pose (Not a requirement for passing but needed to make the PR2 happy)
- Test Set Number

<u>Exercise 1, 2 and 3 pipeline implemented</u>

Exercise 1: Pipeline for filtering and RANSAC plane fitting.

Steps: The following steps have been taken to implement filtering the point cloud obtained from the camera and achieve RANSAC plane fitting

# Load point cloud file
# Implement Voxel Grid Downsampling
# Implement a Pass Through Filter
# Implement Statistical Outlier Filter
# Implement RANSAC plane segmentation
# Implement inlier extraction
# Extract outliers pcl (only tabletop objects)
# Extract inliers pcl (tabletop)

Exercise 2: Pipeline including clustering for segmentation.

Euclidean Clustering
In order to perform Euclidean Clustering, I first constructed a k-d tree from the cloud_objectspoint cloud.
The k-d tree data structure is used in the Euclidian Clustering algorithm to decrease the computational burden of searching for neighboring points. While other efficient algorithms/data structures for nearest neighbor search exist, PCL's Euclidian Clustering algorithm only supports k-d trees.
To construct a k-d tree, I first converted XYZRGB point cloud to XYZ, because PCL's Euclidean Clustering algorithm requires a point cloud with only spatial information. To create this colorless cloud, I used a function called:
'convert XYZRGB to XYZ'. Next, I constructed a k-d tree from it. In short, the summary of the steps followed are given below:
Steps:
# Create a cluster extraction object
# Set tolerances for distance threshold as well as minimum and maximum cluster size (in points)
# Search the k-d tree for clusters
# Extract indices for each of the discovered clusters

# Assign a color corresponding to each segmented object in scene
# Create new cloud containing all clusters, each with unique color

Exercise 3: Features extracted and SVM trained. Object recognition implemented.

To get started generating features, I launched the 'training.launch' file to bring up the Gazebo environment.
Next, in a new terminal, I ran the 'capture_features.py' script to capture and save features for each of the objects in the environment. This script spawns each object in random orientations and computes features based on the point clouds resulting from each of the random orientations.
At this point, I ran the 'train_svm.py' script to train an SVM classifier on my labeled set of features.
When this runs I get some text output at the terminal regarding overall accuracy of the classifier and two plots will pop up showing the relative accuracy of the classifier for the various objects. This is called 'confusion matrix'.
Running the above command will also result in my trained model being saved in a 'model.sav' file. At this point, the following steps are followed to achieve object recognition.
Steps:

```
# Classify the clusters
# Grab the points for the cluster from the extracted outliers (cloud_objects)
    Iterate:
    # convert the cluster from pcl to ROS
    # Extract histogram features
    # Make the prediction, retrieve the label for the result and add it to detected_objects_labels list
    # Publish a label into RViz
    # Add the detected object to the list of detected objects.

# Publish the list of detected objects
```

Pick and Place Setup:

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' file

Challenges and solutions:

1. Finding correct parameters for the filters, e.g. Leaf size for Voxel grid filter, filter-axis and axis min., max. for the pass through filter, max distance for the RANSAC model, cluster tolerance and min/max cluster size for the Euclidean cluster. All of these were achieved through multiple iterations.

2. Get the right number of training set for the intended number of features and finding a linear confusion matrix, improving accuracy on the training.

This is done through collecting large number of training sets(50) for each object, using HSV color feature instead of RGB, and finding right SVM classifier(rbf, non-linear) to train the model.

3. Making the yaml file converting the desired values into ROS messages

Following are some object detection output images for different worlds/tabletop objects in a cluttered environment:

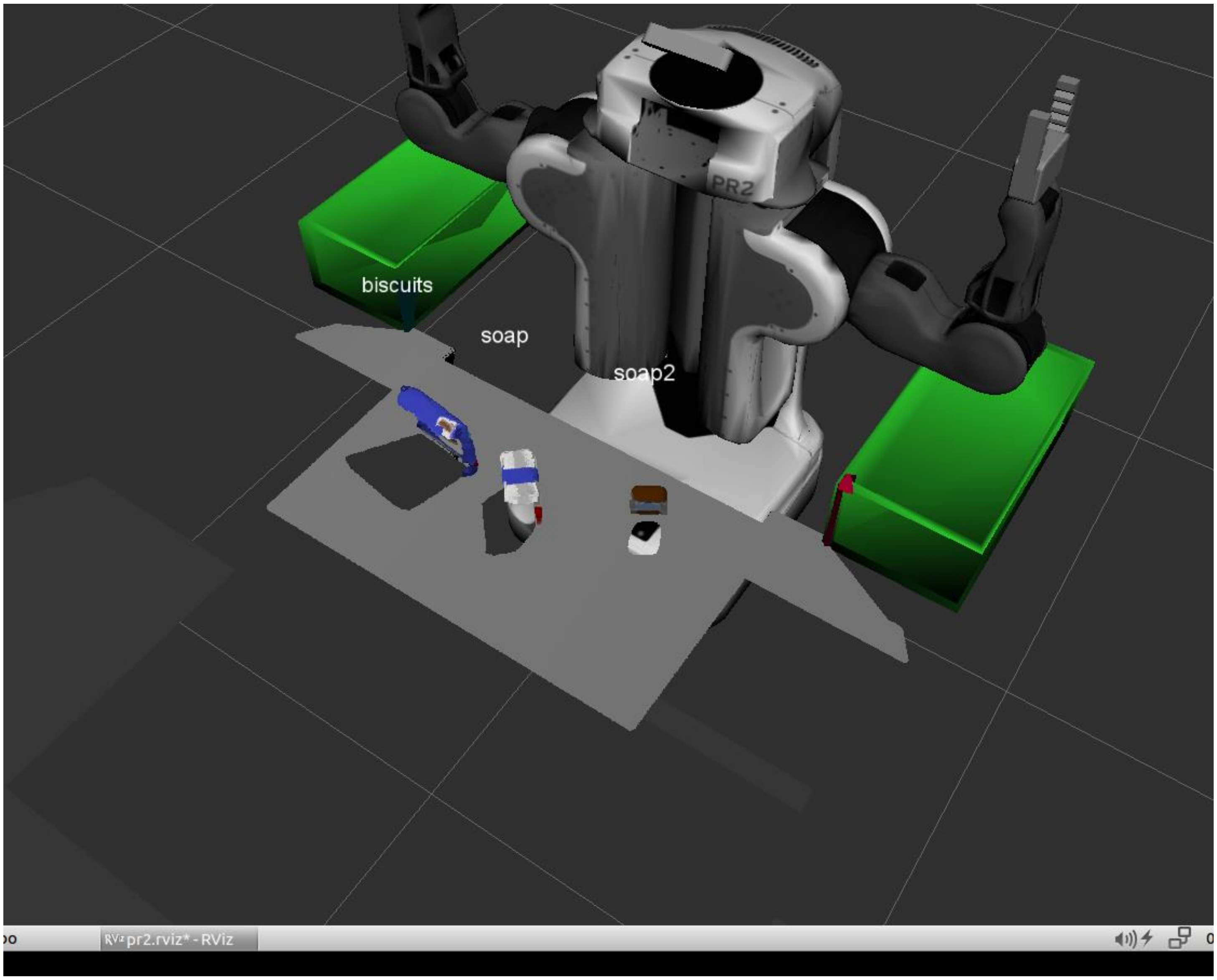
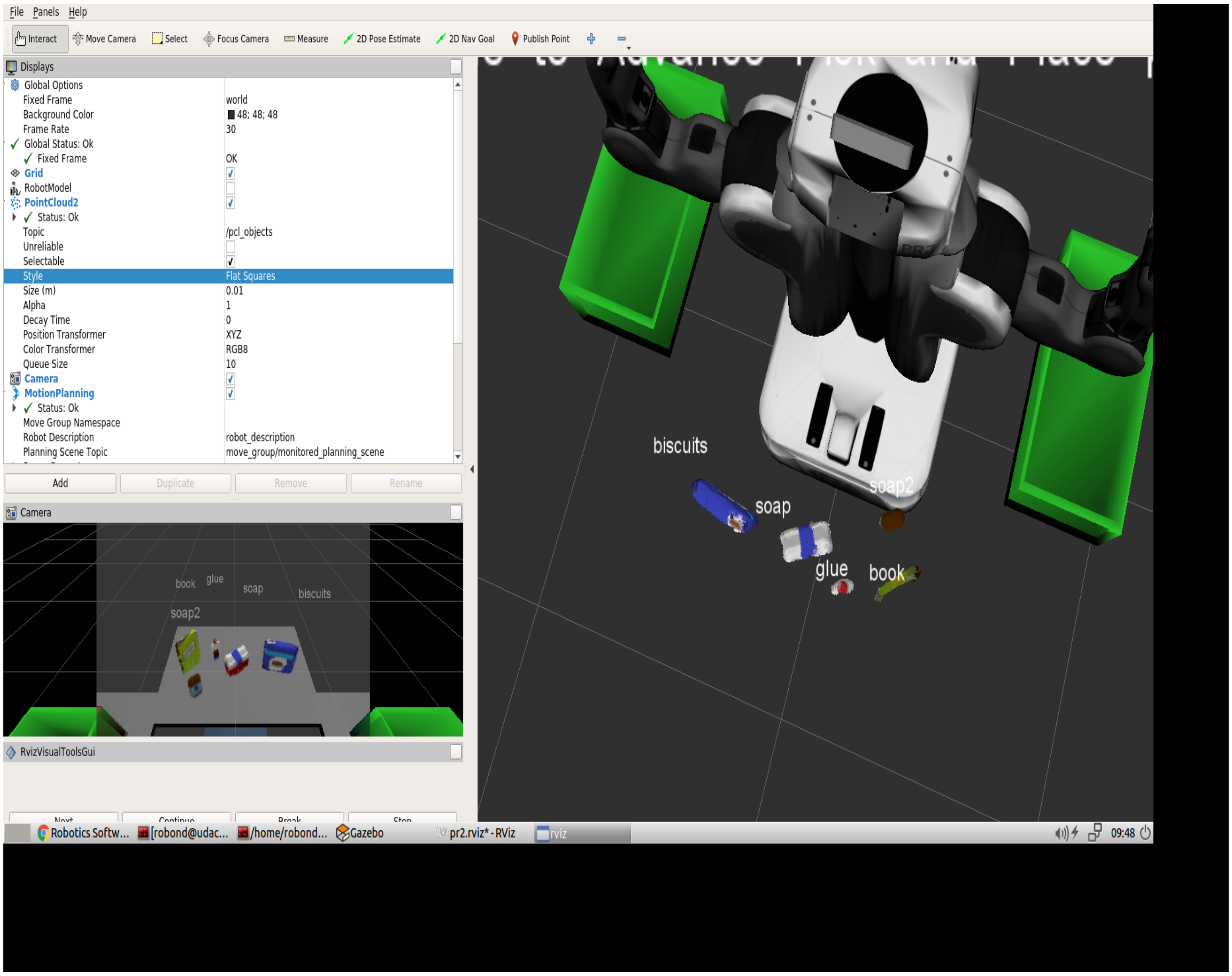
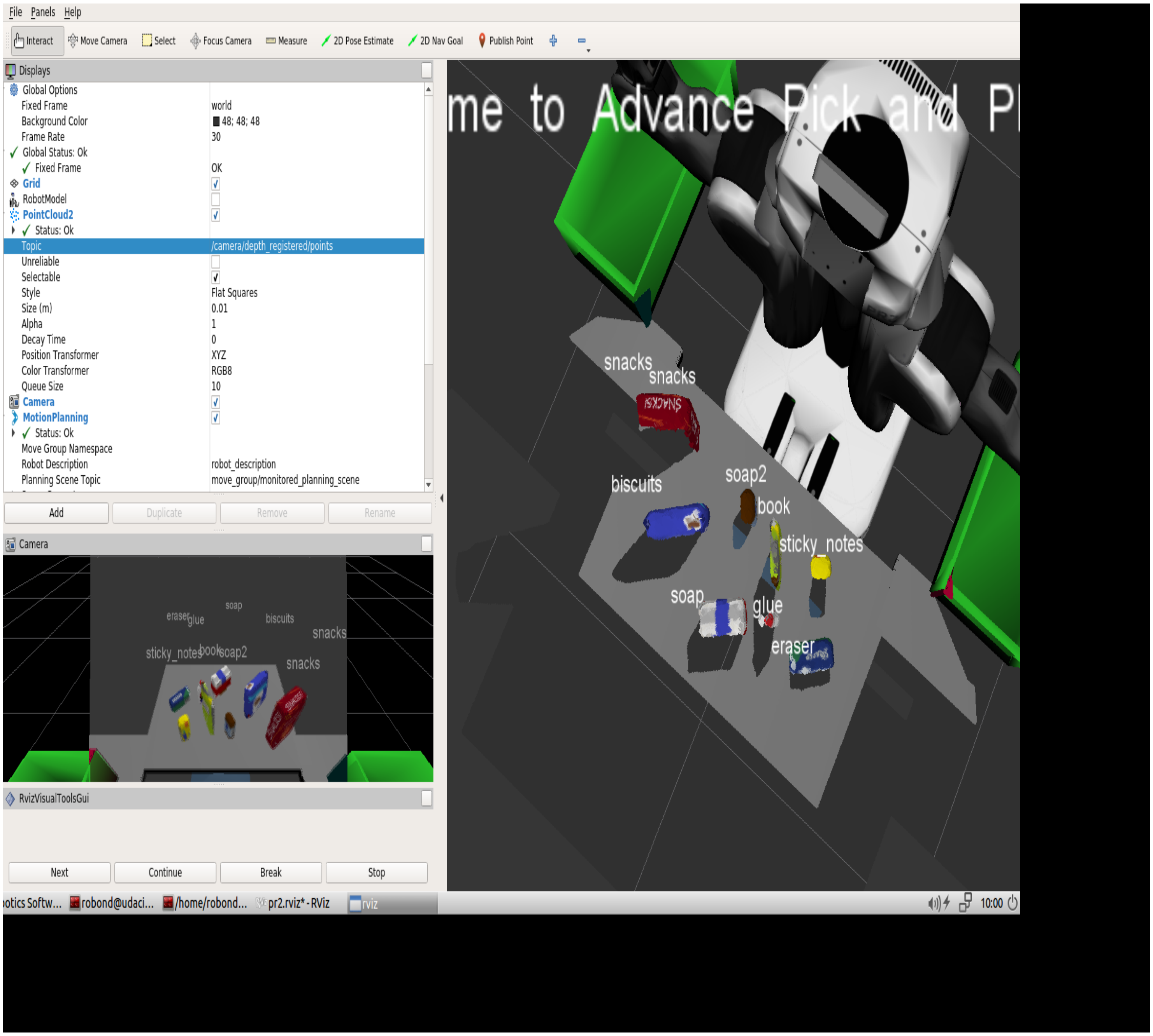


Fig. World 1 Object detection



Global Options	
Fixed Frame	world
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	
PointCloud2	✓
✓ Status: Ok	
Topic	/pcl_objects
Unreliable	
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
Queue Size	10
Camera	✓
MotionPlanning	✓
✓ Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitored_planning_scene

Fig. World 3 Object Detection


File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
Camera
RvizVisualToolsGui
Next Continue Break Stop
me to Advance Pick and Pl
snacks
snacks
biscuits
soap2
book
sticky_notes
soap
glue
eraser
10:00

Global Options	
Fixed Frame	world
Background Color	48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	
PointCloud2	✓
✓ Status: Ok	
Topic	/camera/depth_registered/points
Unreliable	
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
Queue Size	10
Camera	✓
MotionPlanning	✓
✓ Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitored_planning_scene

Fig. World 3 Object Detection