RoboND Perception Project

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

You're reading it!

Exercise 1, 2 and 3 pipeline implemented

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

I first converted the incoming ROS cloud to PCL format so that the useful functions in the PCL library can be used. The first of these the statistical outlier filter which is used to filter out the noise from the camera and leave a (relatively) clean image. I found the image fairly noisy so I had to reduce the std_dev parameter a lot to get rid of most of the noise. In the end I settled on a value of 0.3 - I was concerned to go much lower than this, even though a little noise remained, in case it ended up removing actual pieces of the real data.

Next I made a voxel grid filter to downsample the image. This reduces the amount of memory and processing power required to operate on the pcl cloud. During the earlier exercises I settled on a LEAF_SIZE of 0.005 which seemed to give a good balance between reducing computational requirements without removing too much detail.

Next I added a pass-through filter in the z-dimension to focus on the area of interest and remove all other parts of the PCL cloud. I set the min and max parameters to 0.6 and 1.3, again as determined in the earlier exercises. The min parameter was set slightly higher than that suggested in the lesson notes in order to additionally remove the table edge from the point cloud. I also added an additional pass-through filter in the y-dimension with min and max values of -0.5 and 0.5, respectively, to remove the two drop-boxes from the point cloud. Before I did this they were being included as part of the clusters, which is not intended.

Finally for the initial processing I added a RANSAC plane segmenter to identify the table and hence allow the separation of the cloud into two, one for just the table cloud and one containing all the objects (which is what we are after). I found setting max_distance of 0.01 worked well for the RANSAC segmenter.

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

The next section involved taking the objects' cloud created in step 1 and clustering it into the different objects present. First the colour information was removed (create a white cloud) and then Euclidean Clustering was performed. I set the ClusterTolerance = 0.015, MinClusterSize = 20 and MaxClusterSize = 3000. These parameters had to be adjusted a little from the settings I used in the original exercise since some of the new objects are larger (e.g. the biscuits).

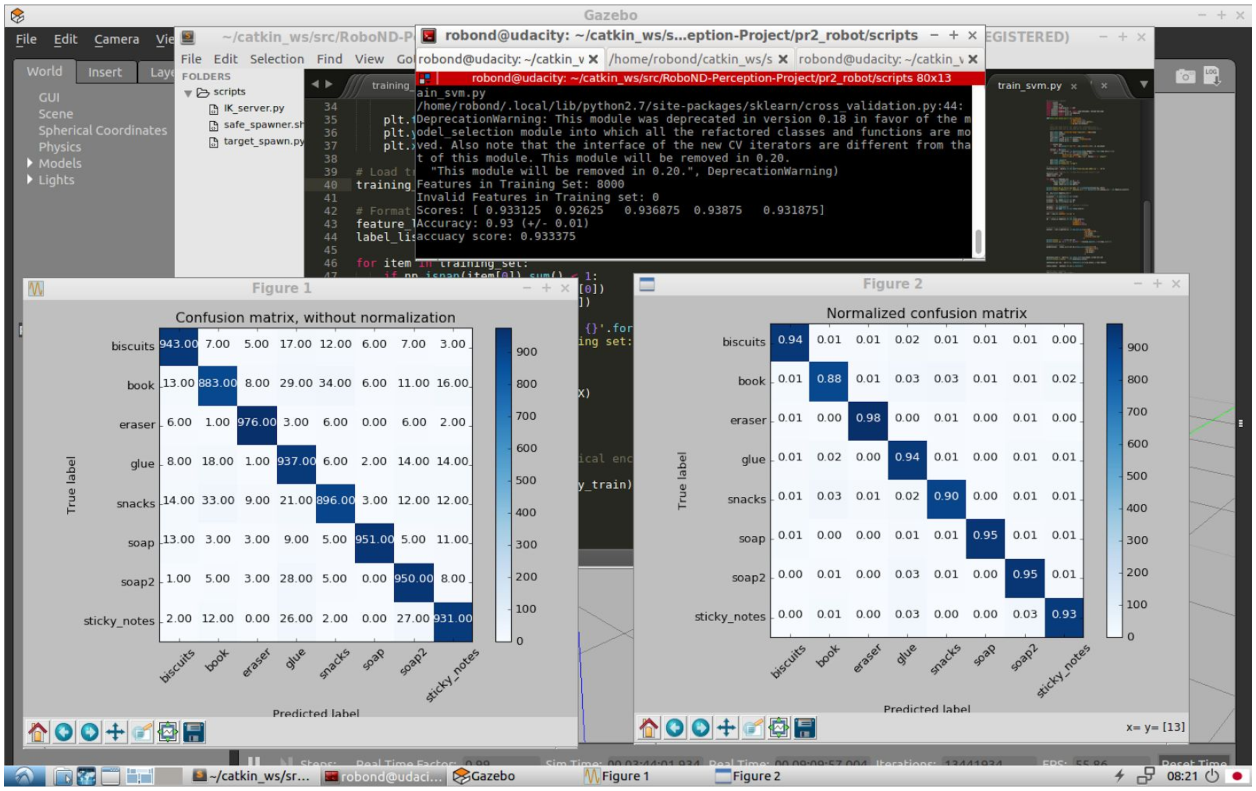
I then apply a different colour to each detected cluster, as shown in the lessons. Finally the PCL clouds are converted back to ROS format and then published to various topics so they can be visualised in RViz.

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

I implemented the compute_color_histograms and compute_normal_histograms functions as discussed in the lesson notes. I set a range of [0,256] for the colours and [-1,1] for the normals. As for bins, in the end I used 32 for each. I tried higher numbers such as 64 and that gave better accuracy according to the train_svm script, but seemed to perform poorly when finally used in RViz. I think they were overfitting, so I changed back to 32 bins.

To train the model I adapted the capture_features script from Exercise 3 so it would run with the additional models. I started off with just a few iterations for each model (i = 10) but the accuracy was not that great (~50%). In the end I used an i = 1000 and this extra data led to a massive jump in accuracy! As shown in Figure 1 the final accuracy of my model is 0.93, which I think is fairly respectable. If I let the training sequence run for even more iterations (i.e. for a couple of days) I think it is possible to improve even further, but it is probably not necessary for this project at this time.

Finally I used the model inside the perception pipeline to identify the the individual object clusters that were detected in step 2. I then publish a label into RViz with their name to check that it is working correctly.



```
ain_svm.py
/home/robond/.local/lib/python2.7/site-packages/sklearn/cross_validation.py:44:
DeprecationWarning: This module was deprecated in version 0.18 in favor of the m
odel_selection module into which all the refactored classes and functions are mo
ved. Also note that the interface of the new CV iterators are different from tha
t of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
Features in Training Set: 8000
Invalid Features in Training set: 0
Scores: [ 0.933125  0.92625   0.936875  0.93875   0.931875]
Accuracy: 0.93 (+/- 0.01)
accuacy score: 0.933375
```

Confusion matrix, without normalization

True label \ Predicted label	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	943.00	7.00	5.00	17.00	12.00	6.00	7.00	3.00
book	13.00	883.00	8.00	29.00	34.00	6.00	11.00	16.00
eraser	6.00	1.00	976.00	3.00	6.00	0.00	6.00	2.00
glue	8.00	18.00	1.00	937.00	6.00	2.00	14.00	14.00
snacks	14.00	33.00	9.00	21.00	896.00	3.00	12.00	12.00
soap	13.00	3.00	3.00	9.00	5.00	951.00	5.00	11.00
soap2	1.00	5.00	3.00	28.00	5.00	0.00	950.00	8.00
sticky_notes	2.00	12.00	0.00	26.00	2.00	0.00	27.00	931.00

Normalized confusion matrix

True label \ Predicted label	biscuits	book	eraser	glue	snacks	soap	soap2	sticky_notes
biscuits	0.94	0.01	0.01	0.02	0.01	0.01	0.01	0.00
book	0.01	0.88	0.01	0.03	0.03	0.01	0.01	0.02
eraser	0.01	0.00	0.98	0.00	0.01	0.00	0.01	0.00
glue	0.01	0.02	0.00	0.94	0.01	0.00	0.01	0.01
snacks	0.01	0.03	0.01	0.02	0.90	0.00	0.01	0.01
soap	0.01	0.00	0.00	0.01	0.01	0.95	0.01	0.01
soap2	0.00	0.01	0.00	0.03	0.01	0.00	0.95	0.01
sticky_notes	0.00	0.01	0.00	0.03	0.00	0.00	0.03	0.93

Figure 1 - Confusion matrix output of train_svm script.

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 PickPlace request output them to .yaml format.

After all the objects were detected and classified in steps 2 and 3 I then call the pr2_mover() function to create the output .yaml files. First the code reads in the object list parameters .yaml file and parses it to find out which objects should be collected in this particular case. Similarly the dropbox yaml file is also loaded and parsed to find out where the objects should ultimately be placed.

Then comes the main loop which iterates over all the items that need to be collected (as determined from the pick list). It first sets the object name based on the .yaml data and converts it into the correct datatype to be understood by ROS.

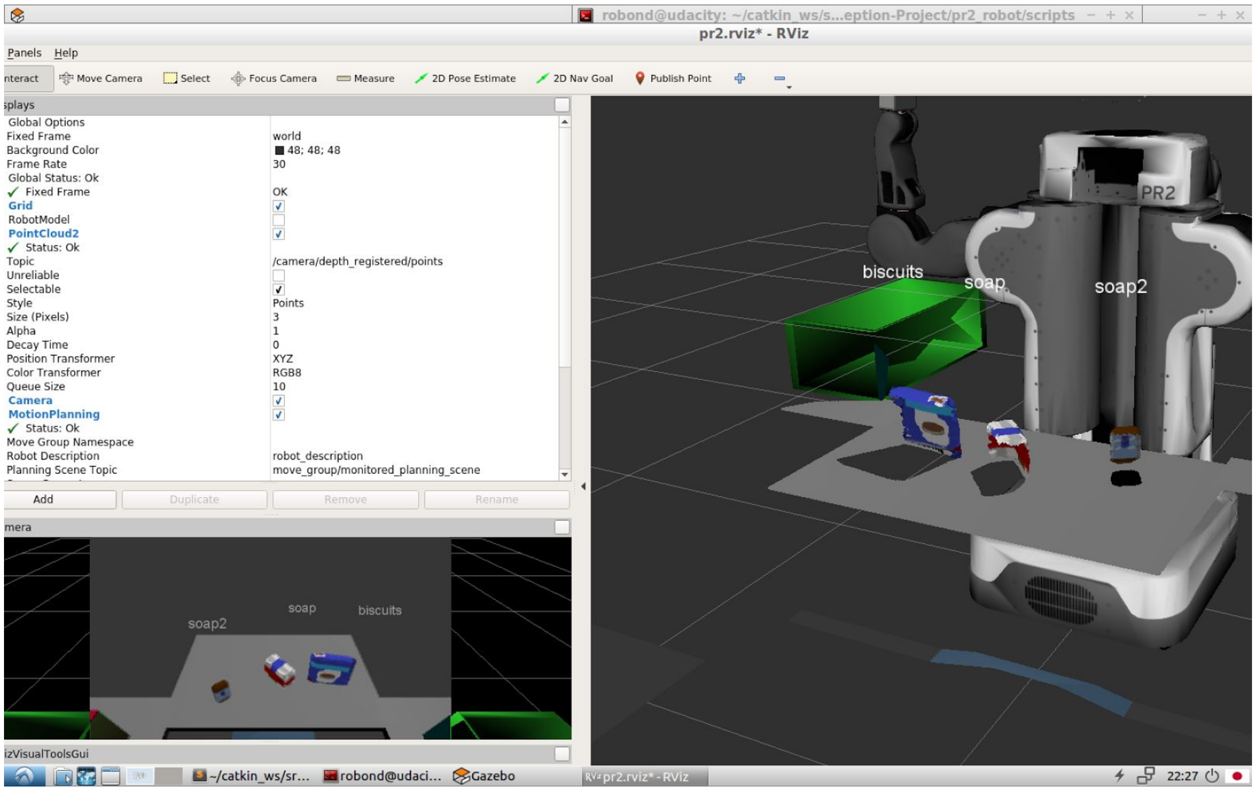
The next step in the loop is to determine the pick_pose. At first all values are set to 0. In our case the final orientation is always 0, so this is OK. As for position, it shouldn't be 0, but this gives a default value in case the object was not detected by the classifier, to avoid an error being created when trying to create the yaml file. The code then checks through the objects that had been detected to see if the correct one does exist, and if so it overrides the pick_pose.position with the correct value as calculated by the centroid of the object, using the method discussed in the lesson notes.

Next the code checks whether the group specified was either red or green, and then sets the correct arm_name (left or right, respectively).

Finally the test_scene_number is specified and the dictionary is created. This is then used to make the final output yaml file using the helper functions provided.

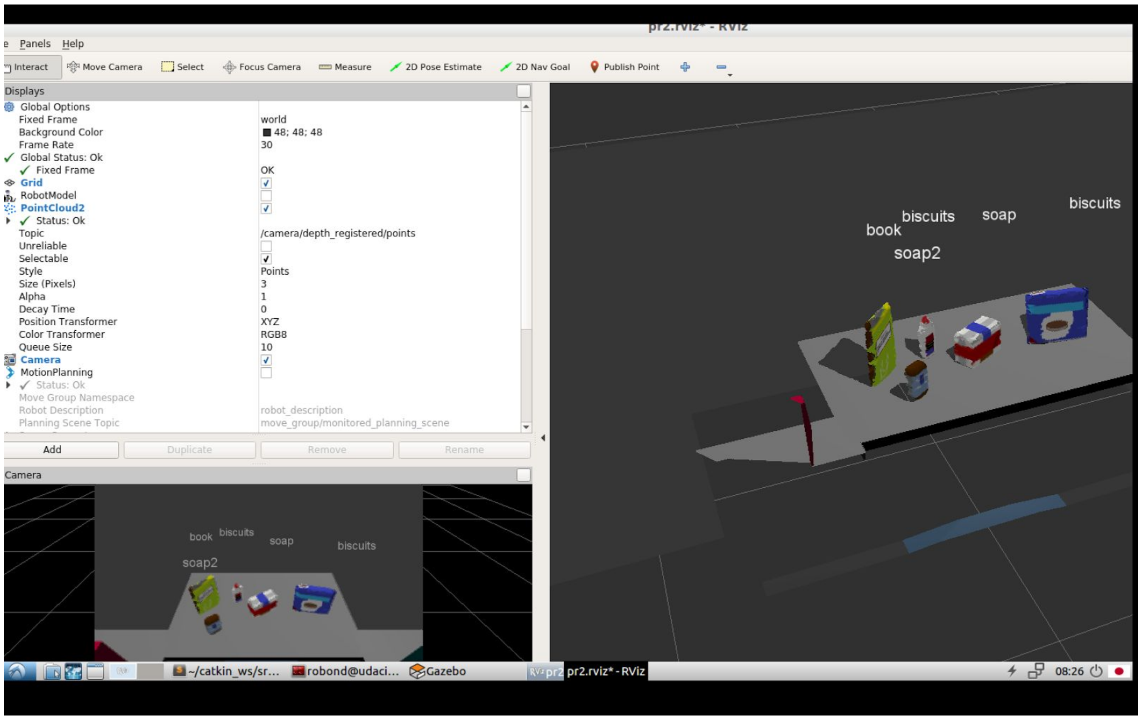
Results

The output yaml files can be found in the Github repo. Figures 2, 3 and 4 show the output of the classifier in RViz for worlds 1, 2 and 3 respectively. In world 1 100% (3/3) of objects are classified correctly. In world 2 80% (4/5) of objects are classified correctly. The one error was that the glue was misclassified as biscuits. In world 3 87.5% (7/8) of objects are classified correctly. The one error is the same as in world 2, with glue being misclassified as biscuits.



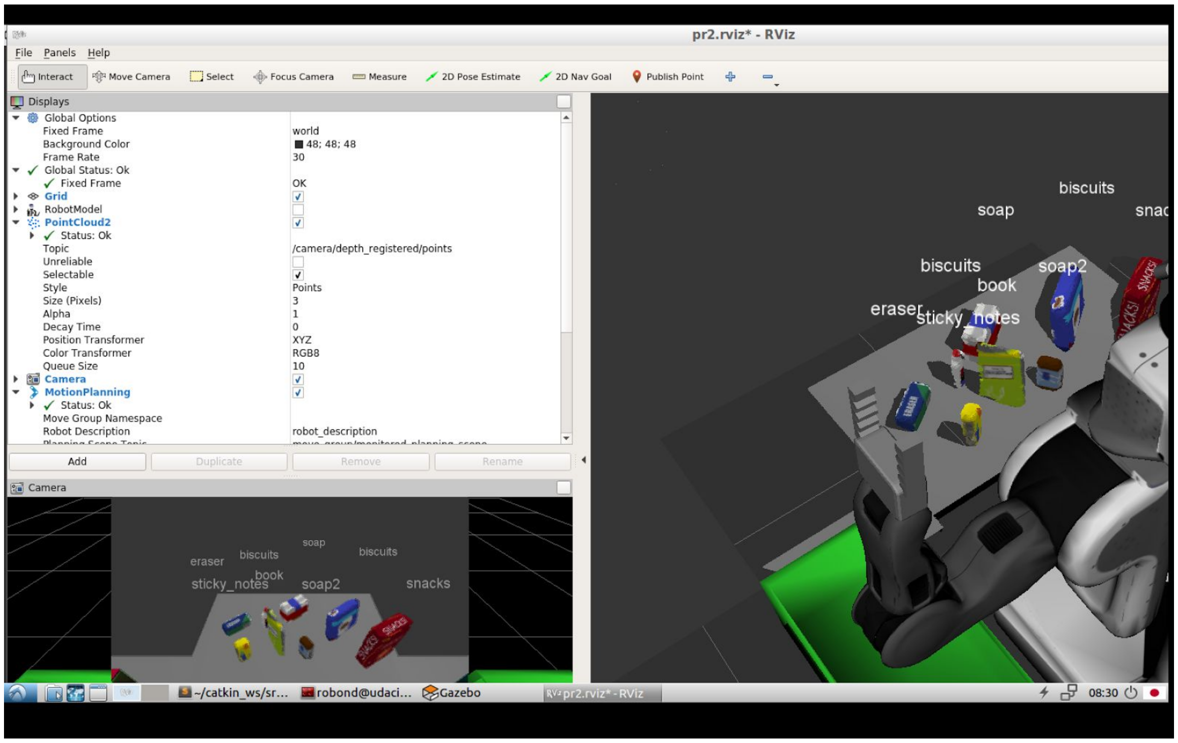
```
Global Options
Fixed Frame                  world
Background Color             ■ 48; 48; 48
Frame Rate                   30
Global Status: Ok
✓ Fixed Frame                OK
Grid                         ✓
RobotModel
PointCloud2                  ✓
✓ Status: Ok
Topic                        /camera/depth_registered/points
Unreliable
Selectable                   ✓
Style                        Points
Size (Pixels)                3
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
```

Figure 2 - Screenshot of object classification run in world 1



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	Points
Size (Pixels)	3
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

Figure 3 - Screenshot of object classification run in world 2



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	Points
Size (Pixels)	3
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

Figure 4 - Screenshot of object classification run in world 3

Conclusion

So overall my classifier performed well enough to meet the requirements of the project. However it both worlds 2 and 3 it misclassified glue as biscuits, a potentially fatal error if this robot was to work in a cafe! Looking at the normalised confusion matrix there is little clue that such an error could be expected. Glue is classified correctly 91% of the time and only mistaken for biscuits 1% of the time. Being mistaken for soap2 would be even more likely at 3% of the time. In face I did notice occasionally in world 2 that the glue was classified correctly, occasionally changing to glue then shortly returning back to biscuits. So this error is probably marginal as it seems the noise can affect it. So one potential solution to improve my pipeline even further would be to turn the noise filter on even more aggressively to try and get rid of more of it. Another potential solution would be to run even more training iterations to try and build up an even better model of each object to get greater prediction accuracy. I could also try tuning the SVM parameters further, for example by using a Grid Search.

Lastly, I plan to continue with the optional steps of this project later, to try and complete the whole perception and pick & place operation - wish me luck!