Udacity Nanodegree Perception Project Write Up

Dennis Forward

Exercise 1 - Segmentation

In this exercise I filled out the RANSAC.py file to filer the point cloud of the table, objects and background.

1. Voxel Downsampling: The first step in segmentation for the point cloud that the robot's sensor sees is Voxel Downsampling. This step reduces the amount of data in the point cloud by creating a voxel size and taking the average position of the points in each voxel, and combining them into one point. In this step I used a leaf size of 0.01 to still get a detailed point cloud of the scene, while keeping the computing time tlow.
2. Pass Through Filtering: The second step was Pass Through Filtering. A pass through minimum of 0.6 and maximum of 1.1 were used to remove all data from the point cloud, except the table top, and the objects on it.
3. RANSAC Segmentation: The third step in the segmentation process was to remove the table from the scene using the RANSAC algorithm. We used RANSAC plane fitting, then used the inliers from that algorithm to find the table top, and then the extracted outliers to reduce the point cloud to only the objects on the top of the table.
4. Outlier Removal: The last step in segmentation would be to remove any outliers using PCLs StatisticalOutlierRemoval filter, or something similar. However, since this project had simulated data, there was no outliers, so this was not necessary.

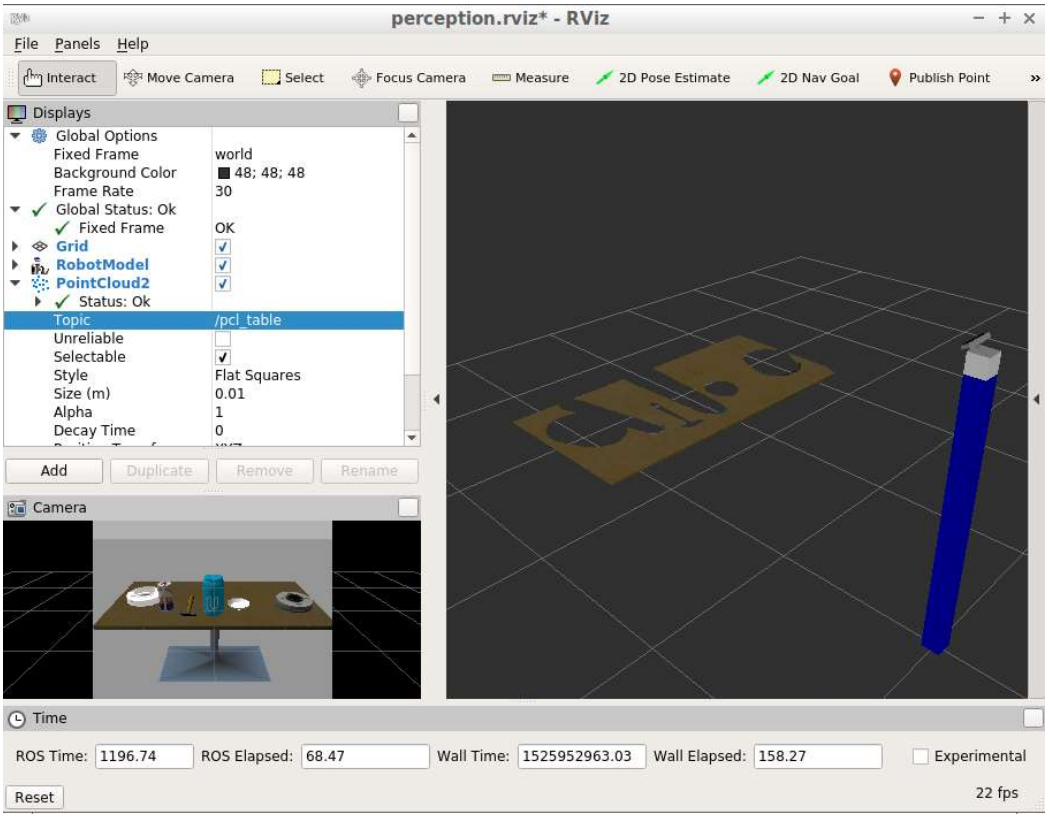
Exercise 2 -

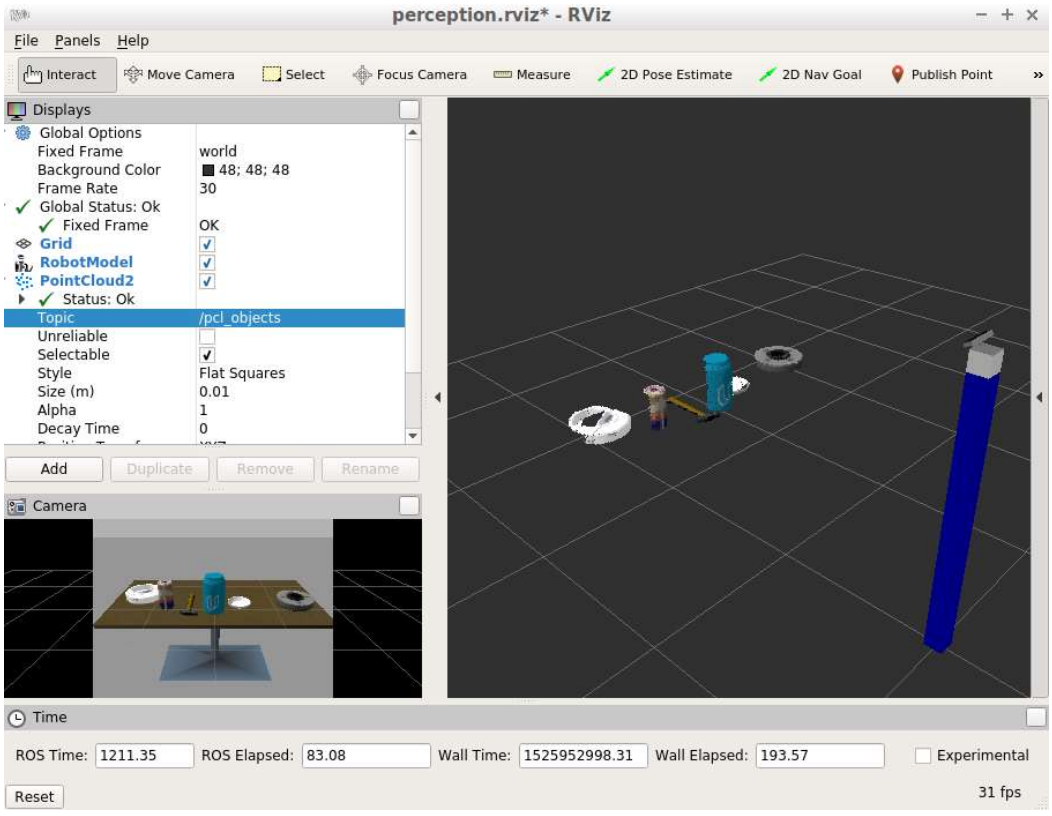
In this exercise we created a ROS Node that took in a point cloud, filtered it, then segmented it into individual objects

I started with the template.py file, changed it's name to segmentation.py, and then filled in all the TODOs.

1. The first step was to initialize the ROS node.
2. The second step was to create a subscriber for the point cloud
3. Third step was to create publishers. We created one for the table point cloud and one for the objects point cloud
4. We then added code to "spin" the code while the node is running. This makes the code loop continuously until we shut down the ROS node
5. We then added the code to actually publish the objects and table publishers that we created in step 2
6. Step 6 was to convert the ROS point cloud that's being sent into our pcl_callback() function into a pcl point cloud. We used the ros_to_pcl function() function in pcl_helper.py to do this
7. We then downsampled our point cloud using a Voxel Grid filter with a leaf size of 0.01.
8. After that we applied a pass through filter with a min of 0.77m and a max of 1.1m (this seemed to separate the table and objects well)
9. Step 9 was to perform RANSAC plane fitting to identify the table, and then from that separated the inliers and outliers to get two different point clouds for the table and objects
10. After separating the object and table point clouds we converted the point clouds back to ROS message format using the pcl_to_ros() function, and published those ROS messages

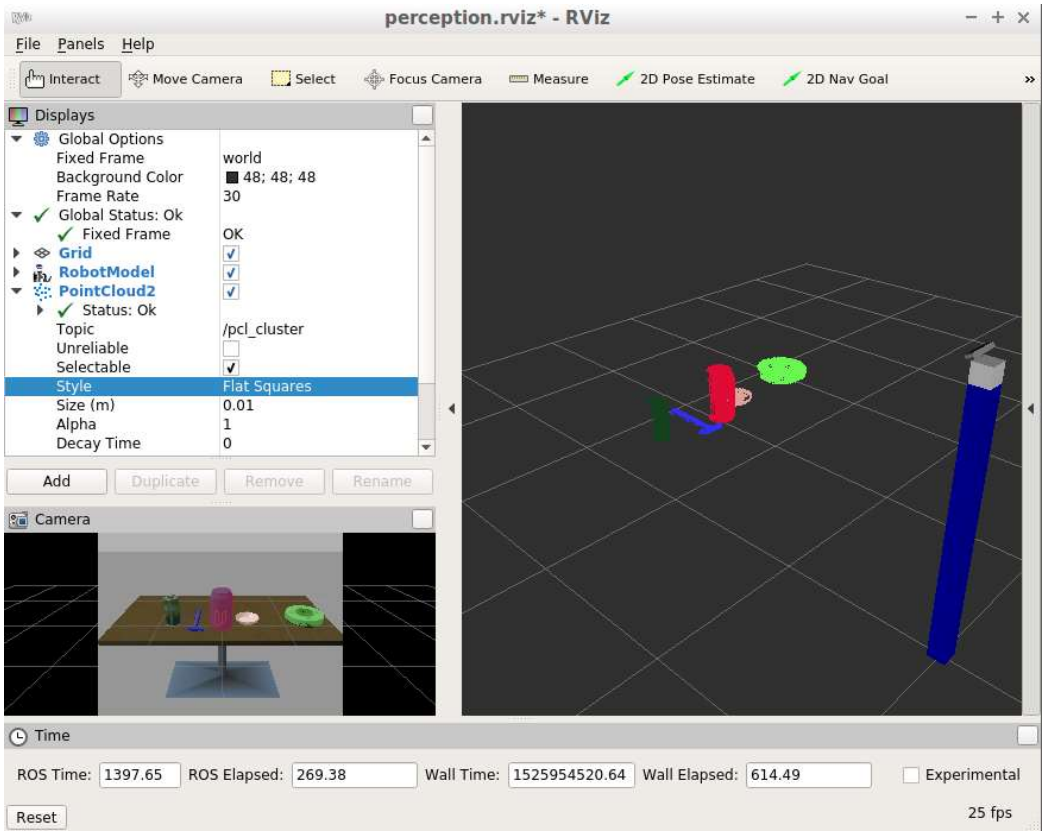
After adding these steps in the code of my segmentation.py file and running it I successfully separated the objects and table.





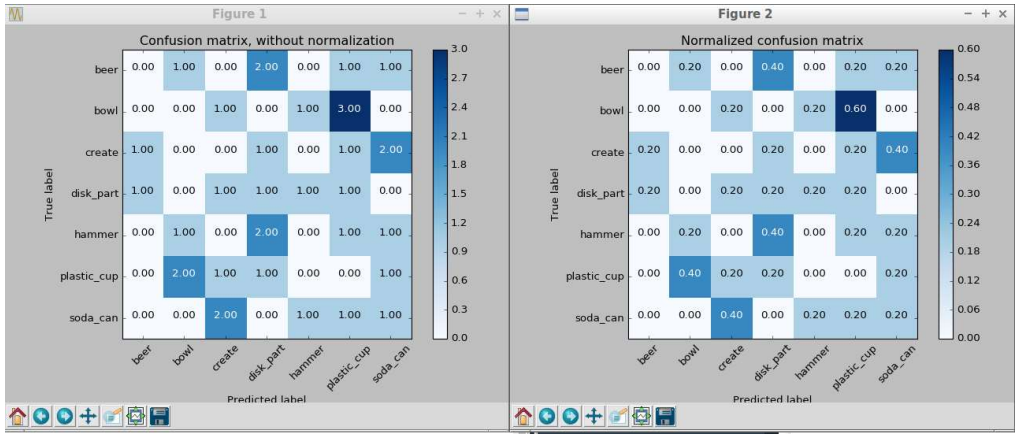
11. The next step was to use Euclidean clustering to segment the points in the cloud into objects. We converted our point cloud from XYZRGB into XYZ using the XYZRGB_to_XYZ() function, the constructed a k-d tree
12. We then performed the clustering using EuclideanClusterExtraction(). A cluster tolerance of 0.025, min cluster size of 25, and max cluster size of 2500 seemed to work well.
13. We then created the final point cloud with each object a different color, and saved it to cluster_cloud.

14. The final step was to create a new publisher and publish the colored cluster_cloud to is. After going through these steps my objects were successfully recognized.



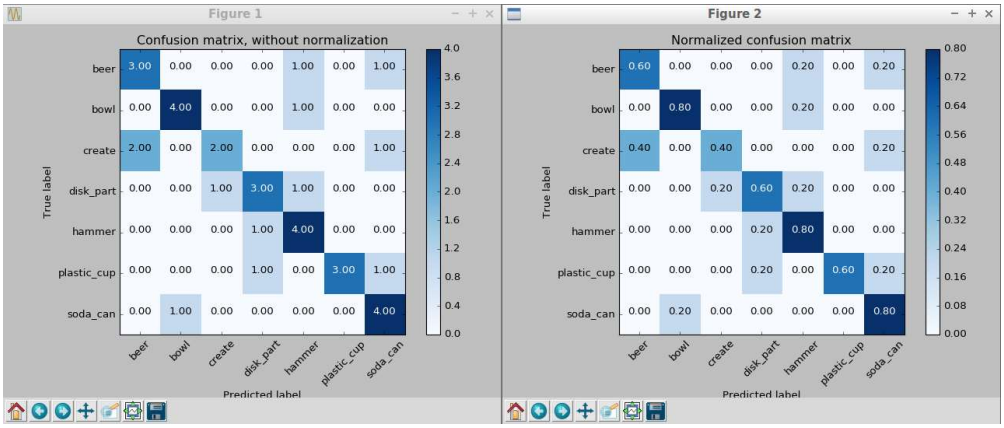
Exercise 3

1. The first step in exercise 3 was to run the sensor stick training.launch and then capture_features.py to save the features for each of the objects in random orientations. The features and labels were saved to the training_set.sav file.
2. The next step was to train our model by running thee train_svm.py file, which gave the following confusion matrices, and saved our model to model.sav

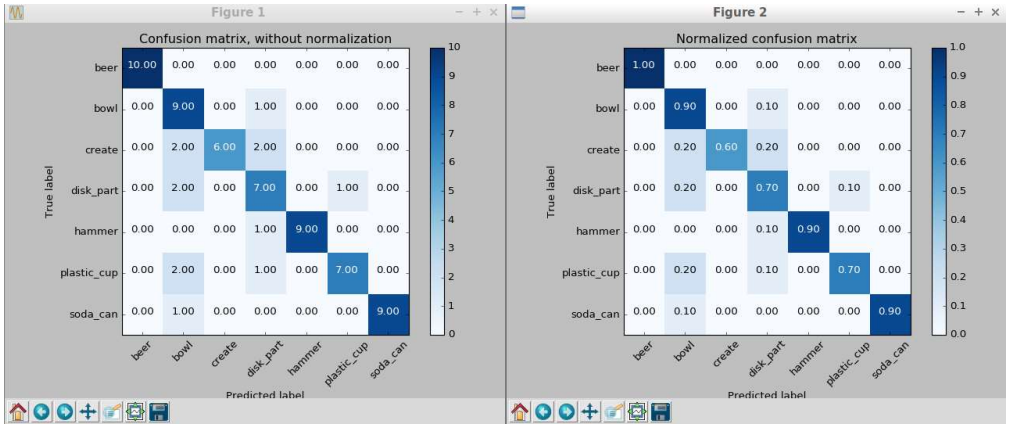


3. We then realize that the reason our confusion matrices look like they do in step 2 is because our features.py file doesn't have the compute_color_histograms() and compute_normal_histograms() functions filled in completely, and they are just returning random numbers. I filled in these two functions by creating bins and

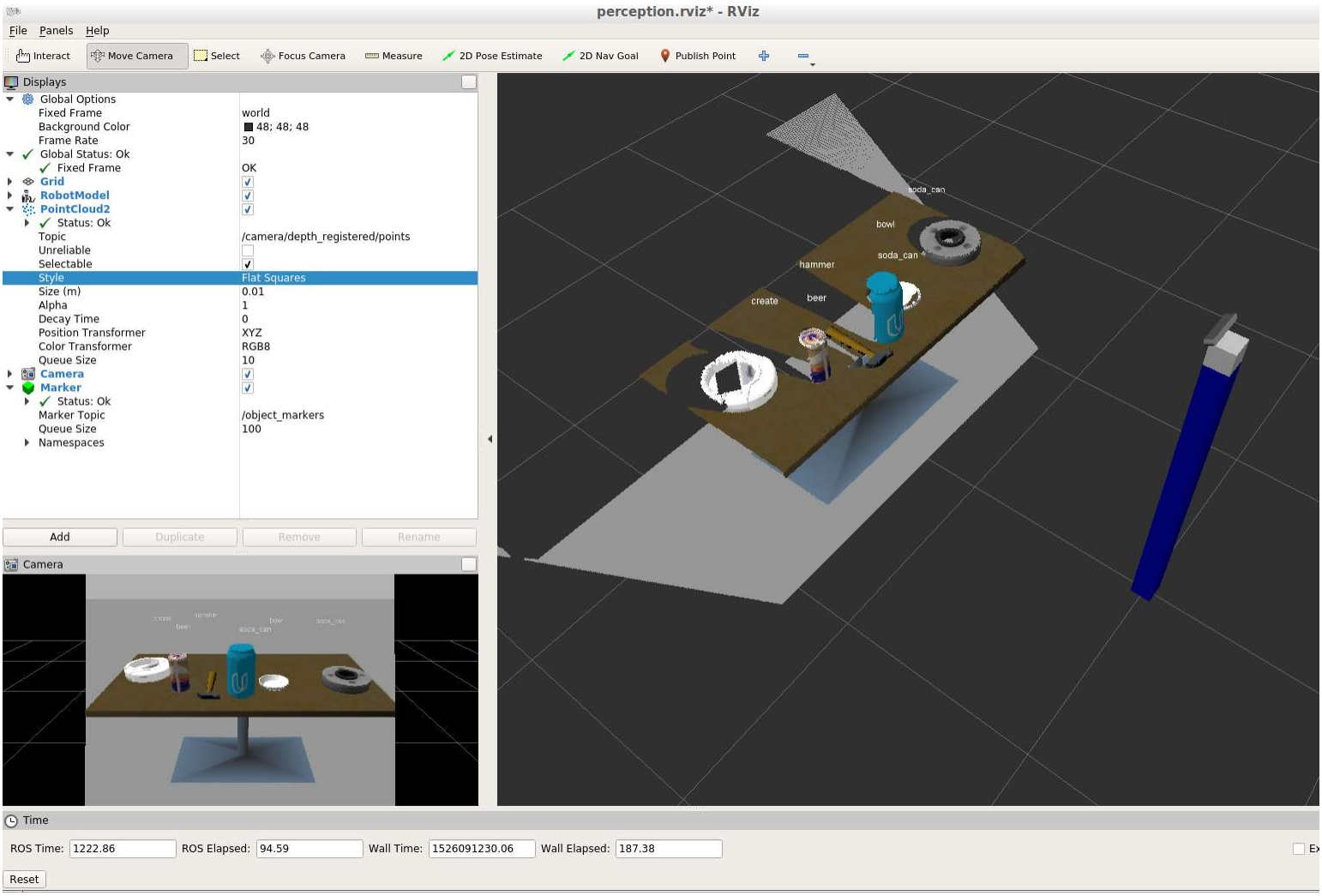
then concatenating them, and normalize them. After doing this the confusion matrices looked like this:



4. To improve on this, in the capture_features.py file I increased the number of times each object spawns randomly from 5 to 10, turned the using_hsv flag to true, and then retrained the SVM. The updated confusion tables looked like the charts below. It was not perfect, but was an improvement.



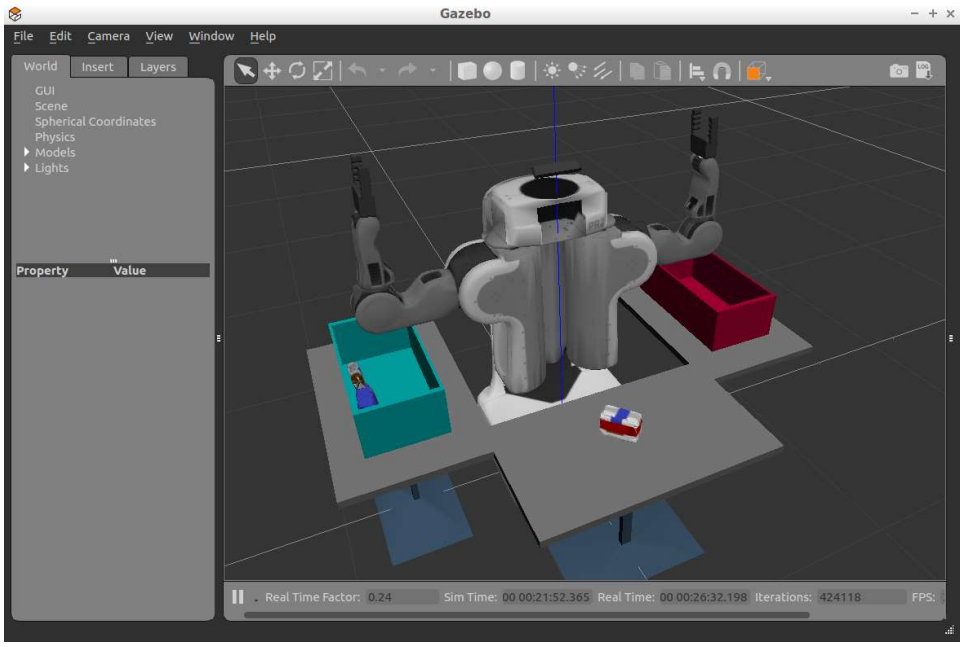
5. I then copied the template.py file and renamed it object_recognition.py as suggested in the lesson. I then added my code from the first two exercises. To the pcl_callback() function.
6. I then created lists and put in a for loop that cycles through each of the clusters and makes predictions
7. Finally, we initialized the ROS node, created subscribers and publishers and published new messages for the detected objects and the object markers. My script and SVN successfully recognized all objects except for the disk part.



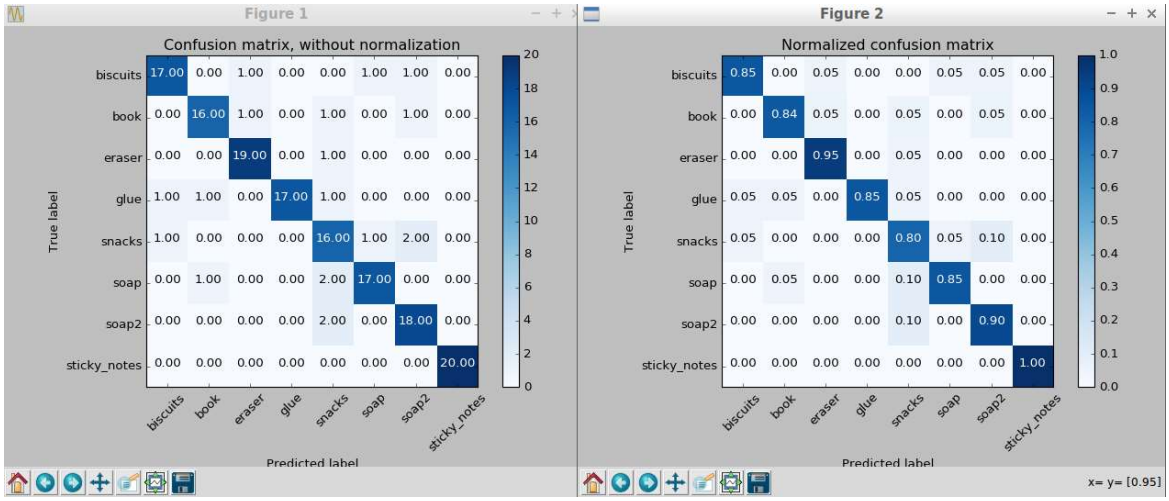
Pick and Place Project Portion

Stopped at Project_3
The Goal of this project is to have the PR2 robot correctly identify the objects on the tables in the three test worlds. The environment is a PR2 Robot, with a table in front of it containing objects, and two tables on the sides with boxes for depositing those objects.

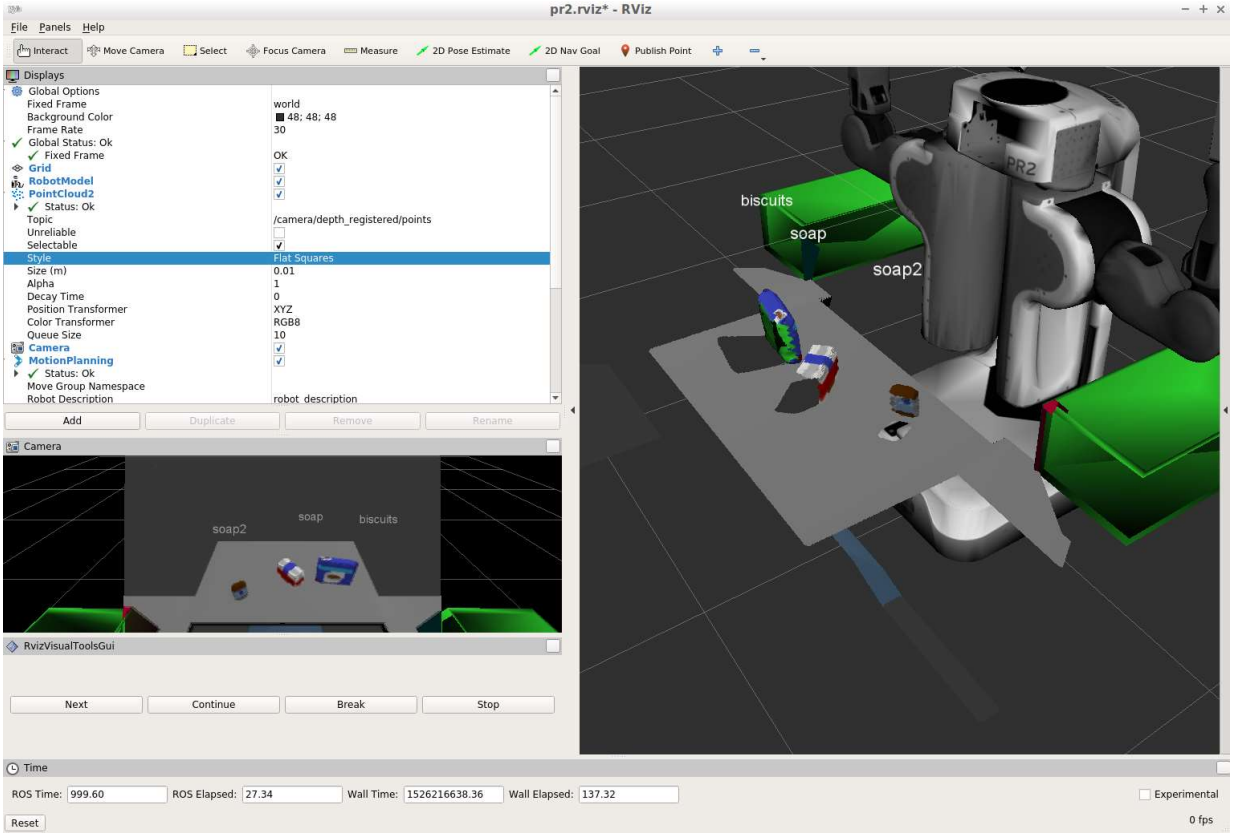
1. I first set up and ran the Demo, and the Robot deposited 2 out of the three objects into the bin. The object that was not put into the bin was recognized, but when the robot attempted to grasp it, the grippers didn't seem to be closed when the arm started moving away from the object, so it was left on the table. The Demo walkthrough notes that this might be the case during the demo run.



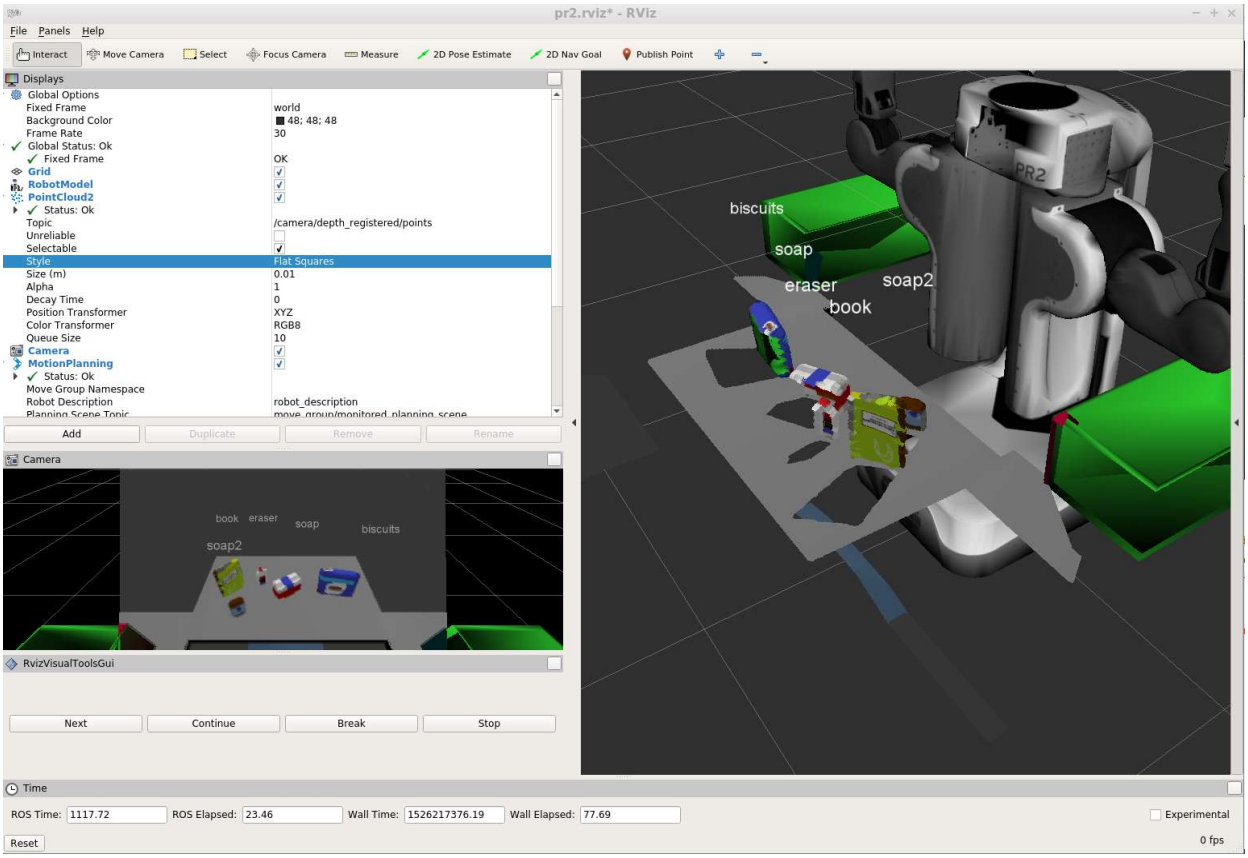
2. I then opened the project_tepmplate.py file and moved in my code from the three earlier Exercises into that file.
3. I had to generate a new set of training features for my SVM with the new objects from the exercises. To do this, I copied the list of objects from the pick_list_3.yaml file into capture_features.py and ran training.launch and capture_features.py to capture the new objects, and then finally ran my train_svm.py. The confusion matrix is below:



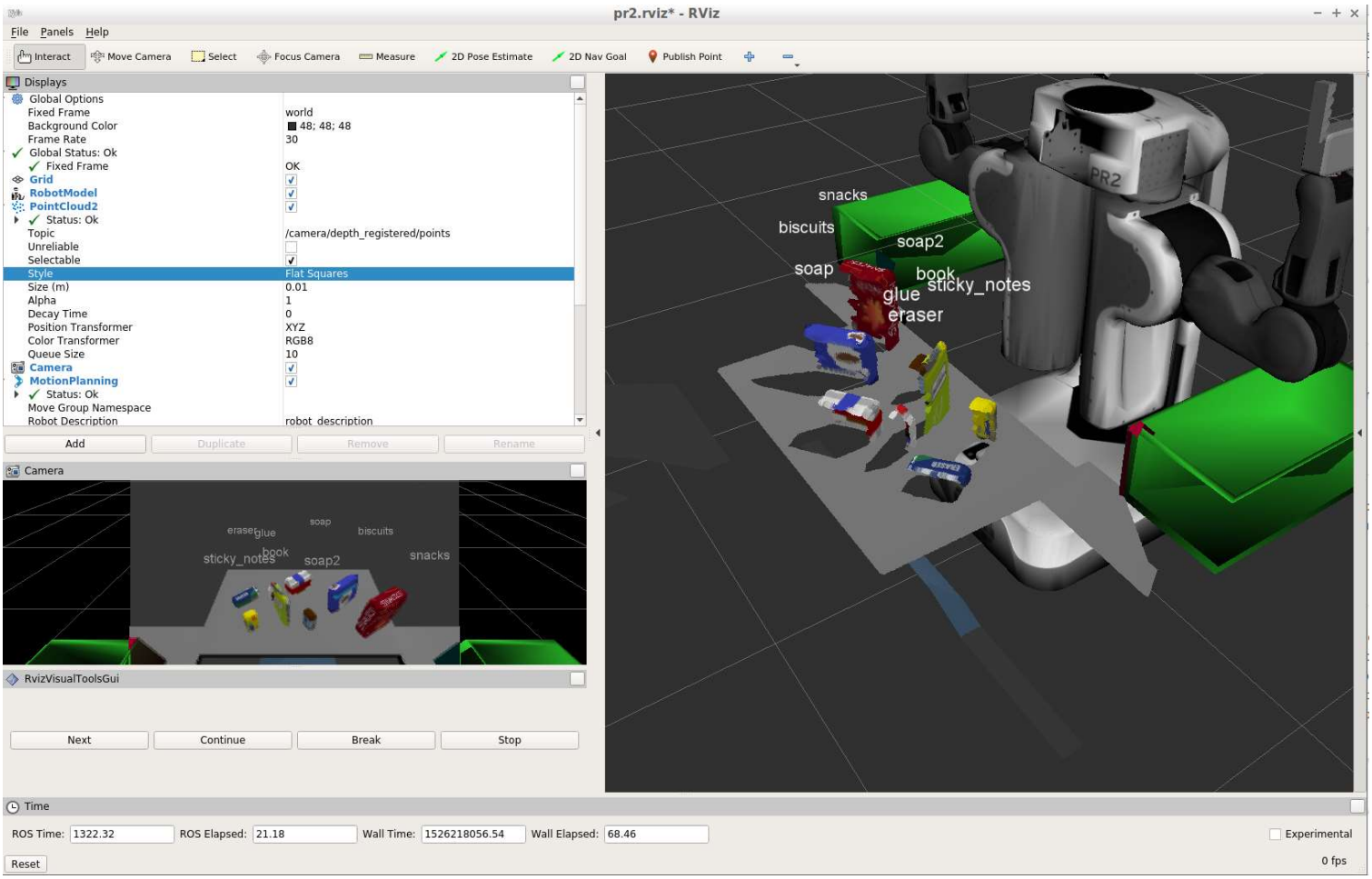
4. The first time I ran the project_template.py file with the first pick list, the robot correctly recognized all three of the objects, and recorded the centroid positions. See the output_1.yaml file for the centroid positions.



5. In test2.world my robot recognized 4 out of the 5 objects, and recorded the centroid positions It incorrectly classified the glue as the eraser. See the output_2.yaml file for the centroid positions.



6. In test3.world my robot correctly recognized all 8 out of 8 of the objects. See the output_3.yaml file for the centroid positions.



Conclusion

This project was a very good intro to perception in robotics. The exercises showed in easy to follow steps and lessons, how point clouds are gathered from RGBD cameras, and how they can be manipulated and studied to get a variety of useful information. I was able to successfully get my PR2 to recognize the vast majority of the objects that were put on the table in front of it. I will definitely be using the lessons I learned from this project in the future in my professional work.