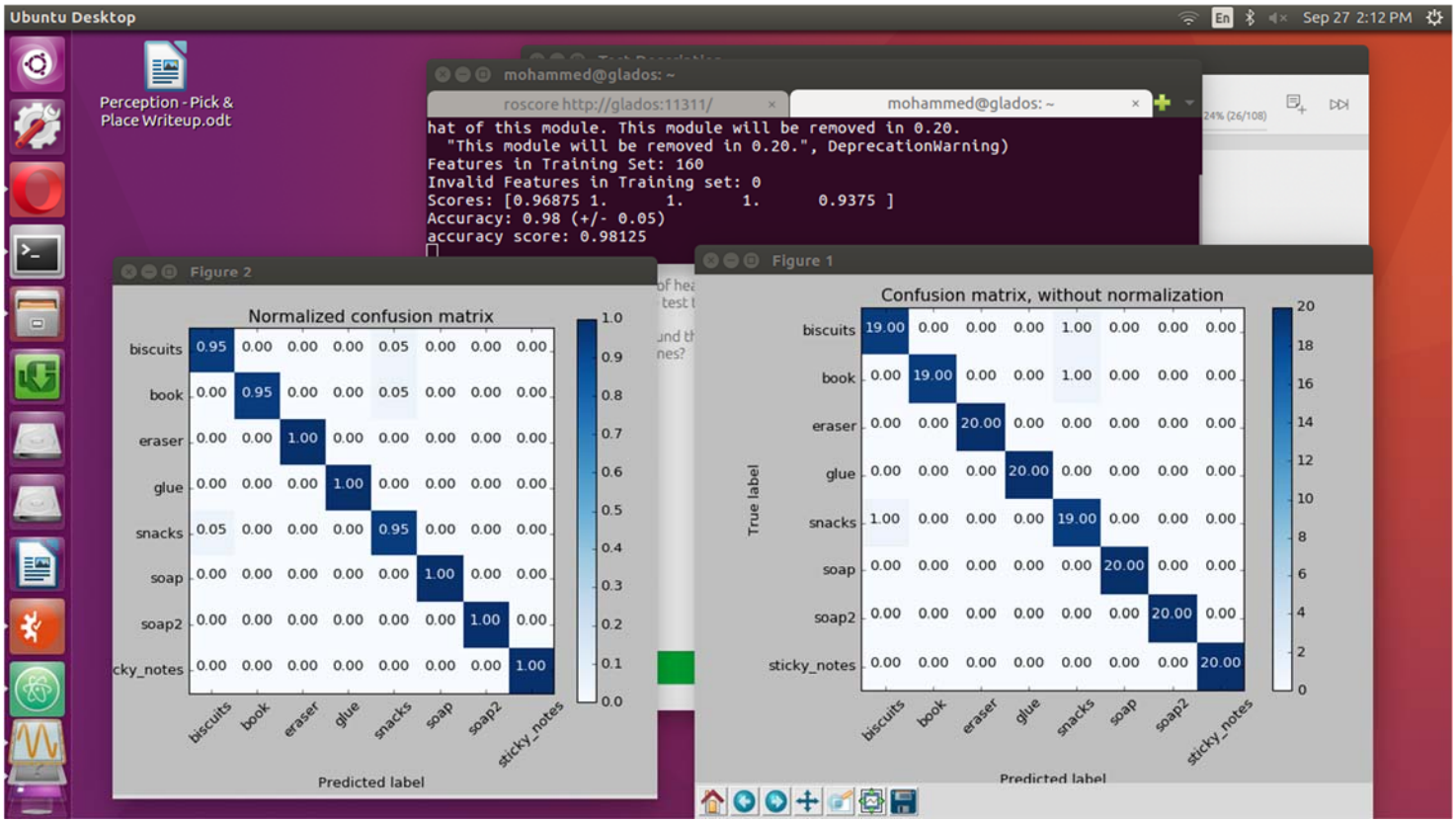
Project: Perception Pick & Place

Required Steps for a Passing Submission:

1. Extract features and train an SVM model on new objects (see `pick_list_*.yaml` in `/pr2_robot/config/` for the list of models you'll be trying to identify).

Just like the perception exercise I used the provided capture features node and then the train_svm node.

I took 20 sample kept the kernel type as linear, which gave a 98% accurate trained SVM Model



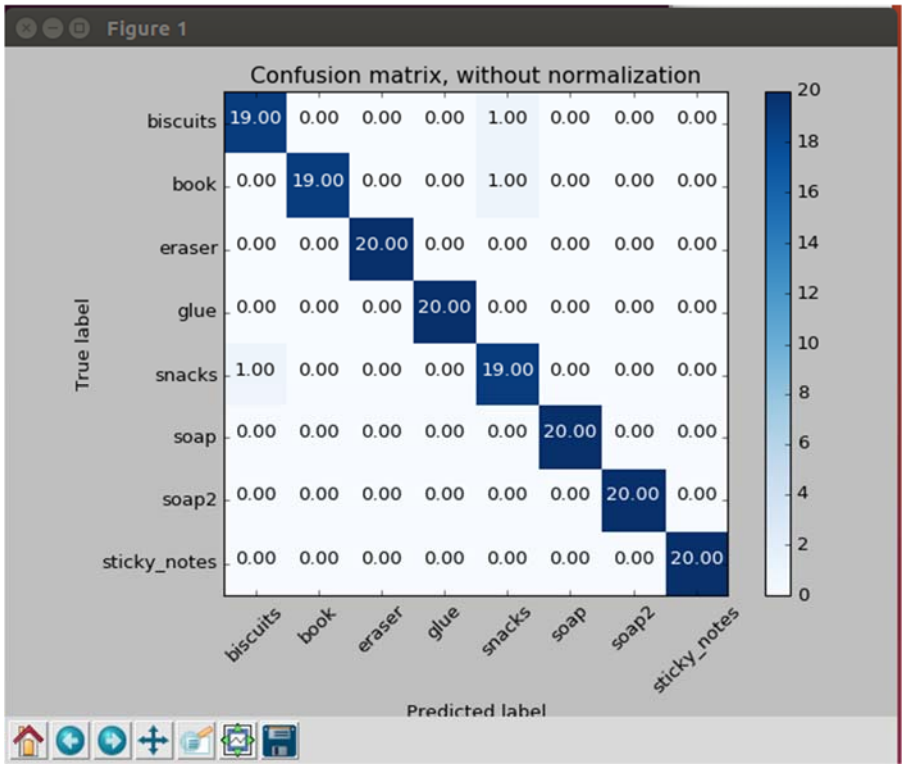
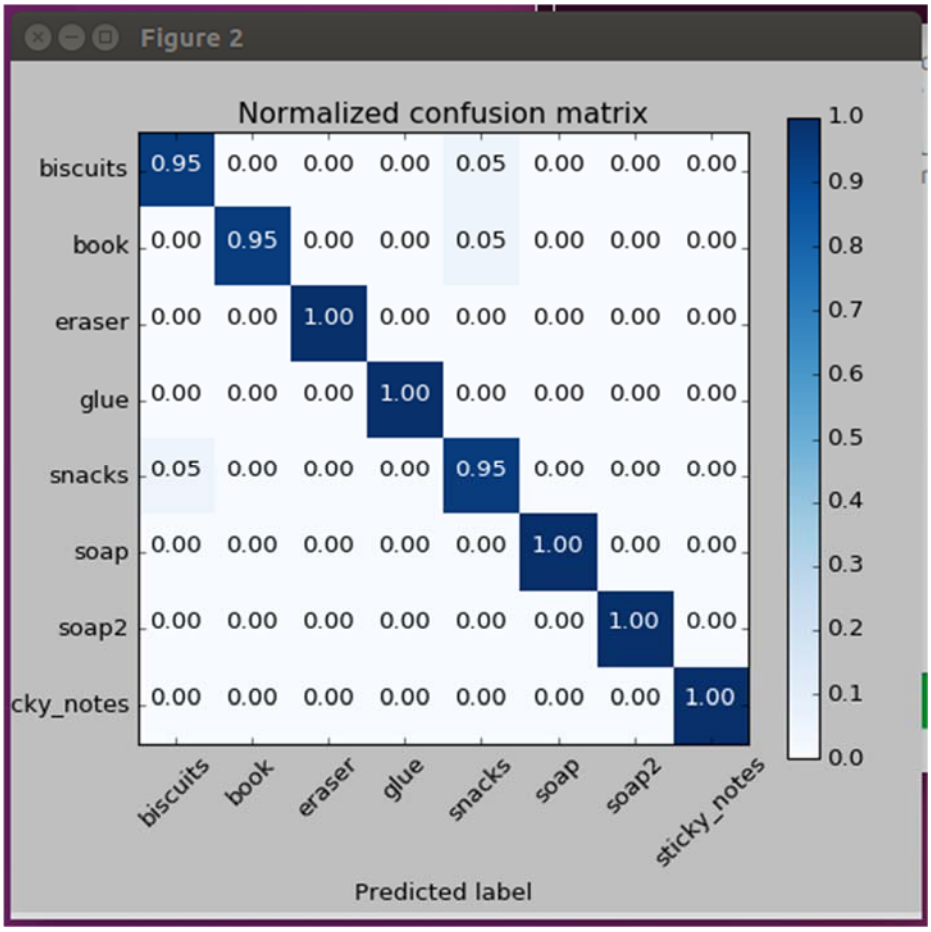


Figure 1
Confusion matrix, without normalization
biscuits
book
eraser
glue
snacks
soap
soap2
sticky_notes
True label
Predicted label
19.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
0.00 19.00 0.00 0.00 1.00 0.00 0.00 0.00
0.00 0.00 20.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 20.00 0.00 0.00 0.00 0.00
1.00 0.00 0.00 0.00 19.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 20.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 20.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 20.00
20
18
16
14
12
10
8
6
4
2
0



2. Write a ROS node and subscribe to `/pr2/world/points` topic. This topic contains noisy point cloud data that you must work with.

Copied the project_template.py script and named it **object_recognition.py (The node I wrote)** and subscribed to the specified topic to receive noisy camera data

3. Use filtering and RANSAC plane fitting to isolate the objects of interest from the rest of the scene.

- Applied Statistical filter to remove the noisy points from the image using a pre defined threshold distance

- Applied Voxel filter to downsample the processed data

- Applied a PassThrough filter in the z axis to keep only the reigon of interest

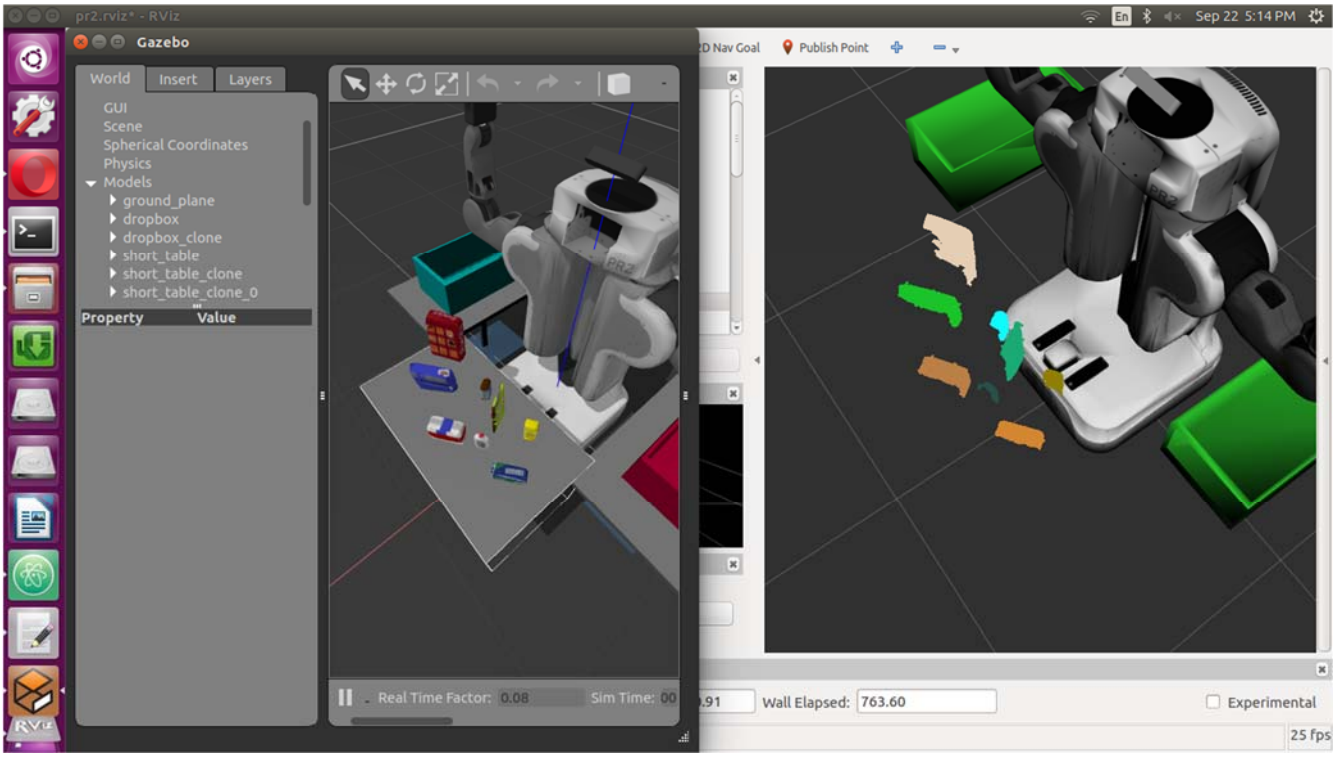
- Applied another PassThrough filter in the y axis, because the sides of the boxes were in the camera field of view and were picked up as clusters which would lead to false positives

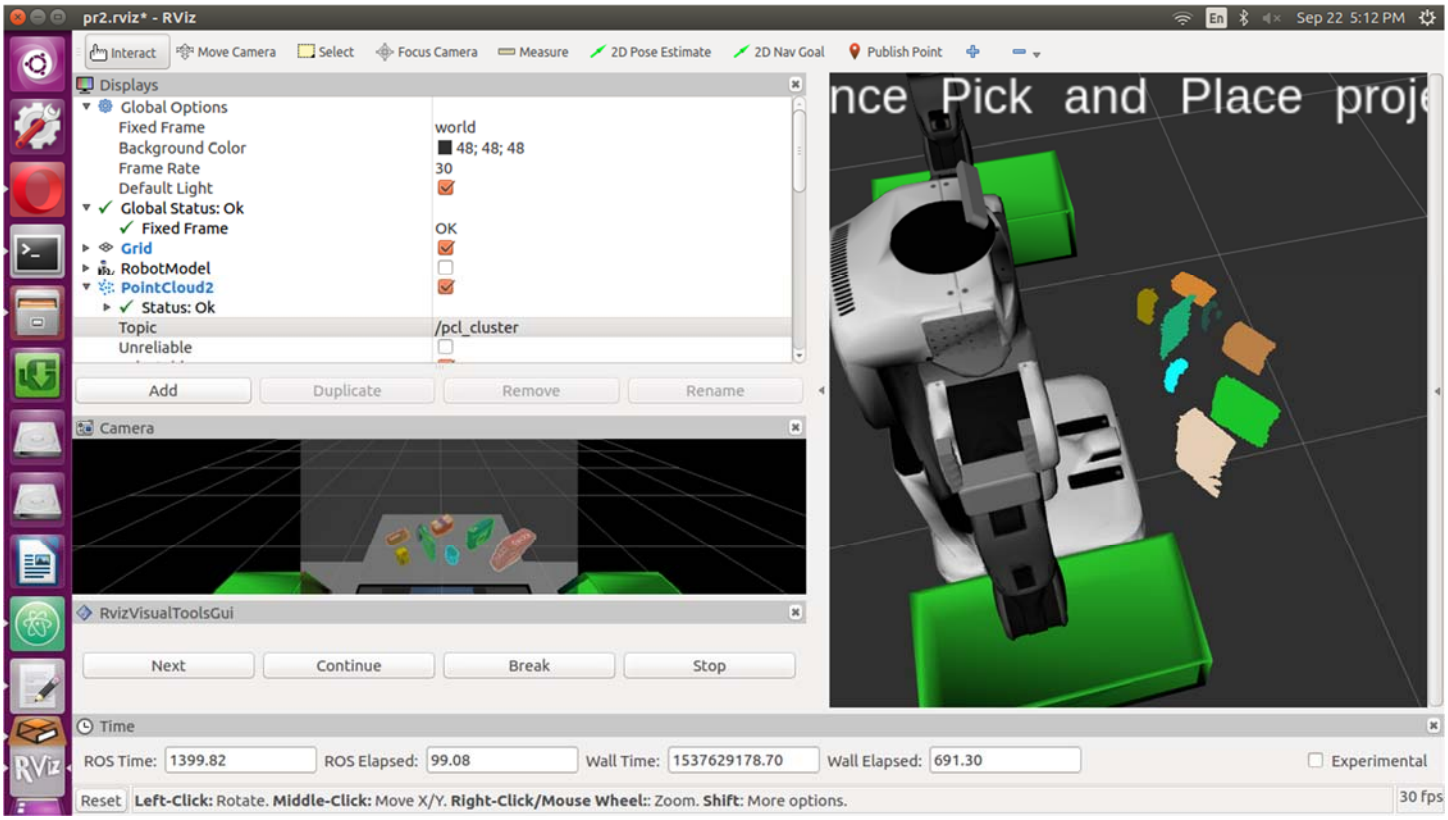
- Applied a RANSAC segmenter with the model of a plane to extract the indices that satisfy the plane model (the table) and those that don't (the objects)

4. Apply Euclidean clustering to create separate clusters for individual items.

Applied Euclidean clustering after specifing the required parameters (cluster elements max and min number, tolerance)

Used the kd-tree method to minimize computation time

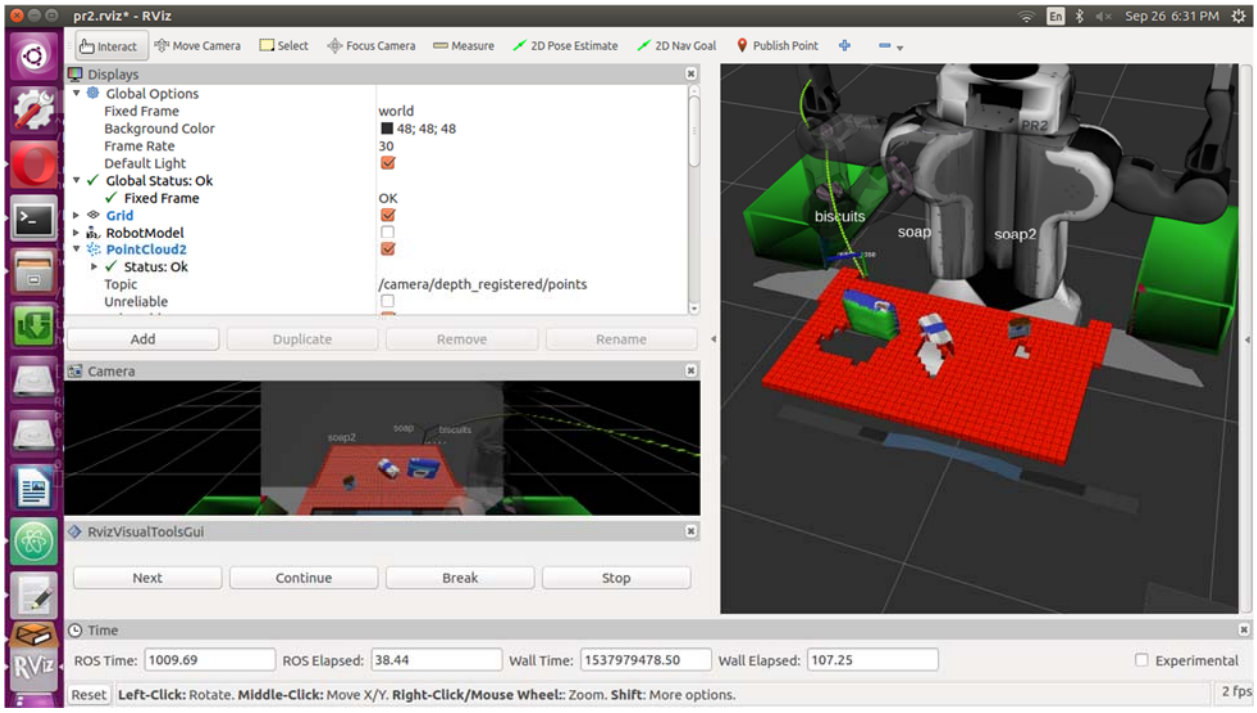


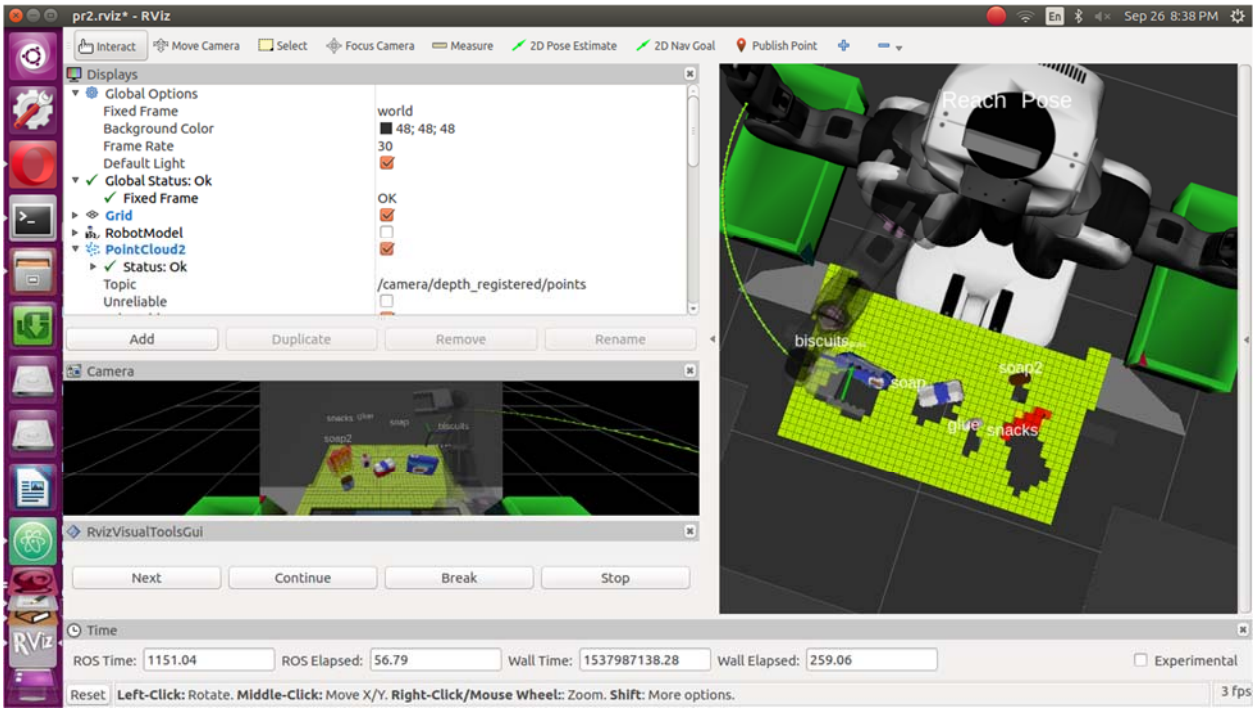


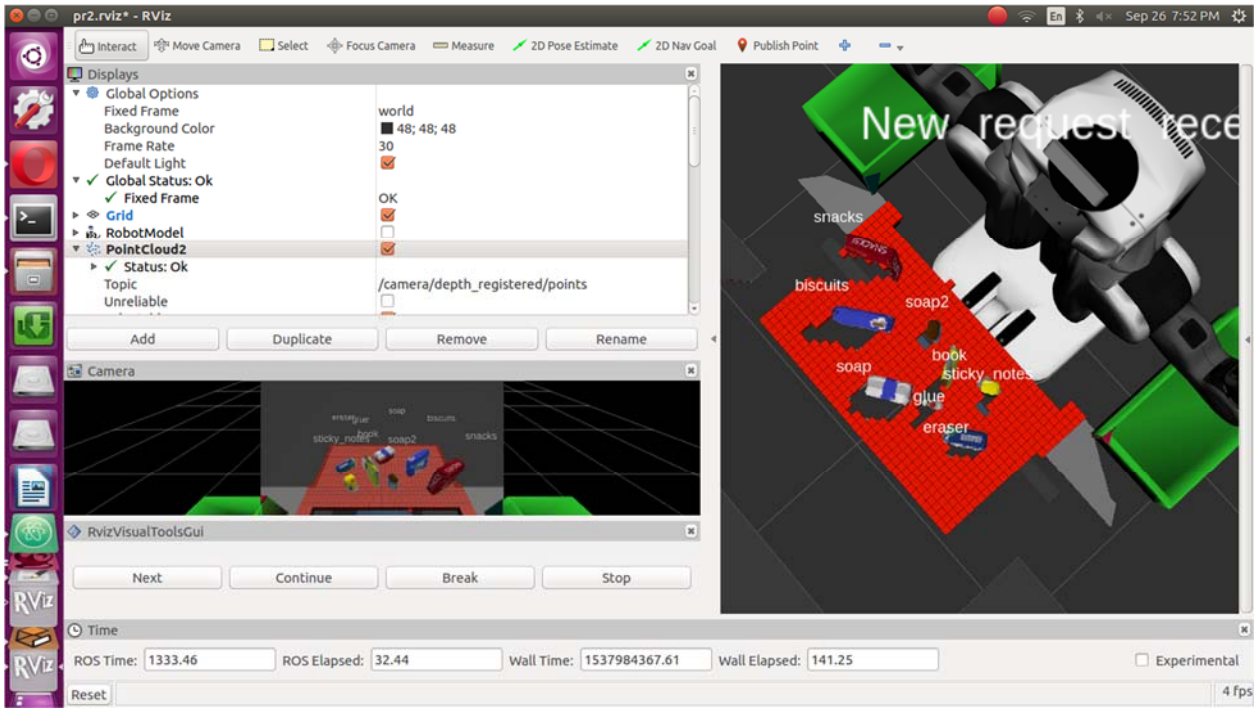
5. Perform object recognition on these objects and assign them labels (markers in Rviz).

Used the previously trained SVM model to identify the clusters obtained from the Euclidean Clustering

The 3 images are from the 3 test worlds.


pr2.rviz* - RViz
Sep 26 6:31 PM
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
world
Background Color
48; 48; 48
Frame Rate
30
Default Light
Global Status: Ok
Fixed Frame
OK
Grid
RobotModel
PointCloud2
Status: Ok
Topic
/camera/depth_registered/points
Unreliable
Add
Duplicate
Remove
Rename
Camera
RvizVisualToolsGui
Next
Continue
Break
Stop
biscuits
soap
soap2
PR2
Time
ROS Time:
1009.69
ROS Elapsed:
38.44
Wall Time:
1537979478.50
Wall Elapsed:
107.25
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
2 fps





6. Calculate the centroid (average in x, y and z) of the set of points belonging to that each object.

I calculated the XYZ position for each required object to be picked up from the

mean value of all points components in the object, except for the Z axis

```
    # For each required object add all the points' components to the lists
    for point in found_cluster:
        x_list.append(point[0])
        y_list.append(point[1])
        z_list.append(point[2])

    # Calculate the points mean value
    cent_x = np.sum(x_list)/len(x_list)
    cent_x = round(cent_x,3)
    cent_y = np.sum(y_list)/len(y_list)
    cent_y = round(cent_y,3)

    # Due to the camera prespective (looks downward onto objects), it
    # captures the points on top of the objects but not the bottom

    # So calculating the mean using the previous method leads to pulling
    # mean value up from the real center of the objects as more points exist
    # in the far end on the upper half than the lower half, and the robot
    # tries to grap in the space on top of the object

    # So we use the average between the max and min point instead and that
    # calculates the origin correctly and the robot is able to grab the
    # object
    cent_z = np.min(z_list) + ((np.max(z_list)-np.min(z_list))/2)
    cent_z = round(cent_z,3)
```

Due to the camera perspective (looks downward onto objects), it captures the points on top of the objects but not the bottom

So calculating the mean using the previous method leads to pulling mean value up from the real center of the objects as more points exist in the far end on the upper half than the lower half, and the robot tries to grap in the space on top of the object

So I used the average between the max and min points instead, and that calculates the origin correctly and the robot is able to grab the object

7. Create ROS messages containing the details of each object (name, pick_pose, etc.) and write these messages out to `.yaml` files, one for each of the 3 scenarios (`test1-3.world` in `/pr2_robot/worlds/`).

After extracting all the required object in the pick list by comparing it to the identified objects on table from the SVM model, I published them to the required topics

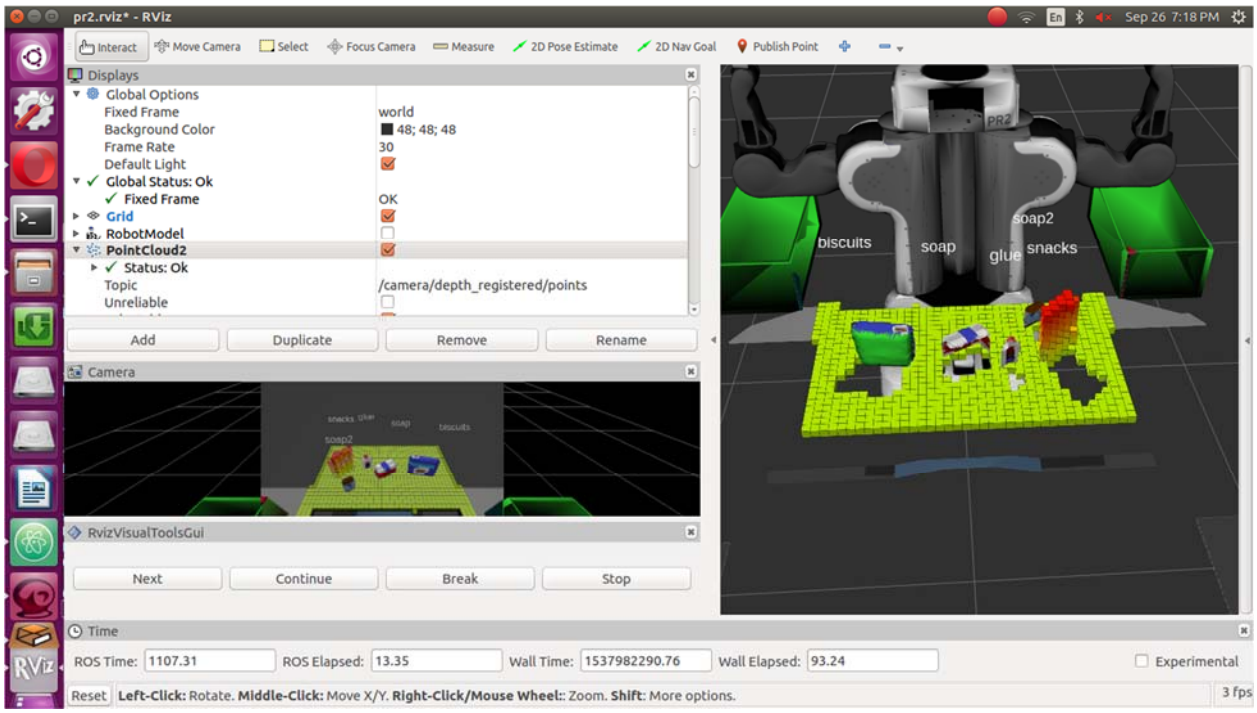
8. Submit a link to your GitHub repo for the project or the Python code for your perception pipeline and your output `.yaml` files (3 `.yaml` files, one for each test world). You must have correctly identified 100% of objects from `pick_list_1.yaml` for `test1.world`, 80% of items from `pick_list_2.yaml` for `test2.world` and 75% of items from `pick_list_3.yaml` in `test3.world`.
The link is
https://github.com/mohammed3asfour/RoboND-Perception-Project.git

Extra Challenges: Complete the Pick & Place
7. To create a collision map, publish a point cloud to the `/pr2/3d_map/points` topic and make sure you change the `point_cloud_topic` to `/pr2/3d_map/points` in `sensors.yaml` in the `/pr2_robot/config/` directory.

I did as the point says, but the node reciving the the collision cloud takes only the first msg and builds on it without resetting which make us unable to make a different collision cloud for each pickup due to objects already picked up.

So I will make one collision cloud with all the objects that won't be picked up.

The code for generating different collision clouds is also included as comments .



8. Rotate the robot to generate collision map of table sides. This can be accomplished by publishing joint angle value(in radians) to `/pr2/world_joint_controller/command`

9. Rotate the robot back to its original state.

For the two steps above.
It took a lot of time, and I tried without rotating the robot and it didn't affect the performance so I removed this step to save time specially the limited resources I have for the sim.

10. Create a ROS Client for the rosservice. In the required steps above, you already created the messages you need to use this service. Checkout the [PickPlace.srv]

I have sent the required message

11. If everything was done correctly, when you pass the appropriate messages to the `pick_place_routine` service, the selected arm will perform pick and place operation and display trajectory in the RViz window

It did and I hit continue to automate the transition between the next steps

12. Place all the objects from your pick list in their respective dropoff box and you have completed the challenge!

Have successfully done that for objects, But the grasp is very buggy which lead to the slipping of the object or even not closing the gripper properly on it.

The trajectory planning doesn't take in consideration the object that the robot is holding which makes the robot's arm evade the objects but the object it holds doesn't necessarily.

I have included two GIFs (3X Speed) in the folder (World 1/PickList 1)and (World 3/PickList 1) to show you the arm picking the object required and placing it correctly in the bin
For World 1 the arm gripper doesn't successfully graps the objects even though it reaches the correct pick up position

In World 3, I wanted to demonstrate the collision cloud as it contains objects that aren't required to be picked up

Extra Notes

I have also Included the training set file, the svm model file, the 3 output files from the 3 worlds, and 2 GIFs for the robot performing the pick and place action. (the GIFs are heavy and may crash the system image viewer, if so please use an internet browser and it will work just fine.)

The performance can be enhanced by obtaining a new collision cloud for each item in the pick list and by rotating the robot to scan enviroment further but with the expense of simulation time.