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).
2. Write a ROS node and subscribe to `/pr2/world/points` topic. This topic contains noisy point cloud data that you must work with.
3. Use filtering and RANSAC plane fitting to isolate the objects of interest from the rest of the scene.
4. Apply Euclidean clustering to create separate clusters for individual items.
5. Perform object recognition on these objects and assign them labels (markers in RViz).
6. Calculate the centroid (average in x, y and z) of the set of points belonging to that each 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/`). See the example `output.yaml` for details on what the output should look like.
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`.
9. Congratulations! Your Done!

Extra Challenges: Complete the Pick & Place

1. 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. This topic is read by Moveit!, which uses this point cloud input to generate a collision map, allowing the robot to plan its trajectory. Keep in mind that later when you go to pick up an object, you must first remove it from this point cloud so it is removed from the collision map!
2. 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`
3. Rotate the robot back to its original state.
4. Create a ROS Client for the “pick_place_routine” rosservice. In the required steps above, you already created the messages you need to use this service. Checkout the PickPlace.srv file to find out what arguments you must pass to this service.
5. 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
6. Place all the objects from your pick list in their respective dropoff box and you have completed the challenge!
7. Looking for a bigger challenge? Load up the `challenge.world` scenario and see if you can get your

perception pipeline working there!

Rubric Points

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.

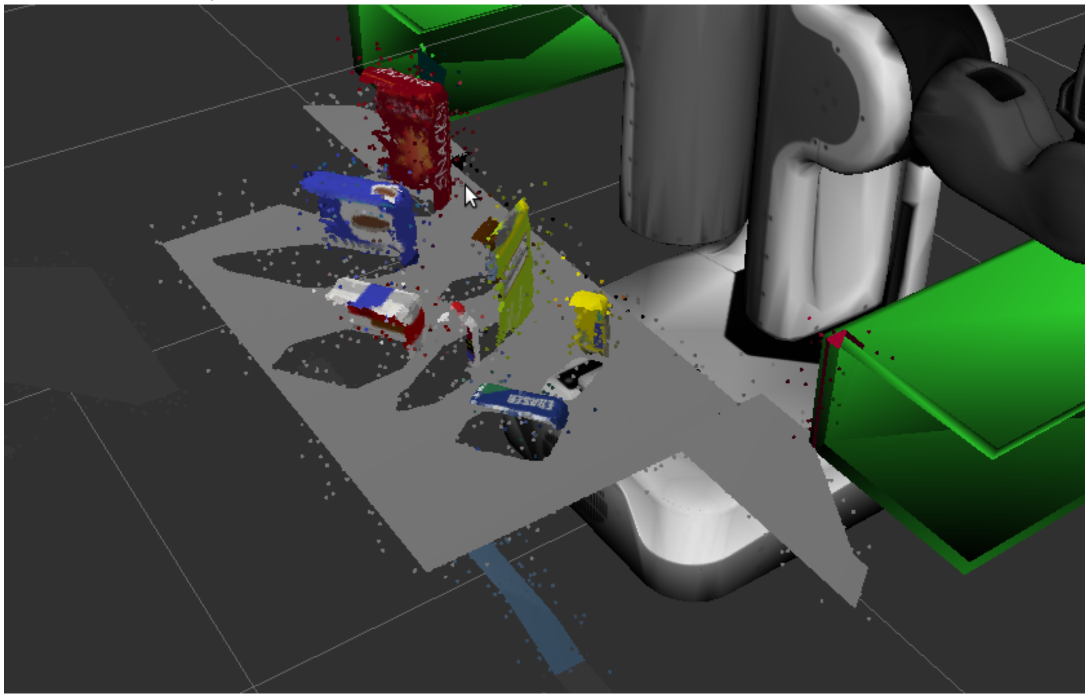
You're reading it!

Exercise 1, 2 and 3 pipeline implemented

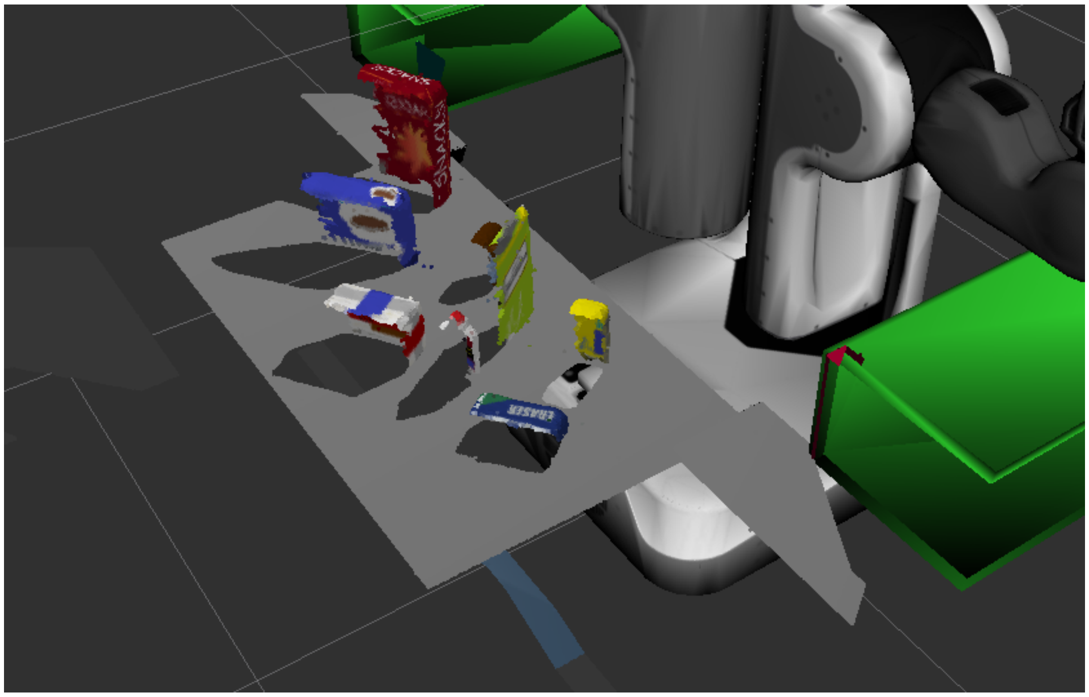
1. Complete Exercise 1 steps. Pipeline for filtering and RANSAC plane fitting implemented.

In this step we are trying to solve the following problem. We have th scene represented as a point cloud but there is a lot of spurious data. In the end we are interested in the objects on the table. We also have table, some background, possibly noise. Here we design a set of operations to extract only the points we are interested in. That is the points of the objects on the table. We will do it in these steps (the links lead to github to the commented code).

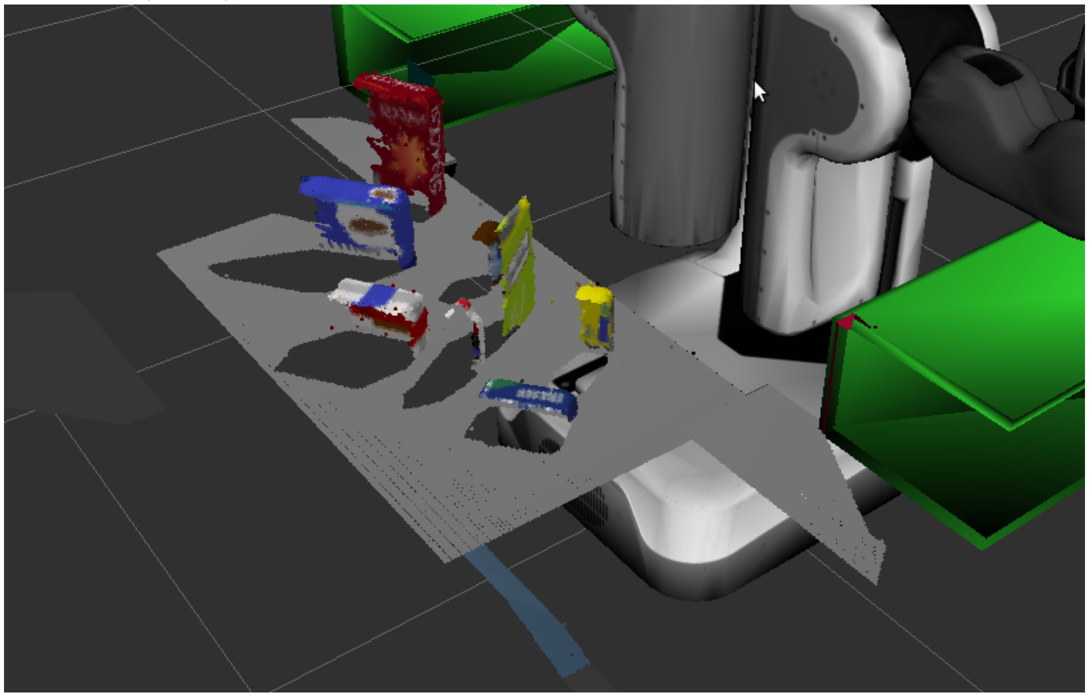
- We start with a noisy scene



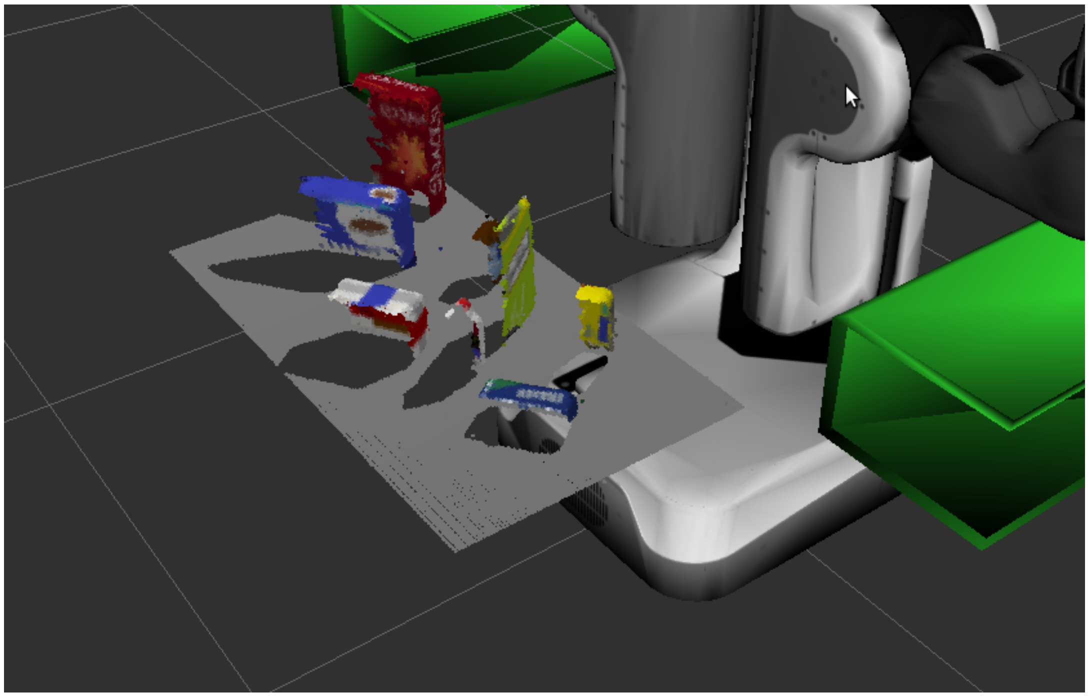
- We remove the noise



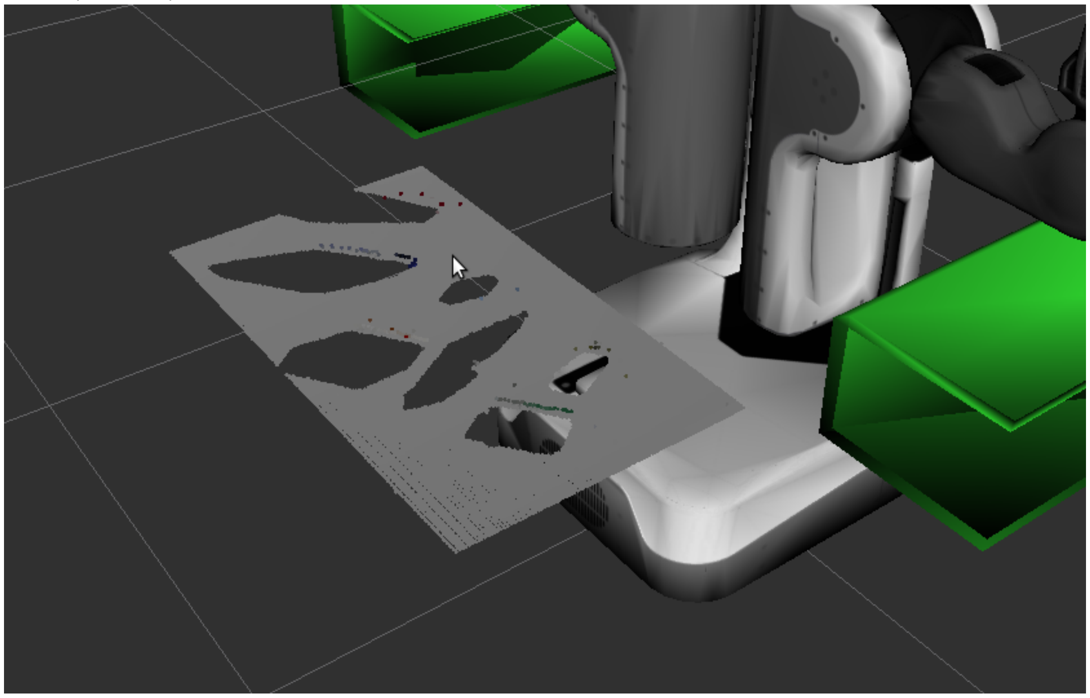
- We downsample the point cloud



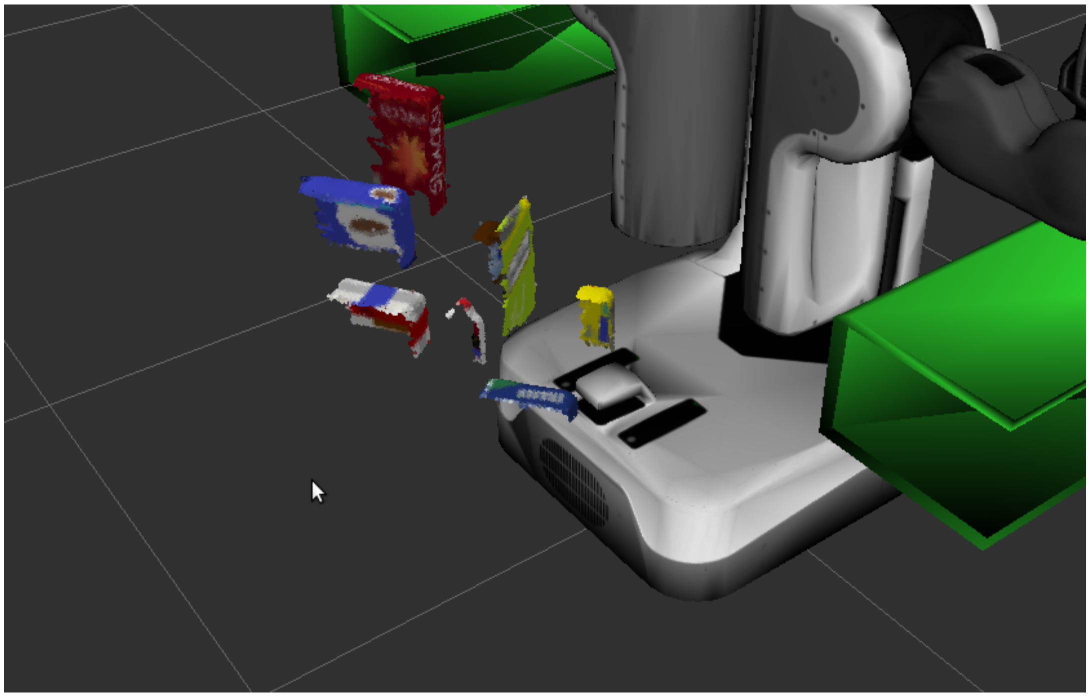
- perform passfilters. This effectively cuts a box like shape from the space. We perform passes along z and y lane. We aim to remove pieces of the table



- RANSAC detection on plane. We try to remove the table
  - Inliers (the table)



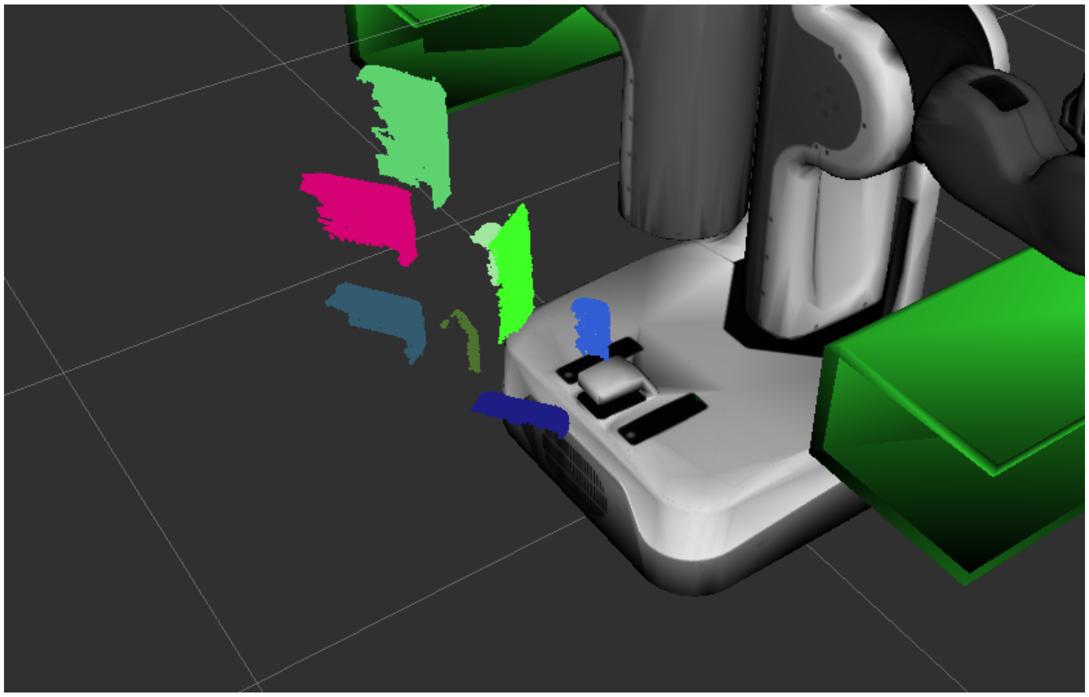
  - Outliers (the table)



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

In this step we completed the clustering of points. The reason for this is that after first step we have only points of interest but they are still part of one point could. Now we try to segment them into individual point clouds ideally representing individual objects. For that we are using Euclidean Clustering (DBSCAN). It is briefly described here.

This is the picture of showing the colored clusters



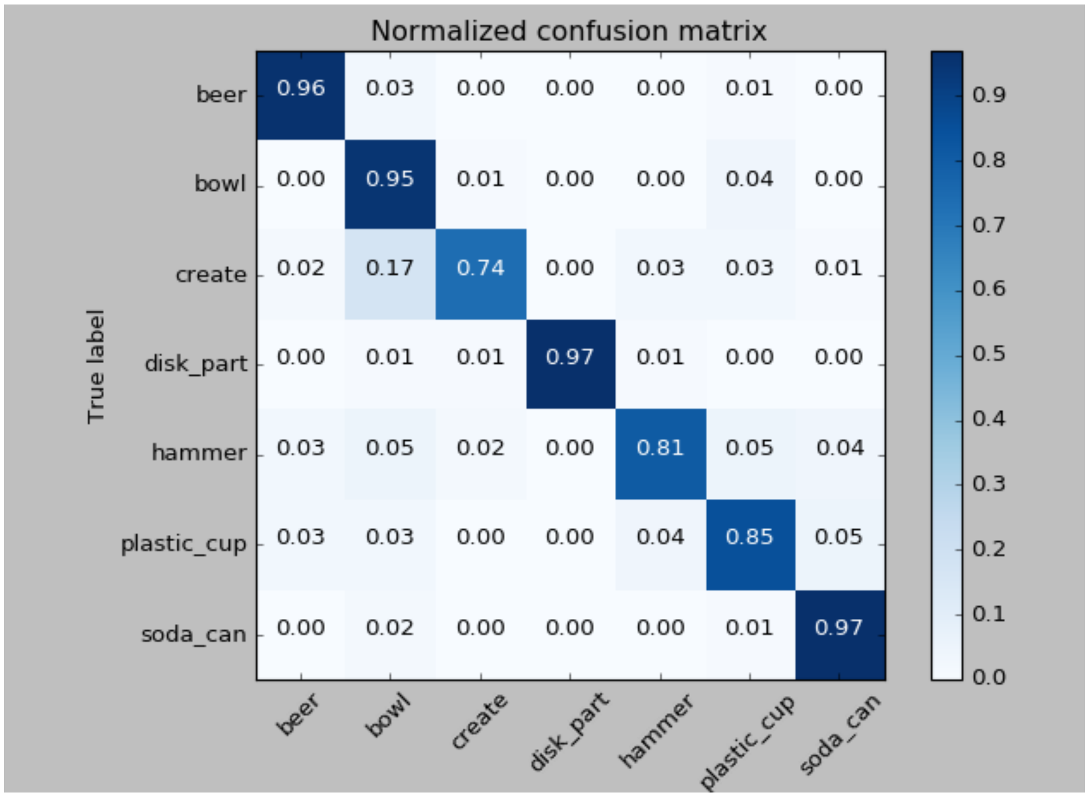
3. Complete Exercise 3 Steps. Features extracted and SVM trained. Object recognition implemented.

To be able to train a classifier we have to first prepare the features. This happens roughly like this. The object "image" is captured in a simulator by rotating randomly an object in fron of camera. We then preprocess it with a script that captures 2 histograms.

- color histogram
- normal histogram

And serialize to a file with a label which is later used for training.

Now we are going to train a classifier. We are sticking with the default SVM. We are showing the confusion matrix which represents the performance of the multiclass classification in a concise and visual manner (for brevity confusion matrix is provided and discussed only for the world 3 which is most complex).



Here we see that the performance is decent most of the objects is correctly classified with accuracy over 90 percent. Worse performance can be seen on create that gets misclassified as bowl as much as in 20 percent of cases. Here the feature was generated on 300 snapshots of each object.

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.

Here we are going to apply our model to the pipeline and generate messages that can be used for either instructing robot to pick and place projects or to generate messages to persist.

The steps needed to perform the task in my case are as follows

- open 2 terminal window.
    - one does not really matter where. Be sure to have GAZEBO_MODEL_PATH exported correctly
    - second place into directory `/home/robond/catkin_ws/src/RoboND-Perception-Project/pr2_robot/scripts`
-
- in the second window copy the model to the working directory `cp ../trained_models/model_3.sav .`
- change world_name variable on line 27 in project_template.py to have the number of the world you are

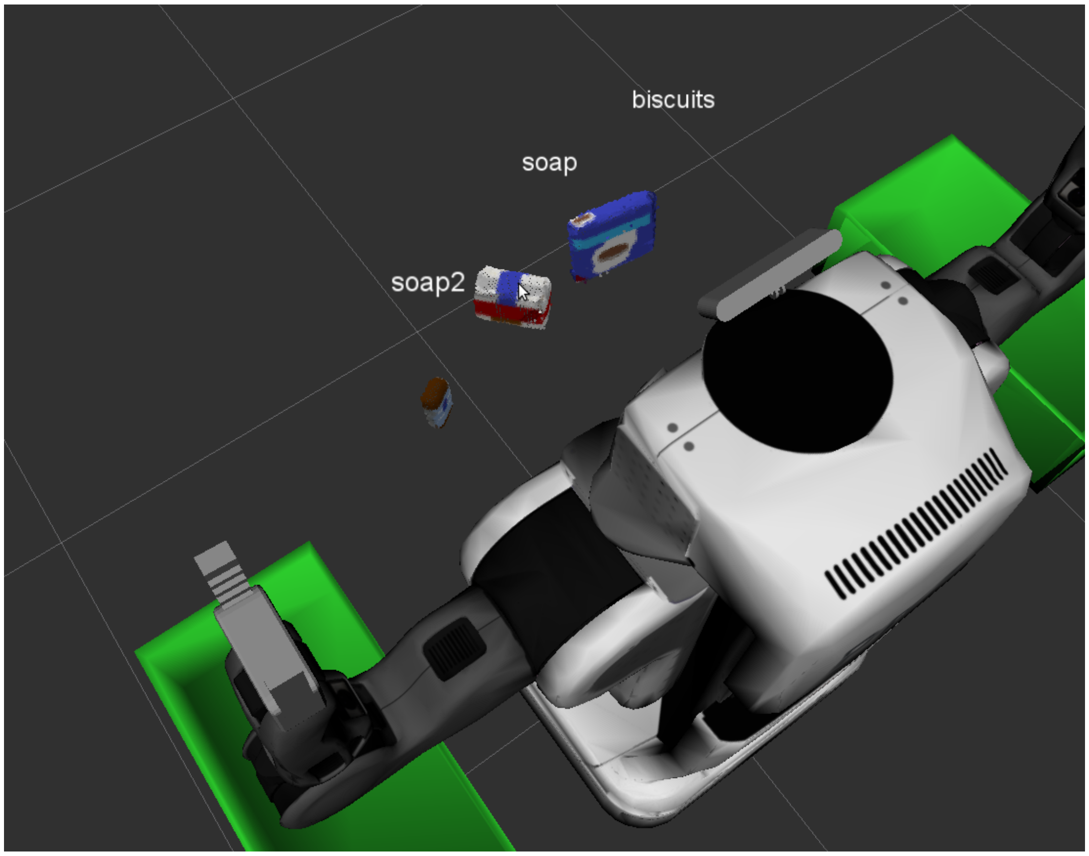
about to explore in our case here

world_name = 3

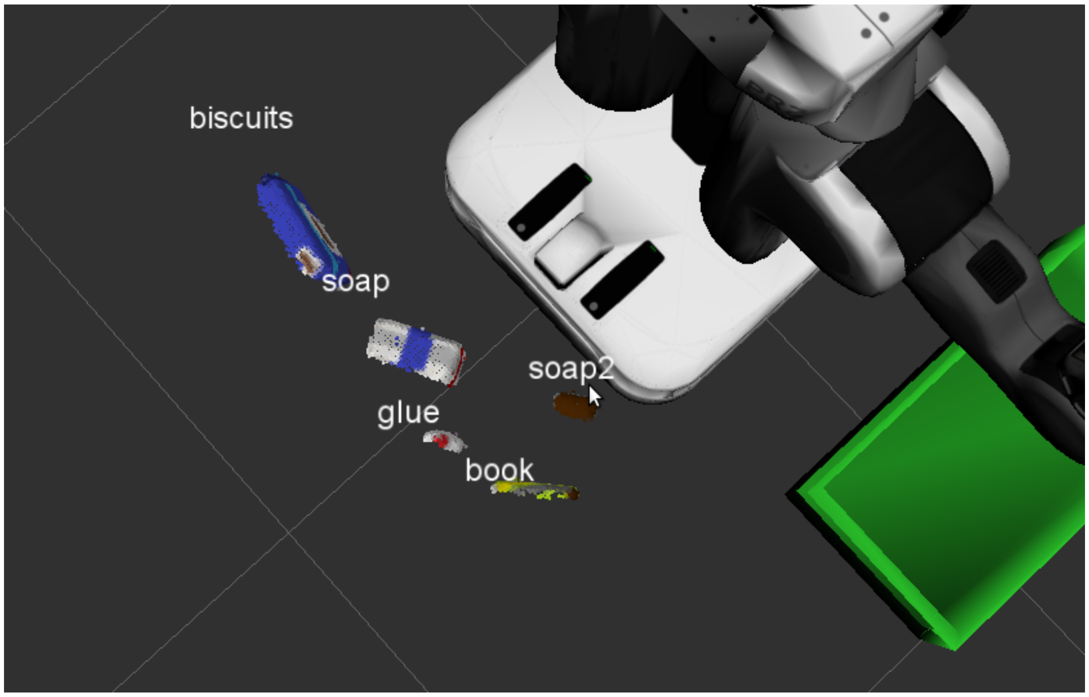
- execute `roslaunch pr2_robot pick_place_project.launch`
- execute `rosrun pr2_robot project_template.py` in the second window

Here are the results for the respective worlds

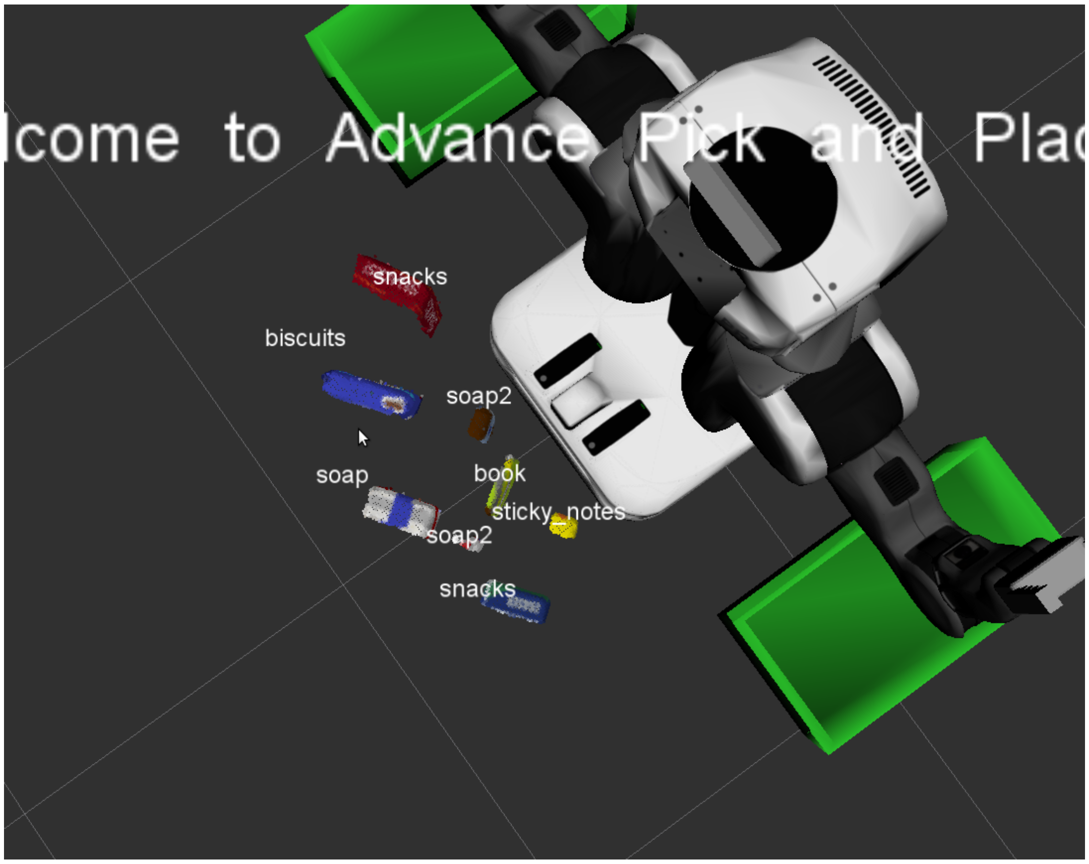
- World 1



- World 2



- World 3



The yaml files with the messages are here.

Discussion

The implementation went fairly smoothly. The PCL library si fantastic and contains majority of tools we need. I had to alter the pipeline a little bit. I had troubles with isolating the table with just the RANSAC but that is fine. Every project is different and there are many paths to the result.

Another problem is the detection itself. As seen on the confusion matrix the classifier performs really well during training but the performance drops once put into RVIZ. I think there needs to be done further work on tuning the hyperparameters. I tried the following

- Switch to HSV
- I did more snapshots per object which helped but the caveat is longer training

Each of these improved the performance

- What I think could help is try changing couple of different classifiers

In the last world we can see there are 2 misclassifications. One is the glue but it is partially ocluded plus it has similar shape and colors like the soap. The other misclassification is the eraser. While similar in shape to buscuits it has very different colot palette.