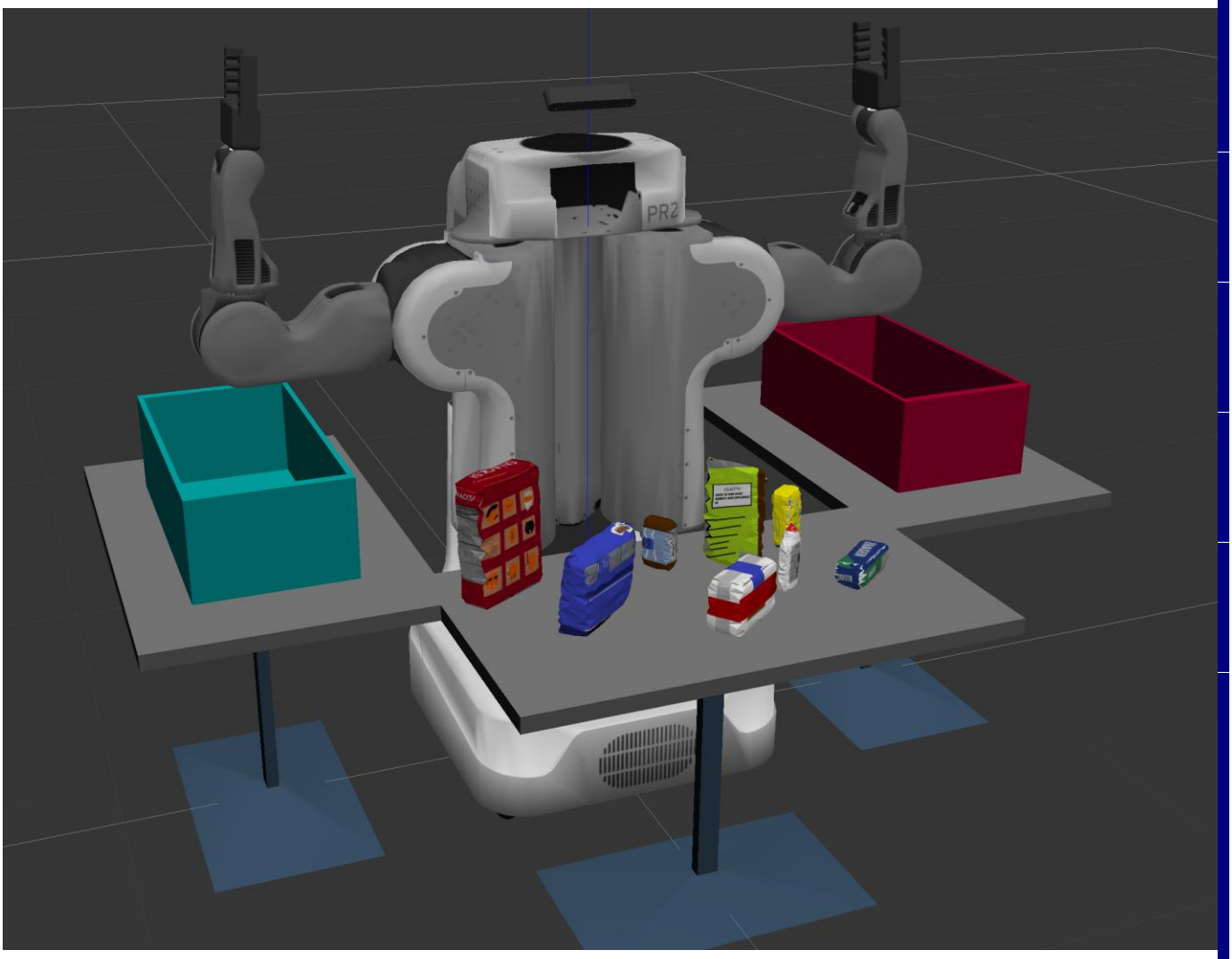
## Perception Project

In this project, I utilize RGB_D camera to capture images with different poses and then used Machine Learning(ML) SVM model to make a classifier to identify target objects in the tabletop. And used PR2-Robot to pick & place objects in corresponding drop boxes.

But before this, you have to know that: sensor has a bit noise besides objects may have some dust so, we have to filter and extract features to recognition them using some algorithms.



## Procedure to solve this project :

**1**. calibrating your camera to avoid unnecessary distortion.

**2**. 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).

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

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

**5**. Apply Euclidean clustering to create separate clusters for individual items.

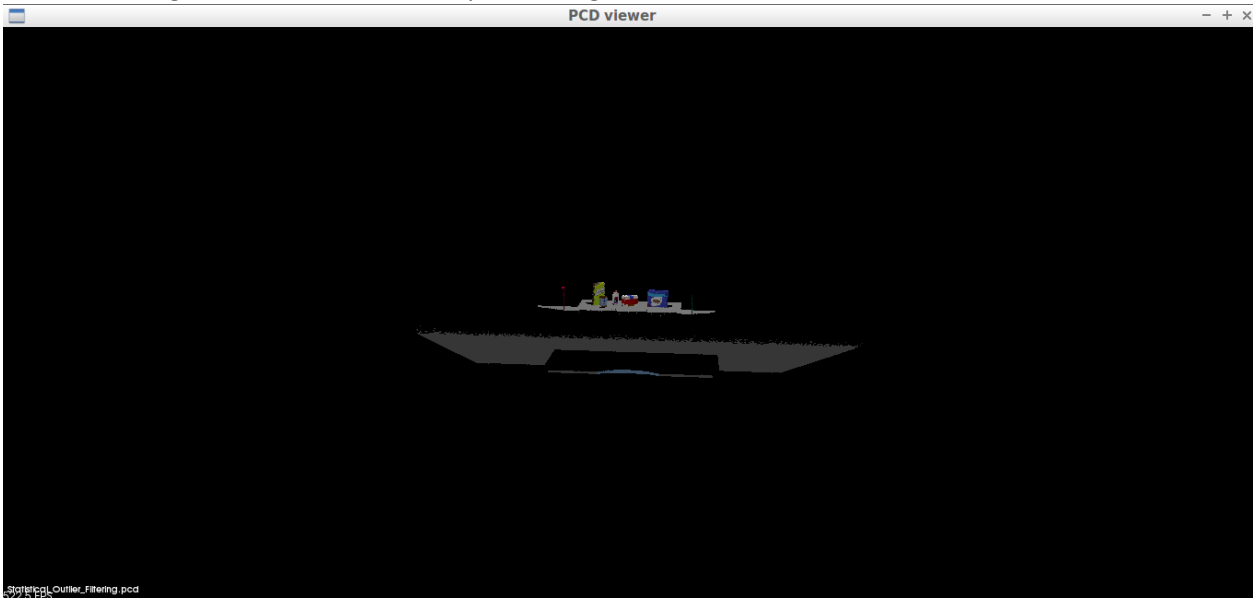
**6**. Perform object recognition on these objects and assign them labels (markers in RViz).

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

**8**. 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.

**9**. I 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`.

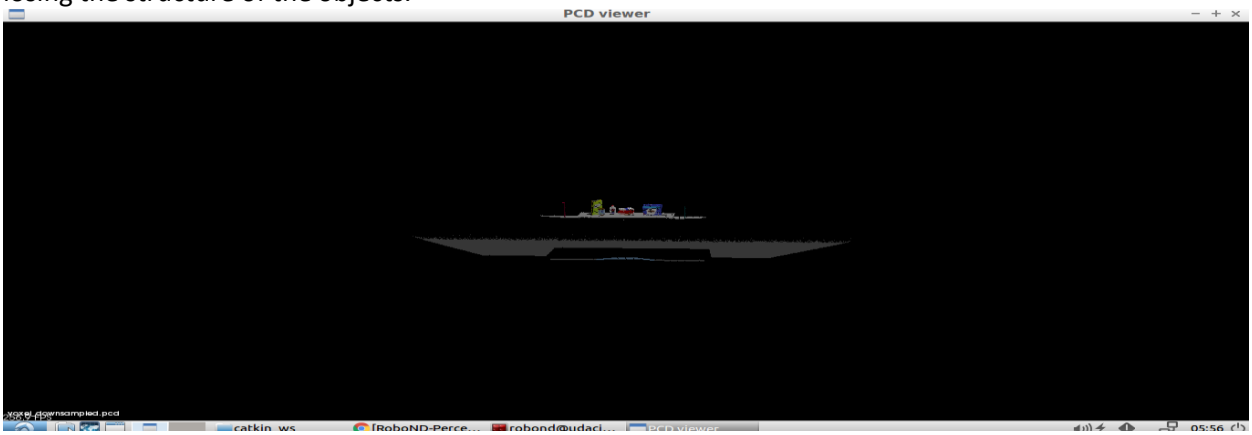
The RGBD is stored as point cloud data to only keep the essential data, removing adversarial data points, and compressing the cloud data.
The raw point cloud object from the PR2 simulation looks the image above. After applying outlier filtering we can remove outliers from the point cloud. Any points whose mean distances are outside a defined interval are removed.
I have tuned the parameters for k = 15 and a standard deviation threshold of 0.03 provided the optimal outlier filtering. Here is the cloud after performing the outlier removal filter.




**outlier removal filtering**

**Voxel grid filter**

After filetring still the data in point is too high we which would require high computational power to process them A voxel grid filter downsamples the data by taking a spatial average of the points in the cloud confined by each voxel. The set of points which lie within the bounds of a voxel are assigned to that voxel and are statistically combined into one output point.
I used an X, Y, and Z voxel grid filter leaf size = 0.01. Which retains the required information, without losing the structure of the objects.




**Voxel grid filter**

**passthrough filter**

To crop out the unnecessary parts from the Point cloud we use Passthrough filter.

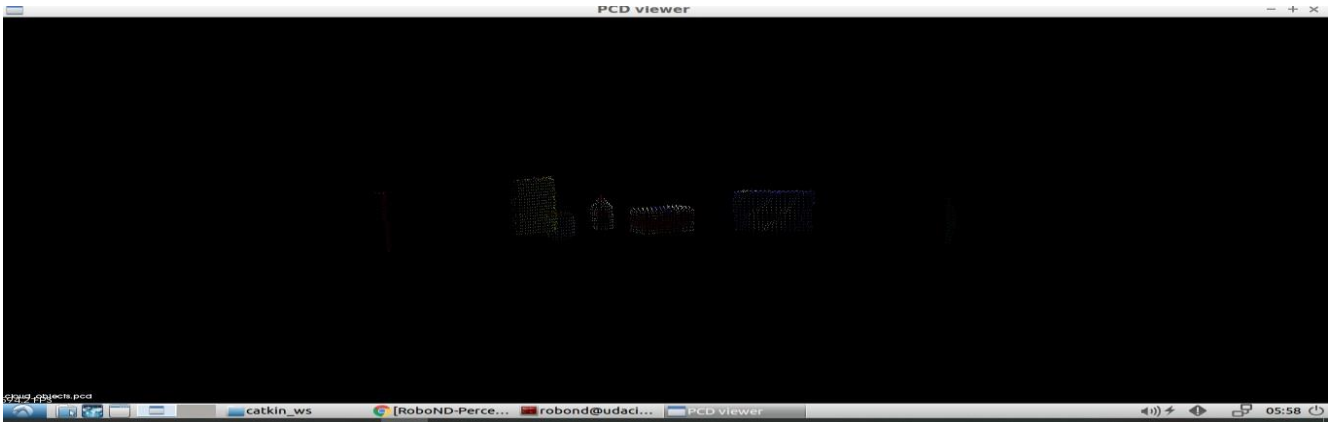
In PR2 robot simulation we use passthrough filters for both the Y and Z axis. For the Y axis, range was -0.4 to 0.4, and for the Z axis,range was 0.61 to 0.9.


**RANSAC Plane Segmentation**

Random Sample Consensus (RANSAC) is used to identify points in the dataset that belong to a particular model.RANSAC can be used to find a particular shape like plane, where to remove table we need a shap, which was a plane,and inliers which can be defined by a particular model with a specific set of parameters, and outliers don't.

I used a RANSAC max distance = 0.01.

The extracted outliers contains the objects on the table, and looks like this:




**RANSAC plane segmentation - extracted outliers**

**Clustering for Segmentation**

To seperate all the objects , where we have a clean point cloud data of outliers which are objects. The two main algorithms possible include:

K-means

K-means clustering algorithm is able to group data points into n groups based on their distance to randomly chosen centroids. But the drawbacks with this algorithm is we have to know how many clusters , we need.

DBSCAN :Density-based spatial cluster of applications with noise (DBSCAN) which clusters data points that are within some threshold distance from their nearest neighbor.

You don't need to know how many clusters to expect in the data. However, you do need to know something about the density of the data points being clustered.

**DBSCAN object cluster**

DBSCAN within the PR2 simulation required converting the XYZRGB point cloud to a XYZ point cloud, making a k-d tree (decreases the computation required), preparing the Euclidean clustering function, and extracting the clusters from the cloud. This process generates a list of points for each detected object.

**Object Recognition**

The object recognition code allows each object within the object cluster to be identified. In order to do this, the system first needs to train a model to learn what each object looks like. Once it has this model, the system will be able to make predictions as to which object it sees.

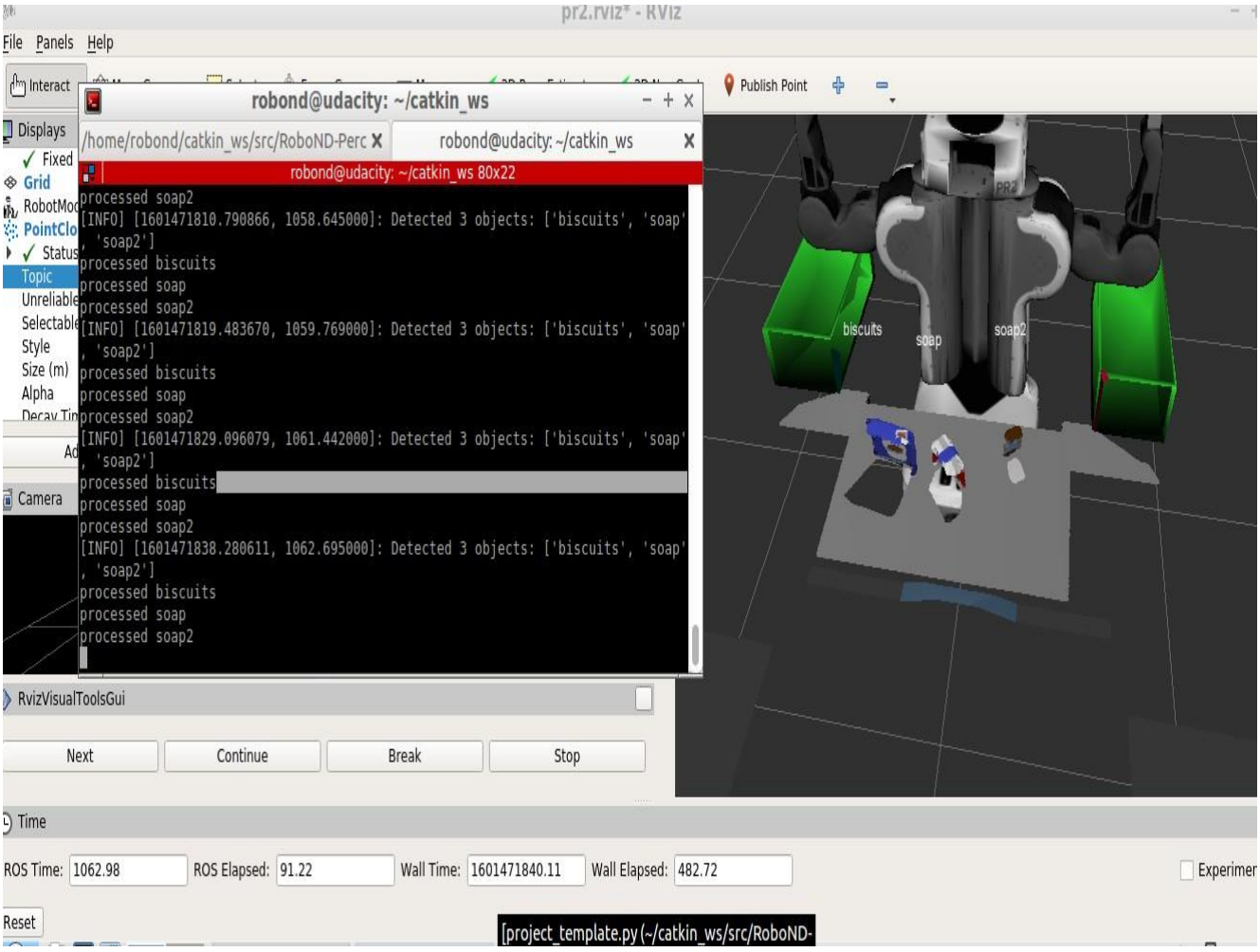
## #For Test 1

The parameters tuned are: ClusterTolerance=0.05 MinClusterSize=130 MaxClusterSize=3000

Passthrough_filter("y",-0.4,0.4)

Passthrough_filter("z",0.6,0.91)

**#Output : 3 objects with 100 /100**



## For Test 2

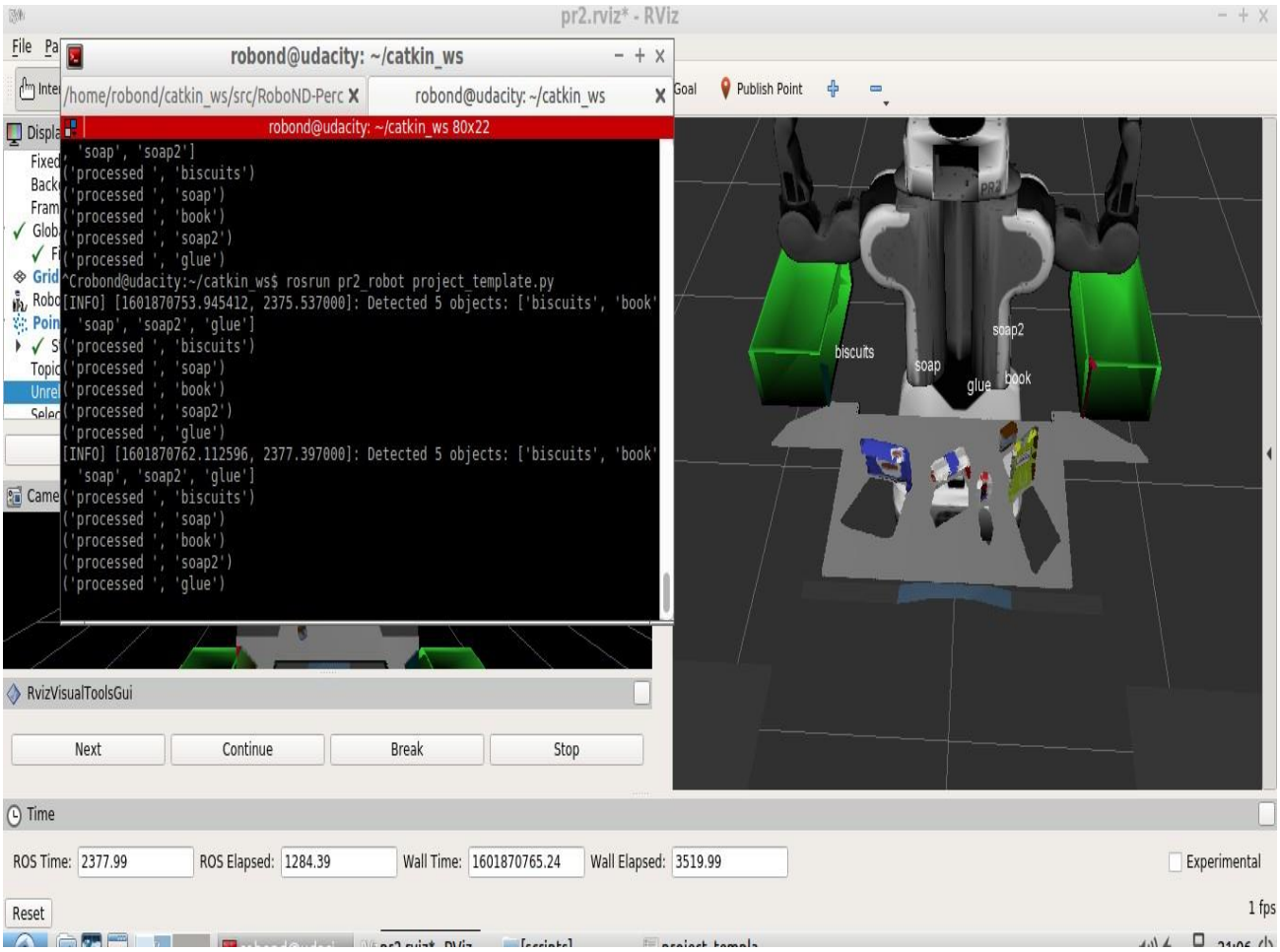
The parameters tuned are: ClusterTolerance=0.05 MinClusterSize=118 MaxClusterSize=3000

Passthrough_filter(“x”,0.08,0.6)

Passthrough_filter(“y”,-0.4,0.4)

Passthrough_filter(“z”,0.6,0.8)

**#Output : 5 objects with 100 /100**



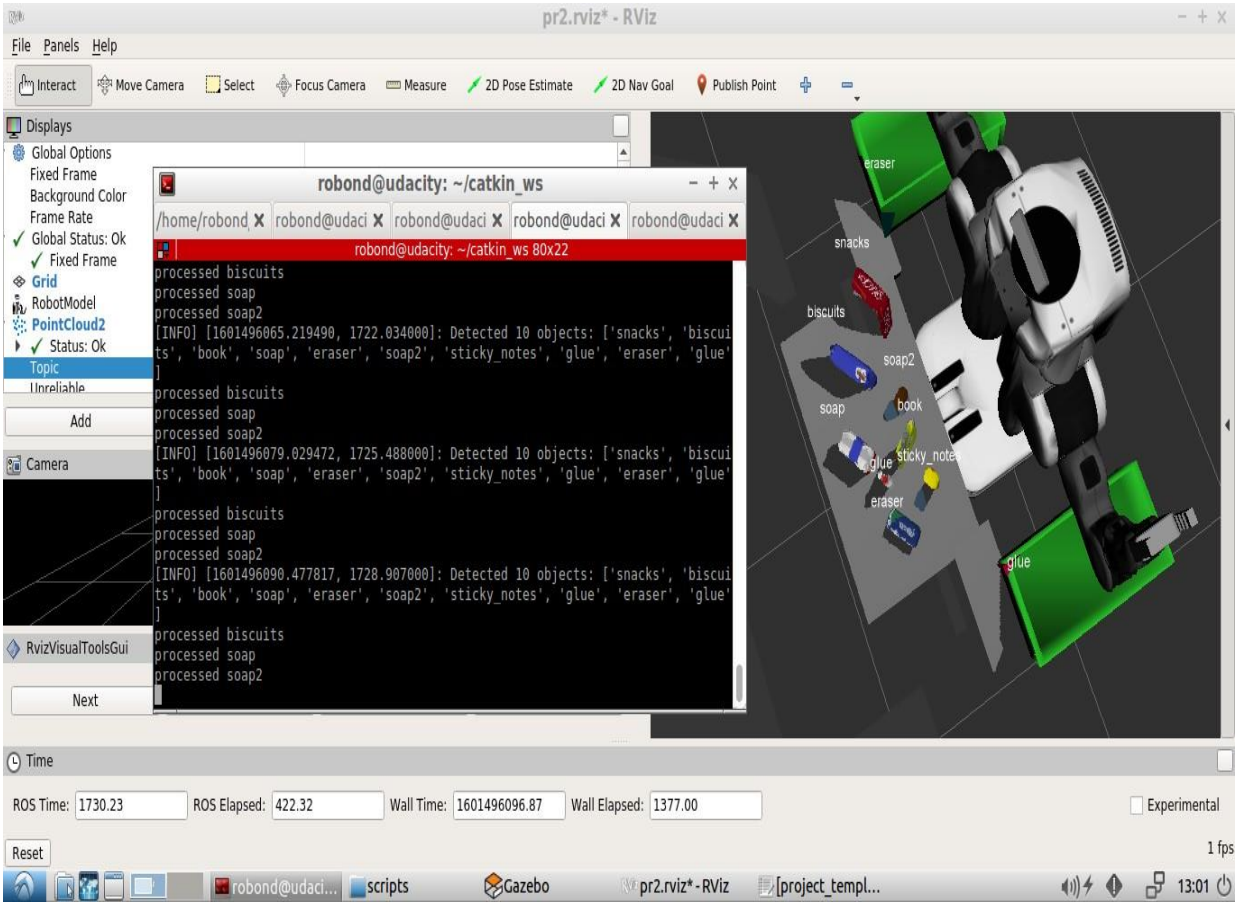
## For Test 3

The parameters tuned are: ClusterTolerance=0.05 MinClusterSize=60 MaxClusterSize=3000

Passthrough_filter("y",-0.4,0.4)

Passthrough_filter("z",0.6,0.91)

**#Output : 10 objects with 100 /100**



**Capture Object Features**

Color histograms are used to measure how each object looks when captured as an image. Each object is positioned in random orientations to give the model a more complete understanding. For better feature extraction, the RGB image can be converted to HSV before being analyzed as a histogram.

For capturing each object in 20 random orientations, using the HSV color space .

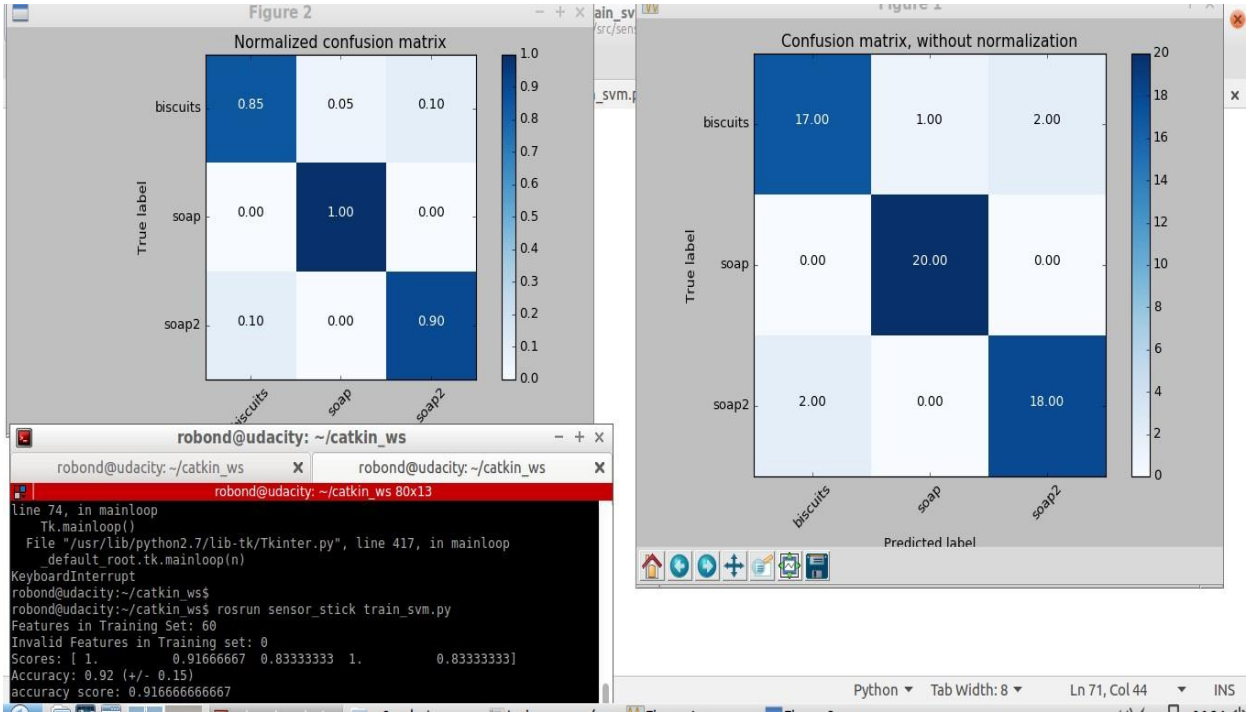
**Train SVM Model**

A support vector machine (SVM) is used to train the model (specifically a SVC).

## #For Test1

```
compute_color_histograms(sample_cloud, using_hsv=True,nbins=32)

compute_normal_histograms(normals,nbins=16)

training_set = pickle.load(open('training_set.sav', 'rb'))

# Create classifier with C=0.1

clf = svm.SVC(kernel='linear',C=0.1)

# Set up 5-fold cross-validation

kf = cross_validation.KFold(len(X_train), n_folds=5, shuffle=True, random_state=10)
```

**#Output : given 0.92 Accuracy and detected 3 objects**



## #For Test 2

compute_color_histograms(sample_cloud, using_hsv=True,nbins=32)

compute_normal_histograms(normals,nbins=16)

training_set = pickle.load(open('training_set_2.sav', 'rb'))

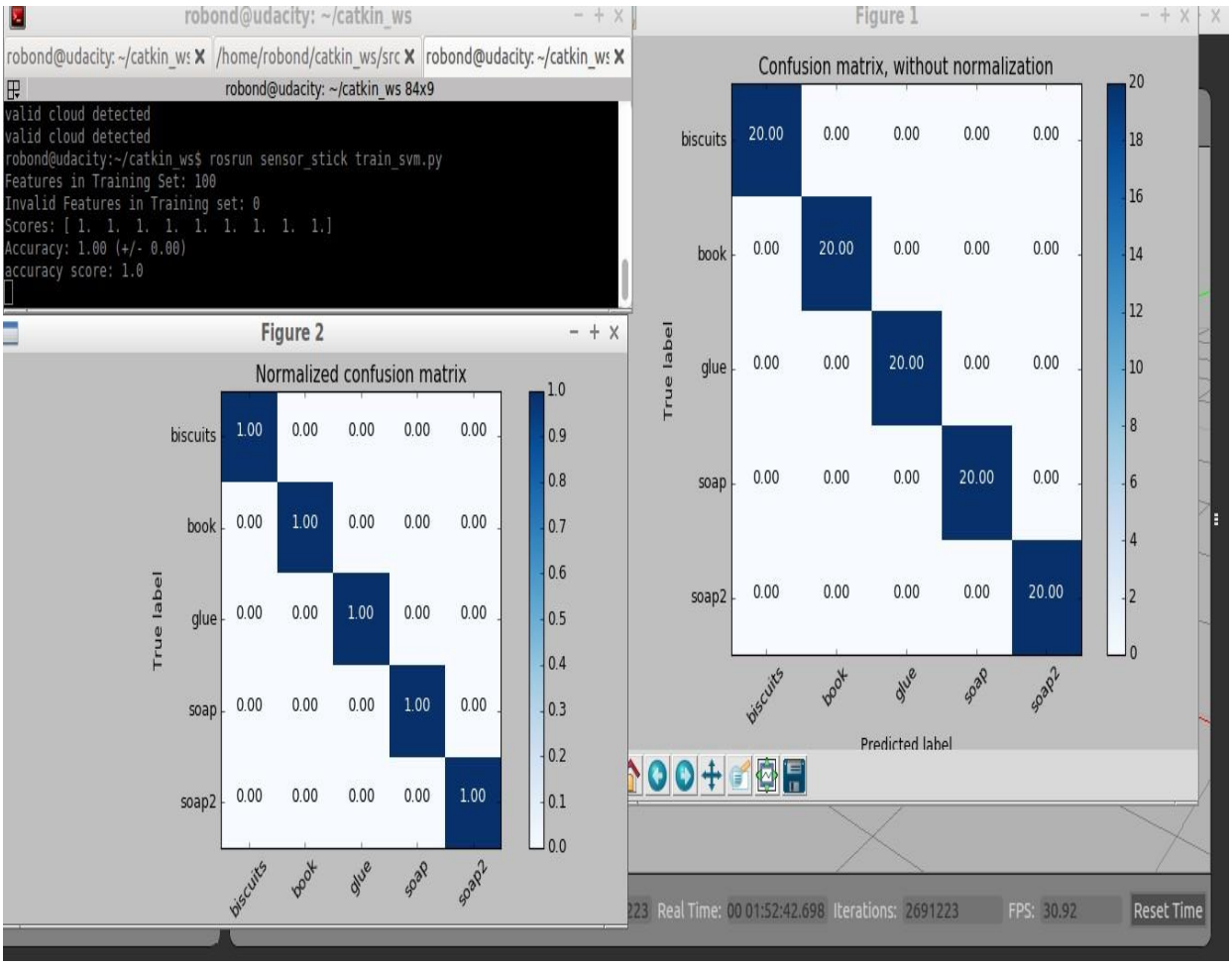
# Create classifier with C=0.1

clf = svm.SVC(kernel='linear',C=0.1)

# Set up 9-fold cross-validation

kf = cross_validation.KFold(len(X_train), n_folds=9, shuffle=True, random_state=110)

**#Output : given 1.0 Accuracy and detected 5 objects**



## #For Test 3

compute_color_histograms(sample_cloud, using_hsv=True,nbins=32)

compute_normal_histograms(normals,nbins=16)

training_set = pickle.load(open('training_set_3.sav', 'rb'))

# Create classifier with C=0.1

clf = svm.SVC(kernel='linear',C=0.1)

# Set up 9-fold cross-validation

kf = cross_validation.KFold(len(X_train), n_folds=9, shuffle=True, random_state=110)

**#Output : given 0.91 Accuracy and detected 10 objects**