# 3D Perception

The 3D perception project was implemented by adding the image processing and object recognition pipeline in *project_template.py*. Initially the obtained ROS cloud message was cleaned using outliner removal filter, then it was downsampled. After downsampling, a passthrough filter was applied to extract only those parts of the scene that included the tabletop and objects. The RANSAC filtering was performed to obtain cloud indices corresponding to the tabletop object, then these were excluded from the cloud to obtain only those cloud points that corresponded to non-tabletop points. The image processing was completed by applying Euclidian clustering to identify clusters where the cloud points are spatially close to each other. Finally to perform object recognition, for each cluster in the cluster list, a feature vector was constructed out of the color and shape histograms. The object label prediction was made using the pre-learned support vector machine

## **Perception Pipeline**

1. Outlier removal: Outlier removal was performed to remove noise in the point cloud (manifested as randomly located points not confined to object boundaries). See Figure1. Points which were farther than 1 standard deviation from the mean distance to k-nearest neighbors were considered as outliers (k = 50)

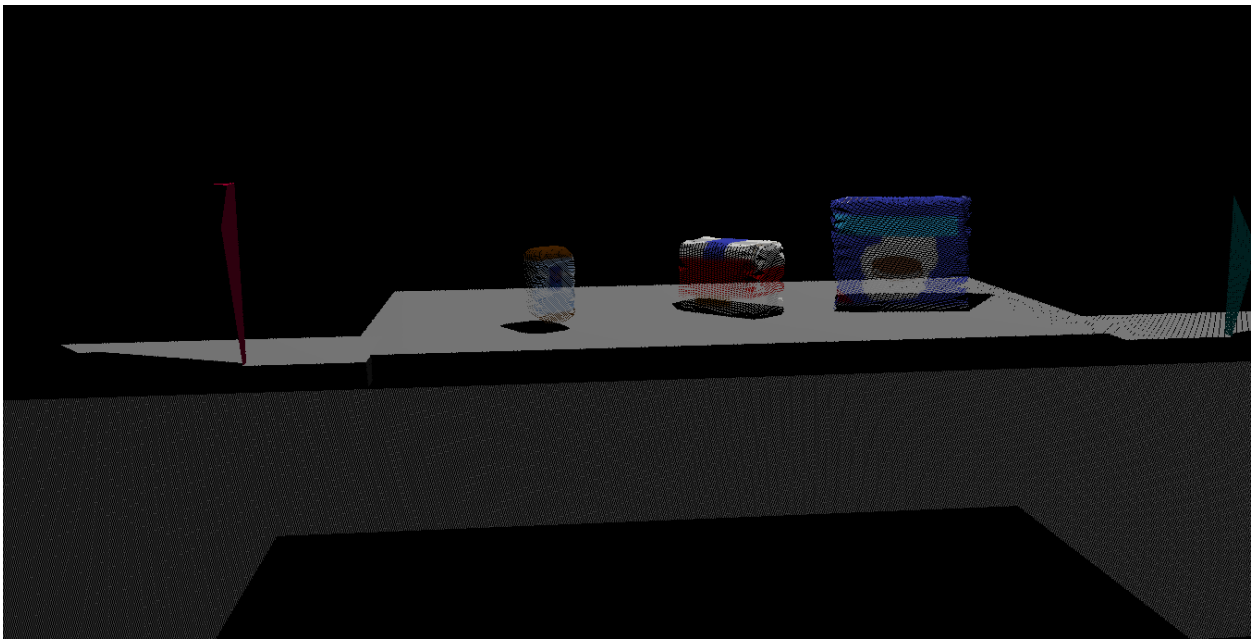

Figure 1 Filtered Point Cloud displaying 3 objects (biscuits, soap and soap2) in the 1$^{st}$ pick list

2. Voxel based downsampling: Voxel based downsampling was performed to reduce the computational load. A voxel size of 0.01m was used to downsample the point cloud (See Figure 2)

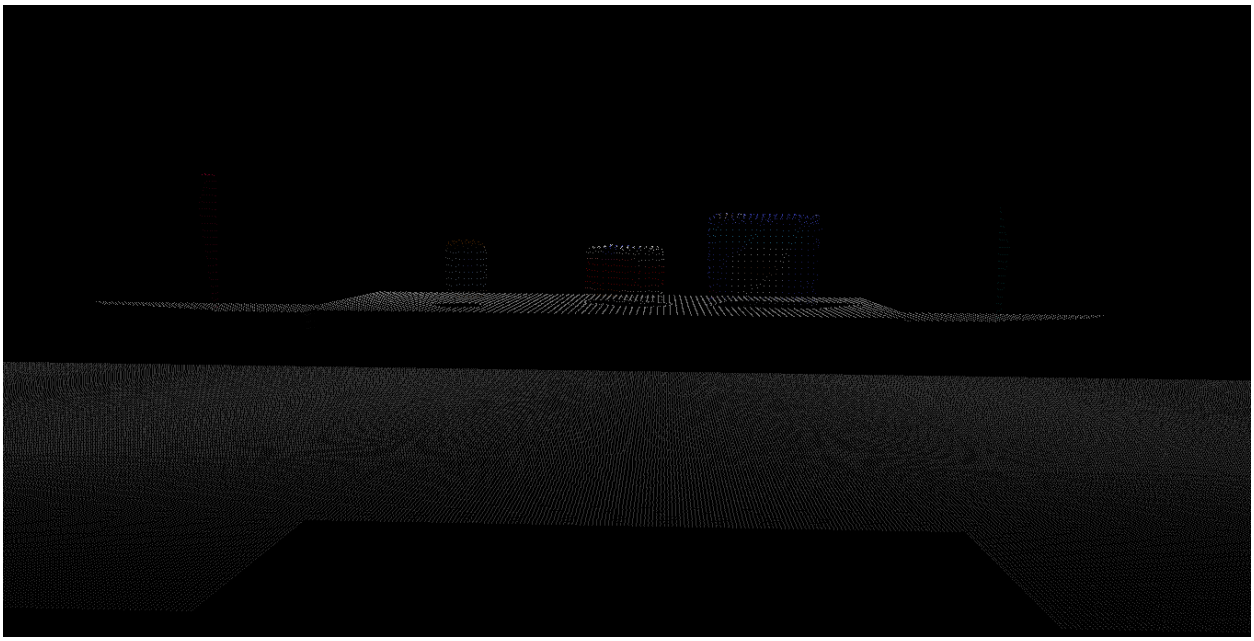

Figure 2. Voxel based downsampled point cloud (leaf size of 0.01 meters). Note that this is sparser than the point cloud in Figure 1

3. Passthrough filtering: A passthrough filter was applied to extract only those point cloud points which represented the tabletop and the objects lying on the table. Cropping parameters (min = 0.5m and max = 1m) were arrived at by trial and error. (see Figure 3) Additional cropping was performed along the Y axis to remove additional point clouds at the left and right end of the table (not shown)

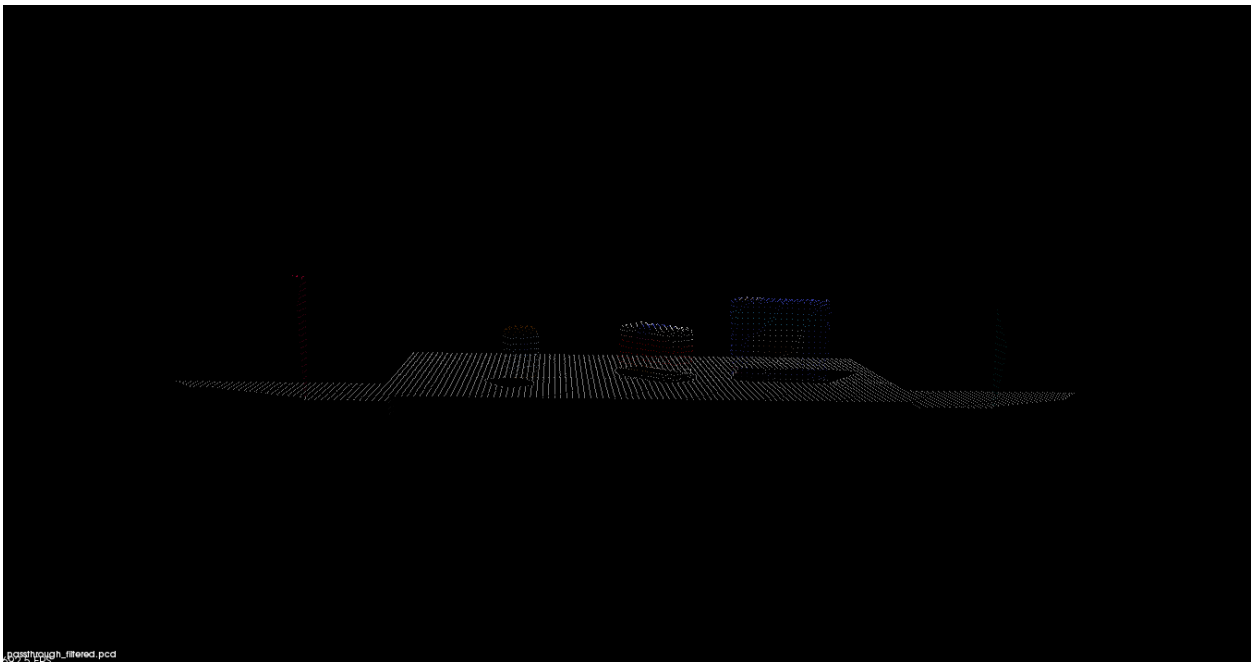


Figure 3. Pass through filtered point cloud with regions above and below removed (min = 0.5m , max = 1m).

4. RANSAC filtering was applied to fit the tabletop shape. The indices of the point cloud corresponding to the tabletop were removed. The remaining points mostly correspond to the objects on the tabletop. Max distance parameters were arrived at by trial and error

Figure 4. RANSAC filtered image with indices corresponding to the table object excluded. Max distance was set to 0.02m

5. Eucledian clustering: For each cluster in the cluster list obtained by Eucledian clustering, color and orientation histograms were calculated and combined to generate a feature vector. The feature vector was then passed onto a pretrained SVM for classification. The obtained labels and corresponding clouds were stored in a *detected_objects* list.

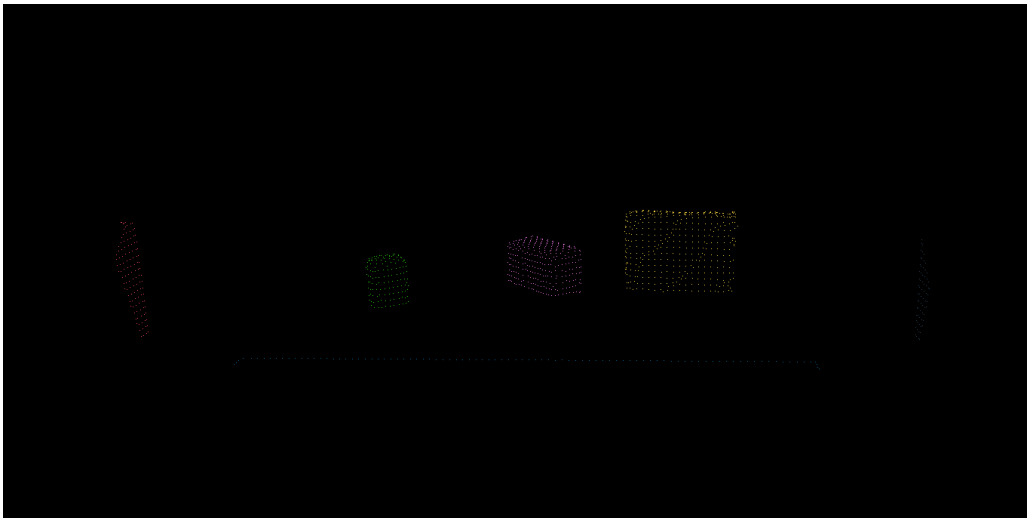

Figure 5. Image of color coded objects is depicted. Each color code represents a particular object in the point cloud. Density based clustering was performed with max cluster size = 10000, cluster tolerance = 0.025

## SVM Training

A gazebo world with a sensor stick in it was launched and *capture_features.py* was run to spawn different objects, measure their features vectors corresponding to the point cloud. The calculated feature vectors were stored in *training_set.sav*. The *train_svm.py* was run to learn SVM parameters from the training set and the learned function was stored in *model.sav*. The parameters were modified until a high enough training score was not obtained (Figure 6).

Modifications were made to the *capture_features.py* in *sensor_stick* package. They are as follows:

1. Number of random orientations for each object was set to 20 to allow a sufficient number of feature vectors examples to train on
2. The models list was modified to include all the items in the 3 pick lists
3. For computing color histogram, *using_hsv* option was set to True

Modifications were made to *features.py*. They are as follows:

1. *compute_color_histogram():* The color histogram was calculated with 32 bins per channel in the HSV color space.
2. *compute_orientation_histogram():* The orientation histogram was calculated with 32 bins per direction i.e. X, Y and Z.

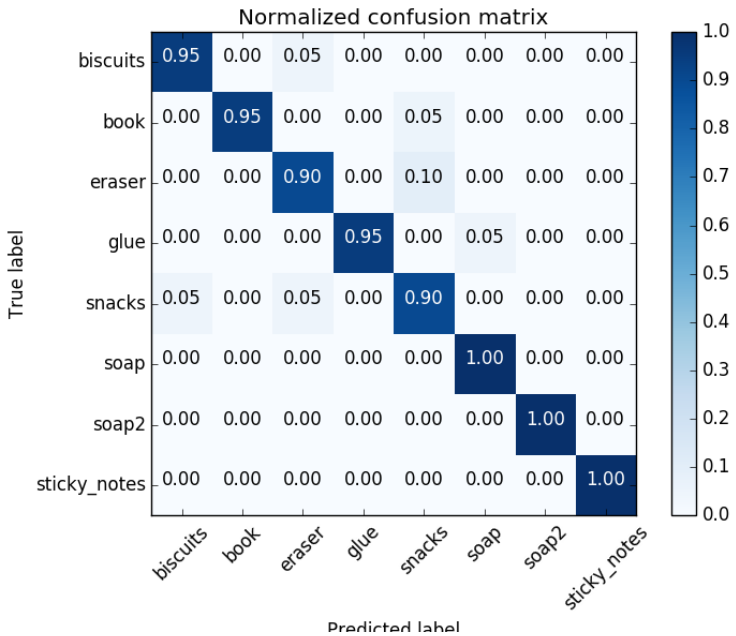


Figure 6. The confusion matrix showing the training score. It is evident that the misclassification rate is less than 5% in all cases. This indicates that the SVM has robustly learned the decision boundaries.

## Results

The trained SVM was run inside the *pcl_callback()* method. In each scene, the trained svm outputted a label corresponding to each object in the scene and a request message was generated for pick and place

operation for each object. The request messages were stored in .yaml output files. These have been provided in the project folder. The results are summarized in the table below and corresponding snapshots of recognized objects are also provided (Fig. 7a, b and c)

| **Scene name** | **Correct/Total** | **Success Rate** |
|---|---|---|
| World1 | 3/3 | 100% |
| World2 | 4/5 | 80% |
| World3 | 8/8 | 100% |

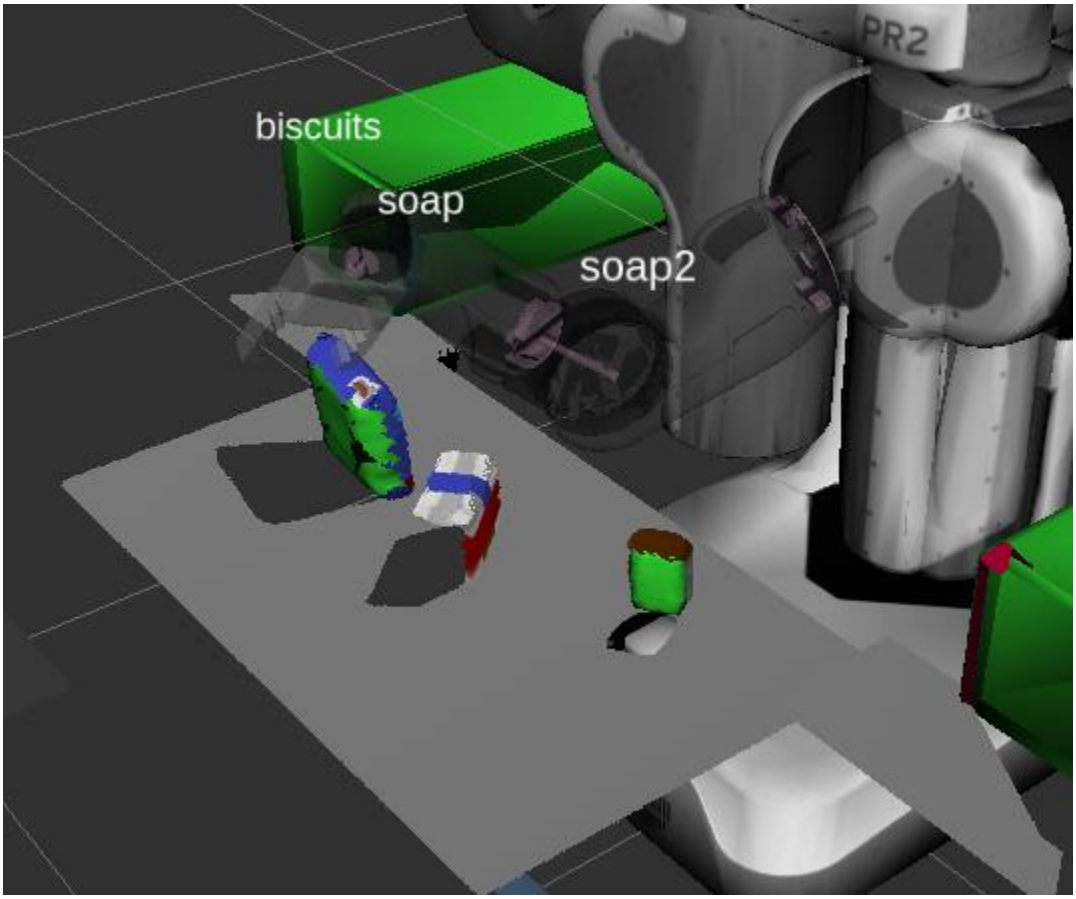


Figure 7a. Image showing 3/3 recognized objects in test world 1


Figure 7b. Image showing 4/5 correctly recognized objects in test world 2. There is an additional classification of biscuit in the top left corner.

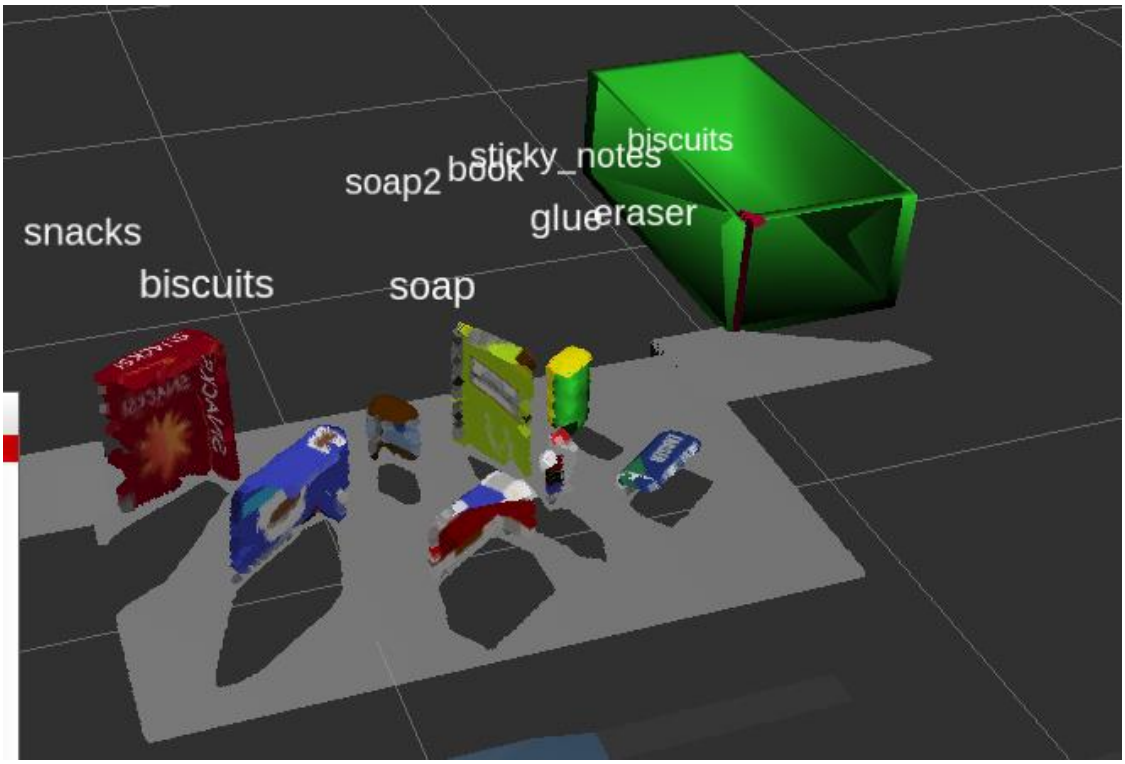


Figure 7c. Image showing 8/8 correctly recognized objects in test world 3. There is an additional false position corresponding to the biscuit in the top right corner.