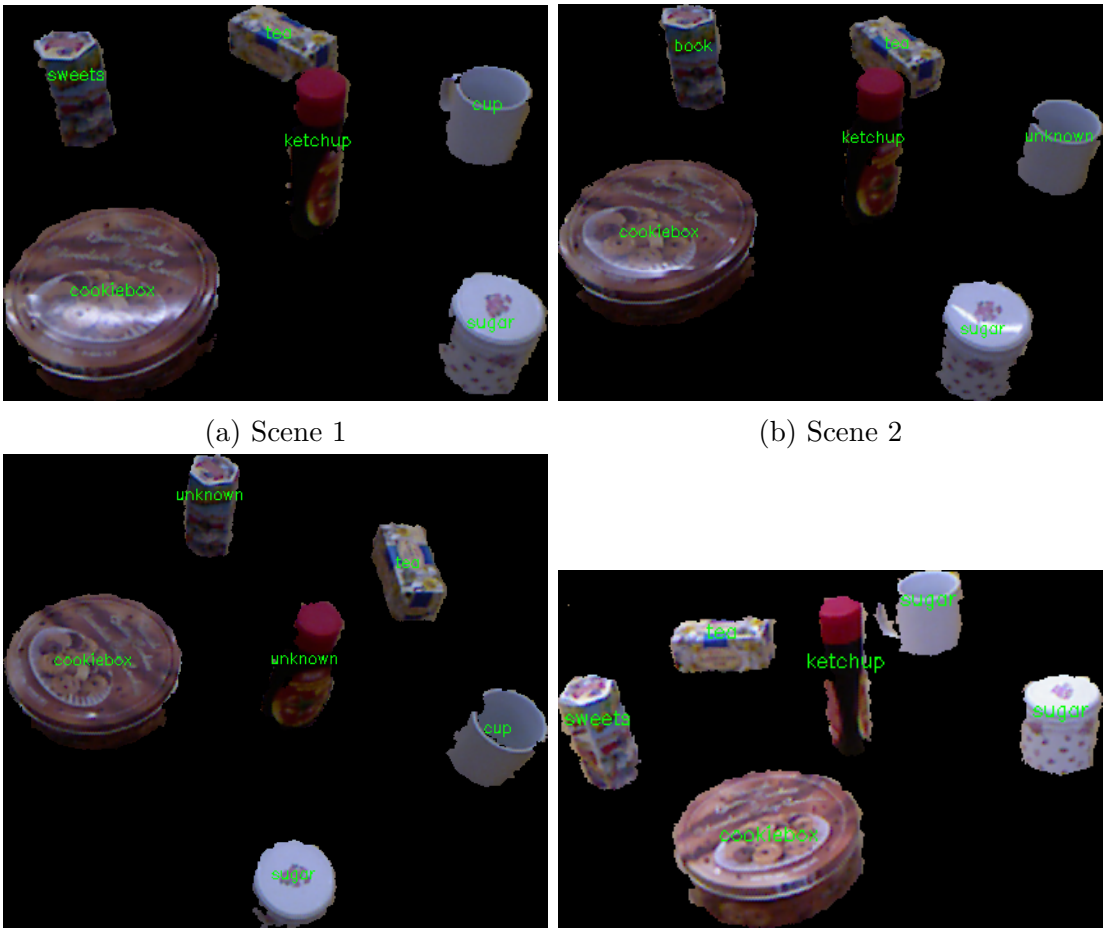
Exercise 6: 3D Object classification

Max Tamussino, 01611815

January 15, 2021

Contents

1 Results



(a) Scene 1

(b) Scene 2

(c) Scene 3

(d) Scene 4

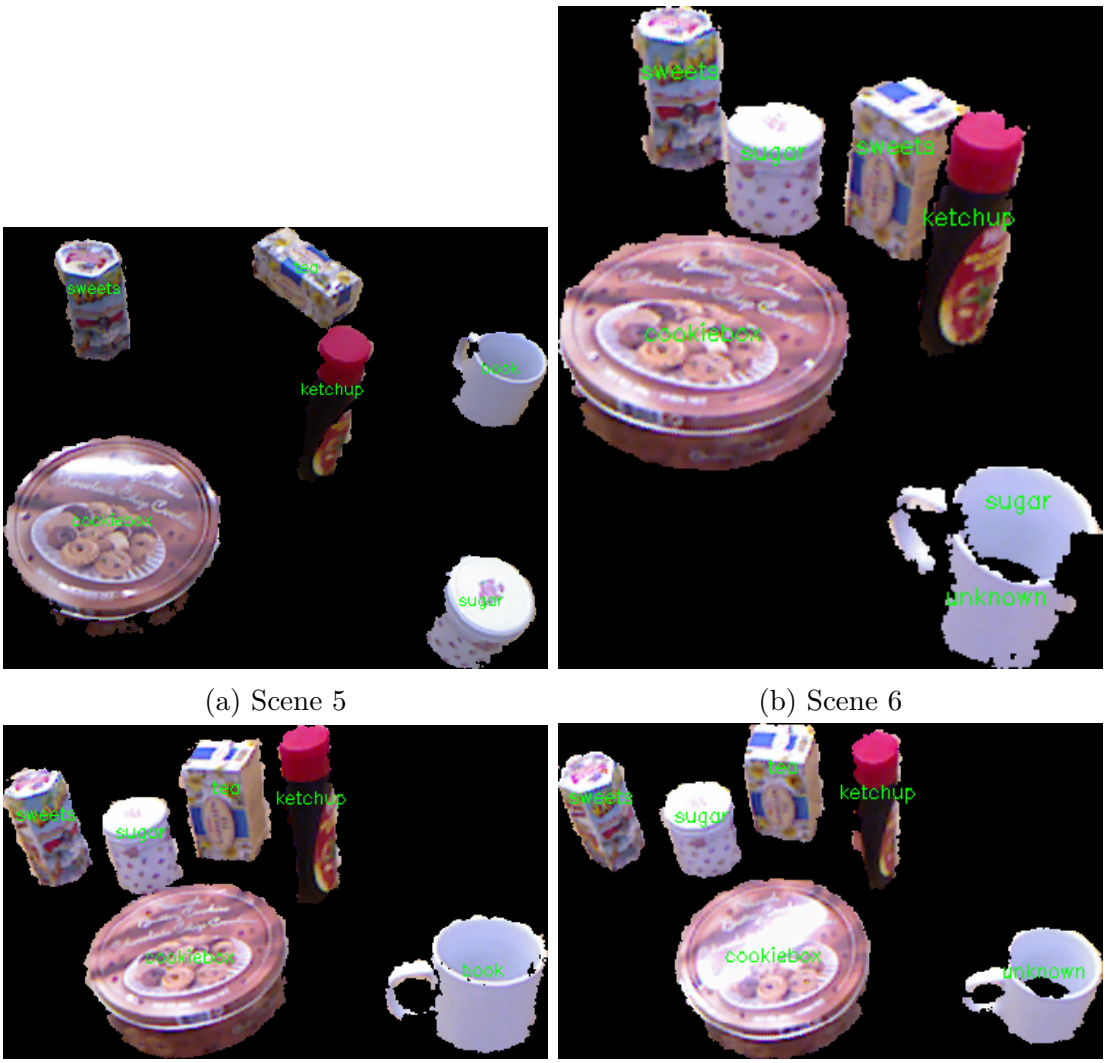
Figure 1: Result of the implemented 3D object recognition for scenes 1 to 4

2 Approach

To achieve the results given in the previous section, multiple steps were taken. This section describes the steps of the chosen approach. The input is a 3D pointcloud. Scene 9 (see Figure 3a) is used to demonstrate the individual steps.

2.1 Dominant plane removal

To find the dominant plane, the table the objects are standing on, the RANSAC algorithm was applied. The inliers of this plane are then removed from the pointcloud. This colored scene pointcloud is projected to 2D for the SIFT matching discussed in following sections. The annotated version of this image is depicted in Figure 2d.



(a) Scene 5

(b) Scene 6

(c) Scene 7

(d) Scene 8

Figure 2: Result of the implemented 3D object recognition for scenes 5 to 8

2.2 Clustering

To prepare the pointcloud for the clustering, it is downsampled to reduce computational effort. For this purpose, a voxel grid is used. The clustering algorithm DBSCAN is then applied to the pointcloud. The pointcloud is projected to 2D image space (see `utility.py`), which is shown in **??**.

2.3 Filling

The clustered 2D pointcloud projection is then transformed to continuous object areas. This is done using morphological transformations, in particular the closing operation, which consists of dilation followed by erosion. A 5×5 circular kernel is used, the result is depicted in Figure 4a (see `main.py`, lines 121 and 122). This operation sometimes introduces unwanted additional colours at cluster boarders to the image.

(a) Scene 9 (b) Scene 10

Figure 3: Result of the implemented 3D object recognition for scenes 9 to 10

2.4 Cropping

The resulting labels image is cropped, black borders are removed. This is shown in Figure 4a. The exact same cropping is applied to the scene image, to keep matching dimensions. This is done to reduce computational effort for feature matching. The cropping function is implemented in `utility.py`.

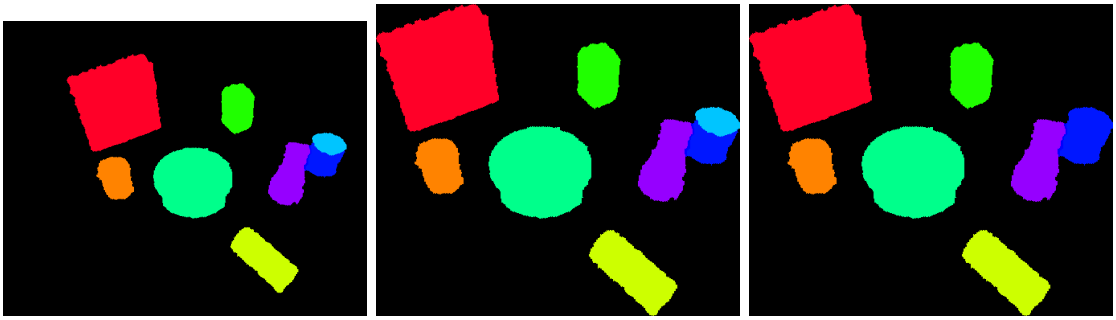
2.5 Cluster merging

Because objects like cups are often clustered into two separate objects in 3D, an additional steps needs to be done to join those clusters. Firstly, for each cluster, the border of its area is calculated using the morphological gradient. It is then checked whether the border area contains the label colours of other adjacent objects. If so, a colour histogram of those objects is created and then compared. The distance between colour histograms is calculated by the maximum of actual colour difference and a scaled difference of the number of their appearances. If the colour histogram is similar, the clusters are merged. This is implemented in `merge_clusters.py`.

By comparing Figure 4b to Figure 4c, the effect can be seen - the purple cluster is not merged with the blue ones, because the colour histograms of the ketchup bottle and the cup are very different. In Figure 2b, the colour histograms of the two cup parts were too different and did not merge as expected.

2.6 Object comparison

This step is done for every training pointcloud available. Every pointcloud has a known class name.



(a) Filling (b) Cropping (c) Cluster merging

Figure 4: Result of filling, cropping and cluster merging in scene 9

2.6.1 Projection

The training pointcloud is projected to 2D and the resulting image is cropped to reduce computational effort. Black borders are removed in this step.

2.6.2 RGB SIFT matching

The training image is matched to the scene image using SIFT features, repeating the process for every RGB colour channel (see `sift_matching.py`), greatly increasing detection performance however also increasing computation time (see Section 3). For every keypoint, the best two matches are calculated. The better match of those is considered, if its distance is smaller than 0.7 times the distance of the second match - filtering out bad matches. The coordinates of all considered matches in the scene image are determined.

2.6.3 Object hypothesis scoring

This step is done for every previously determined match coordinate. If the match coordinate lies within a labelled cluster area, the label colour of this area is examined. If there already is a hypothesis for this label colour, the current score is increased. The value by which it is increased is determined by the inverse of the available matches for this object, to prevent higher detection probabilities for larger training images. If there is no hypothesis for this colour, the validity of the colour has to be checked - morphological transformations may introduce unwanted colours (clusters). A valid colour then leads to a new entry in the object hypothesis list, being classified as *unknown*. The implementation can be found in `categorise_matches.py`.

2.7 Object hypothesis update

After checking all matches, it is determined if any current score is higher than the previously highest score. If so, the current training image's class is used as the new hypothesis for this label color - but only if more than two matches lead to this decision, avoiding false positive detections based on little information. This particular function may be found in `categorise_matches.py` on lines 83-87.

3 Discussion

Generally, the proposed method does not assume only one single instance of each class in a scene. This leads to a certain disadvantage for the measured detection rate, as this assumption would rule out most object classes for the few badly classified objects in a scene, which could lead to a correct guess for those in most cases. This assumption was omitted as in real scenes, there could be many instances of one object class in a scene (eg. multiple cups on a table). The proposed method moreover leads to only few false positives, marking badly recognised objects as *unknown*. This is desirable, as it makes clear where information is missing and where the object was correctly recognised. There are 6 false positives in a total of 60 objects.

3.1 Detection performance

The detection of objects varies slightly between different tries. Therefore, for the detection performance experiments, the best result out of five tries was used.

The proposed method for 3D object detection classifies 83.3 % of the given objects correctly, which is shown in more detail in Table 1. Because the clustering and labelling of the different objects worked correctly in most of the scenes (except for the cluster merging failure in scene 6, see Figure 2b), the issue occurs during SIFT feature matching. Large and texturous objects like the book, the cookie box and the ketchup bottle are detected correctly more often (even without RGB SIFT), because the SIFT feature matching works better with many available keypoints. Especially low texture objects like the cup are however very badly detected.

The effect of using all RGB color channels for SIFT matching can also be obtained from Table 1. The detection performance shows a significant decrease when this feature is disabled, lowering the overall detection rate to 45.0 %. This experiment yielded many objects being detected as *unknown*, indicating no strong enough object hypothesis. This occurs due to the the set minimum of three matches for a valid hypothesis. The experiment was repeated without this threshold, showing false results instead of classifying objects as unknown - this was not considered to be a more desirable result.

Class	Book	Cookiebox	Cup	Ketchup	Sugar	Sweets	Tea	Total
Appearances	2	10	8	10	10	10	10	60
Correct RGB	2	10	3	9	9	8	9	50
Correct Grey	2	8	0	5	4	5	3	27
Ratio RGB	100 %	100 %	37.5 %	90 %	90 %	80 %	90 %	83.3 %
Ratio Grey	100 %	80 %	0 %	50 %	40 %	50 %	30 %	45.0 %

Table 1: Correct detections of object classes when present, comparing RGB SIFT with simple greyscale SIFT

3.2 Execution time

The execution time for all significant parts of the algorithm was measured. The mean value of measurements for every given scene was used. It was examined, which influence image cropping and RGB SIFT have on these times. The results are displayed in Table 2. It is clear that image cropping significantly reduces the time needed for 2D cluster merging and SIFT matching. The average total time per scene was reduced from 35.7 s to 21.2 s. Using RGB color SIFT matching, although it greatly increased detection performance, increased the execution time (without image cropping) to 69.9 s. This is due to the high average execution time for SIFT matching of 376 ms per training pointcloud. Combined, the two features lead to a lower than standard average execution time of 34.6 s per scene.

Task	Mean execution time per scene [ms]			
	Standard	Cropping	RGB SIFT	Both
Plane fitting	4549.1	4369.0	4606.4	4333.5
3D clustering	162.3	156.8	153.6	153.8
2D cluster merging	8788.5	4248.1	10821.0	4248.3
SIFT (per pointcloud)	153.3	85.3	376.0	180.2
Total	35713.0	21211.9	69875.8	34853.6

Table 2: Mean execution time comparison of proposed method, using the mean of all provided scenes