376.054 Machine Vision and Cognitive Robotics Open Challenge: 3D Object Recognition

Severin Jäger, 01613004

January 11, 2021

Contents

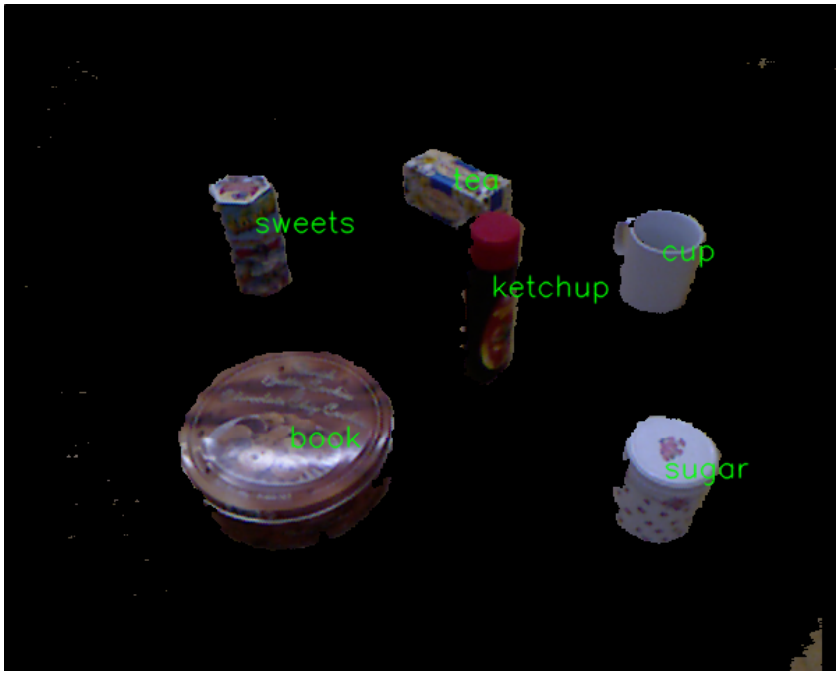


Figure 1: Recognition results for point cloud 0

1 Recognition Results

The recognition results of the implemented pipeline are shown in Figures 1 to 10. It is apparent that the detection quality depends on the perspective of the input RGBD image. Additionally, the results are not deterministic, thus they vary in case of rerunning the whole algorithm.

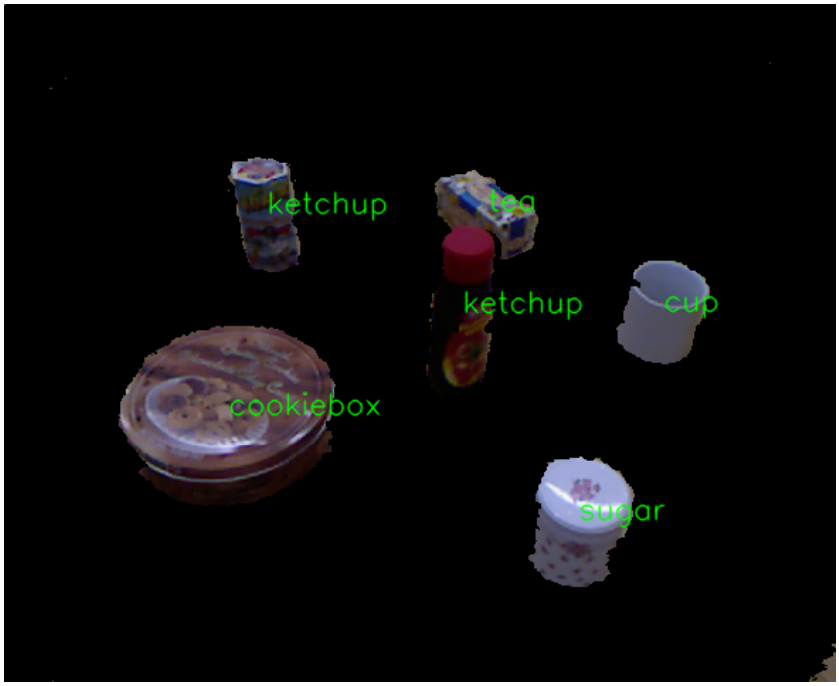


Figure 2: Recognition results for point cloud 1

Figure 3: Recognition results for point cloud 2

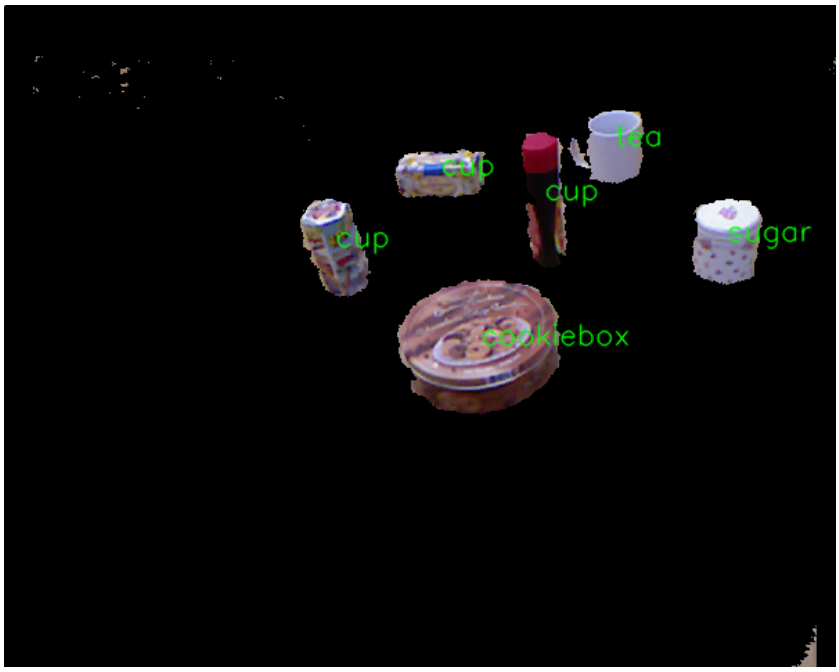


Figure 4: Recognition results for point cloud 3

Figure 5: Recognition results for point cloud 4

Figure 6: Recognition results for point cloud 5

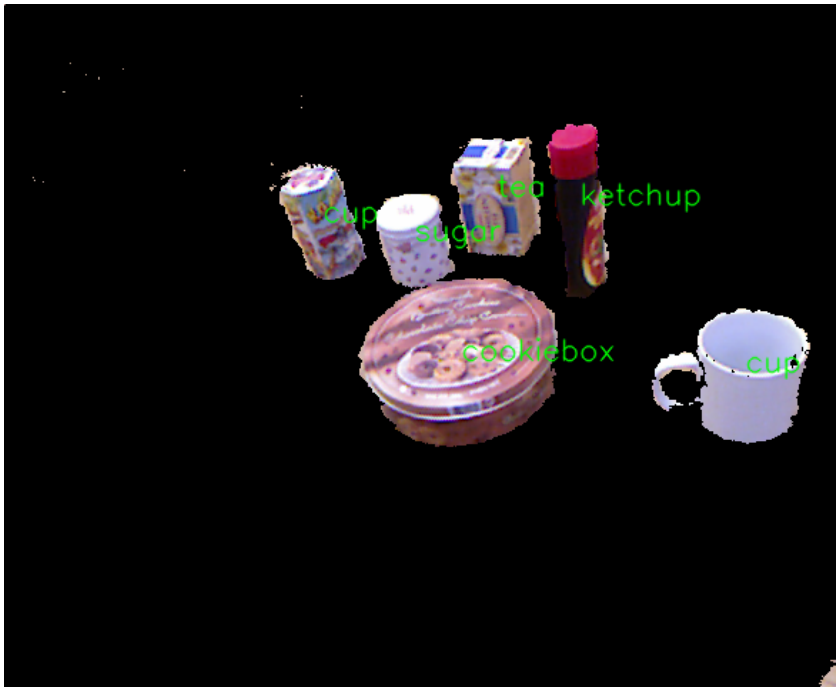


Figure 7: Recognition results for point cloud 6

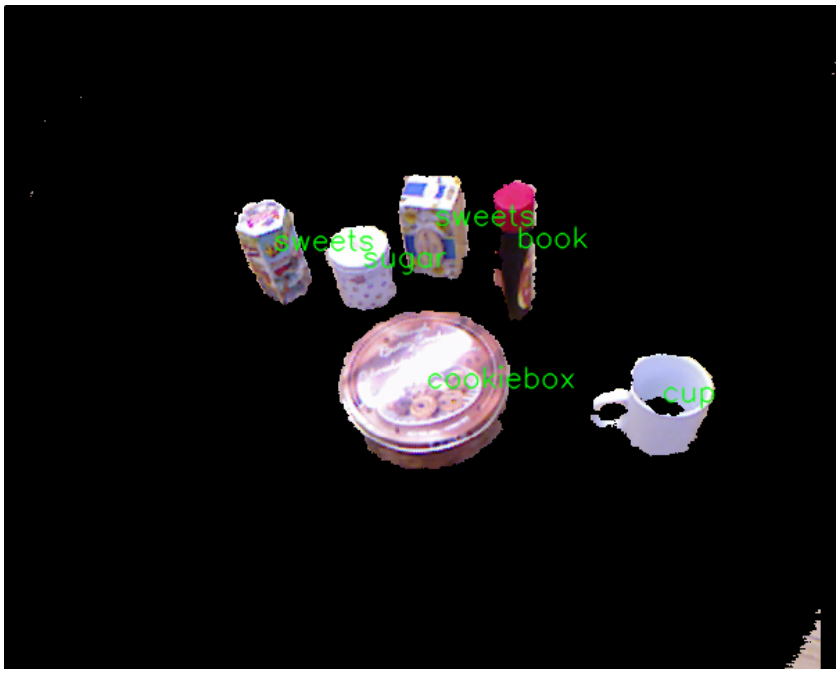


Figure 8: Recognition results for point cloud 7

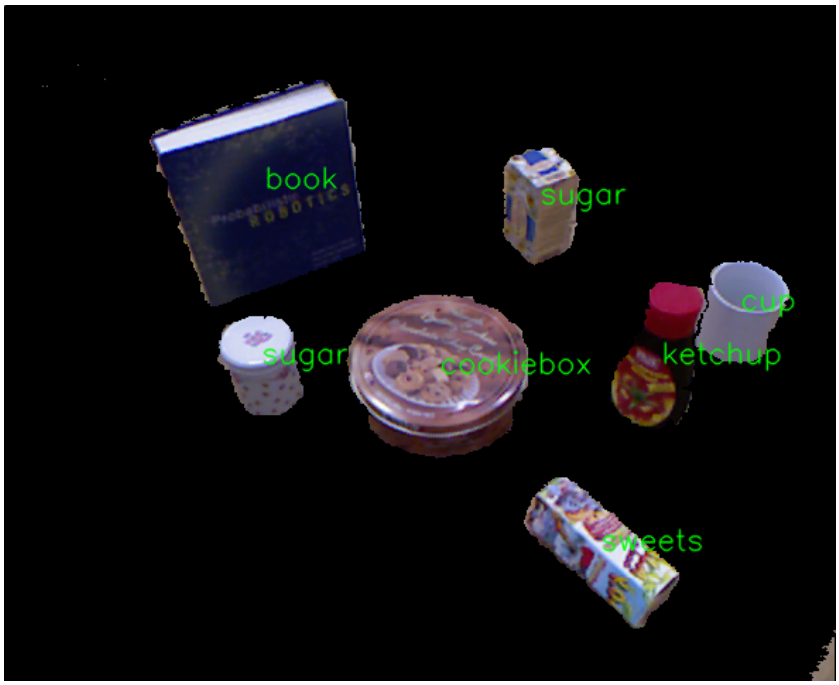


Figure 9: Recognition results for point cloud 8

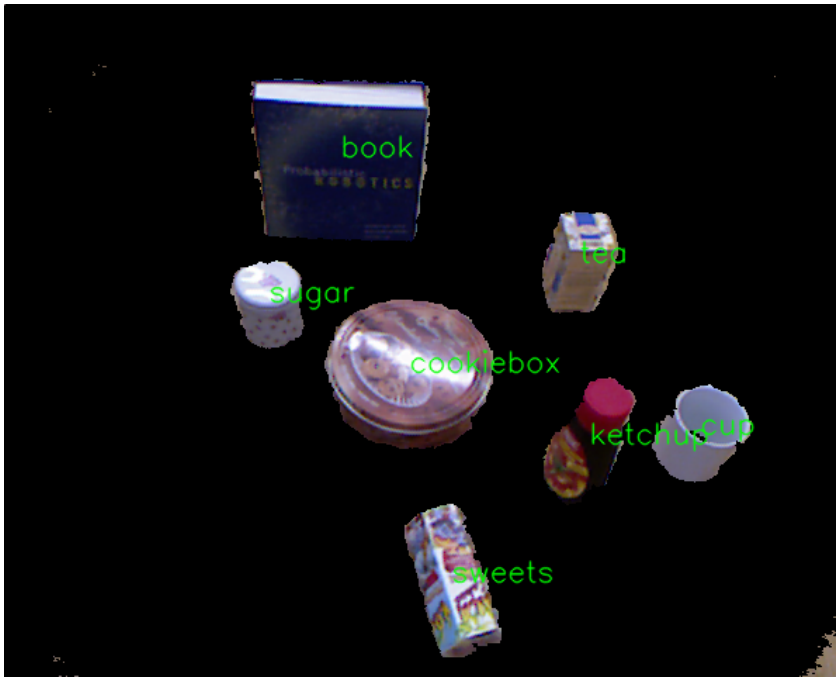


Figure 10: Recognition results for point cloud 9

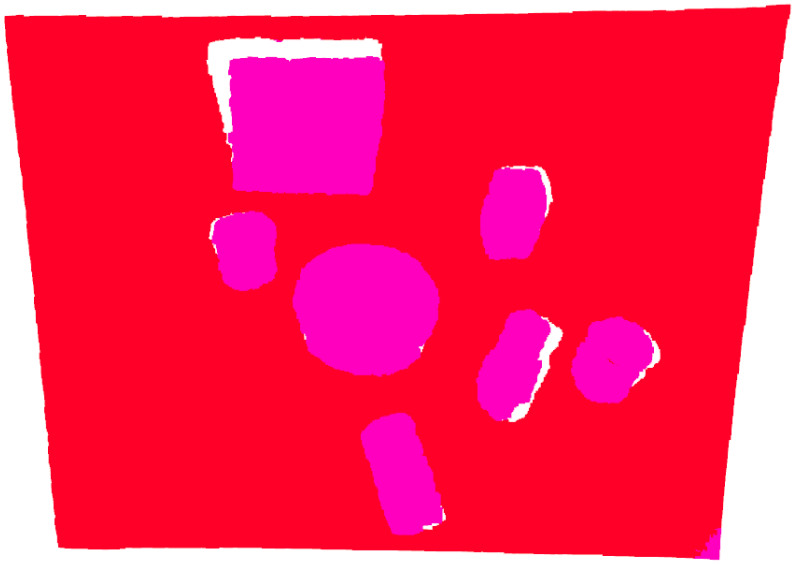


Figure 11: Detection of the ground plane with RANSAC

2 Discussion of the Approach

In this exercise, a 3D recognition pipeline was implemented. In the following, all relevant steps are discussed. The relevant code can be found in the `main.py` file which is structured in the same manner as this section. However, some relevant functions are located in the files `cluster_matching.py` and `points_to_image.py`.

2.1 Ground Plane Detection

In the beginning, the ground plane (in case of the test images this was the floor) was detected an removed. This simplifies the clustering steps in the following significantly, at least as long as the object are placed with a reasonable distance. As discussed in Exercise 5, RANSAC was used to detect the plane, however the Open3d method `segment_plane` was used instead of a hand-crafted approach. Figure 11 shows the result of the plane detection in the point cloud.

The respective code is located in the main method starting from line 46 and offers the following parameters:

- `ransac_inlier_dist`: The maximal distance towards the estimated plane within which a point is considered as inlier. The chosen value is 1.5 cm.
- `ransac_iterations`: The number of RANSAC iterations, thus the number of random samples. A value of 7000 was used.

In the following, all points belonging to the dominant plane are removed, thus only the points belonging to the objects of interest and some artefacts remain.

Figure 12: Image with removed ground plane

2.2 Point Cloud to Image Conversion

In this step, the previously described point cloud is converted into an image. Therefore, the camera geometry has to be considered. In homogeneous coordinates, the pixel position p can be calculated from the camera coordinates ${}^{C}p$ as

$$p = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K\,{}^{C}p = K \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{1}$$

with the perspective projection matrix

$$K = \begin{pmatrix} f_{x,rgb} & 0 & c_{x,rgb} & 0 \\ 0 & f_{y,rgb} & c_{y,rgb} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2}$$

The resulting pixel coordinates (u, v) have to be unwrapped (as they might be negative) and rounded, then the can be used to create the image. A possible result is shown in Figure 12.

The respective code can be found in the main method starting from line 61 and in the file `points_to_image.py`.

Figure 13: Detected clusters in the point cloud

2.3 Clustering

This step deals with the segmentation of the image which is achieved by the means of clustering of the point cloud from Section 2.1. As clustering is a computationally expensive task, the point cloud is down-sampled. This is done with voxels. In the following, the Open3d implementation of the DBSCAN clustering is applied to create the cluster labels. One resulting clustered point cloud is visualised in Figure 13.

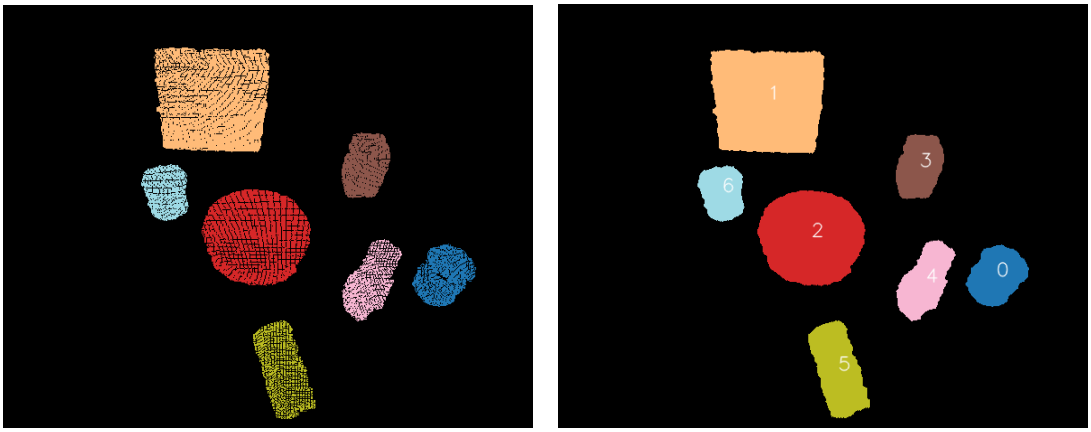
As the clustering parameters (s. below for details) that achieved the best results tends to find small clusters that belong to the floor (like the one in the right of Figure 13), it is necessary to remove very small clusters before proceeding. Therefore, simple threshold is used.

The cluster encoding is achieved by the means of colouring the respective points with a colourmap. So the colour value is `colourmap(label/max_label)`.

The clustering code can be found in the main file starting from line 66. It features the following parameters:

- `dbscan_eps`: The parameter `eps` of the Open3d DBSCAN clustering algorithm implementation. The used value is 0.02.
- `dbscan_min_points`: The parameter `min_points` of the Open3d DBSCAN clustering algorithm implementation. The used value is 50.
- `cluster_min_points`: Clusters with less points are removed. The used value is 300.
- `voxel_size`: The voxel size for the down-sampling algorithm. The used value is 0.002.

Unfortunately, these parameters are not able to achieve optimal clustering in all cases. As shown in Figures 3 and 6, the cup is segmented as two objects. This is not surprising, as from some perspectives it looks like two objects in the point cloud. This problem can be mitigated by increasing `dbscan_eps`, however this is unsatisfactory as it leads to two distinct objects being detected as one in other point clouds. Thus there is still room for improved clustering approaches.



(a) (b)

Figure 14: Clusters projected to an image before (a) and after (b) the morphological transformation

2.4 Clusters to Image Conversion

In this step, the detected clusters are mapped to the 2D image space in order to segment the image created in Section 2.2. Again, the functions from `points_to_image.py` are used. As the point cloud was down-sampled before clustering, the resulting image shows numerous holes (s. Figure 13 (a)). To overcome this problem, a simple morphological transformation filling the holes is applied. This results in images like the one displayed in Figure 13 (b).

The respective code is located in the main method starting at line 113. Its only parameter is `kernel_size`, which describes the size of the kernel used for the morphological transformation.

2.5 Feature Matching

During the previously described steps, a suitable segmentation of the input image was achieved. In the following, the actual classification of the objects can be performed. Similar to Exercise 3, SIFT is used for matching the features in training images to the ones in the scene. This is done in the following way:

For each object class, there are several training point clouds. The following steps are performed for each of them. In the meantime, a data structure consisting of the current class estimation and the related metric η (s. below) is kept The initially, η is 0.

1. The point cloud is projected to a 2D image using the `points_to_image.py` functions.
2. SIFT is used to calculate key points and descriptors in the training image.
3. The training features are matched to previously calculated SIFT features in the scene (s. Figure 15).
4. For each cluster, the following steps are performed:

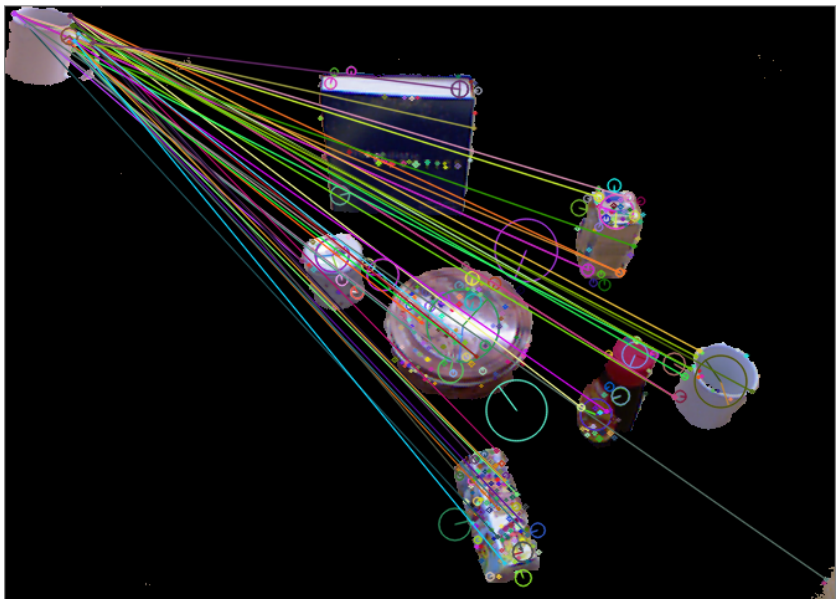


Figure 15: Matching of training and scene features

(a) The number of matches related to key points in this cluster $n_{cluster}$ are counted.

(b) The ratio between the matches in the cluster and the overall matches is calculated as $r = n_{cluster}/n_{total}$. This solves problems arising from different training images with different numbers of features.

(c) In case $n_{cluster} > n_{min}$ and $r > \eta$, η is set to r and the current class hypothesis is updated.

Eventually, there is a hypothesis for each cluster. However, it is possible that no suitable cluster was found. In this case, the estimated class is *unknown*.

The respective code is in located in the main method starting at line 132. The methods used to map the matches to a cluster are located in the `cluster_matching.py` file. It offers the following parameters:

- `sift_threshold`: Denotes the maximal distance of a match for the match to be considered. This helps by reducing the computational effort by removing less relevant matches and is set to 400.
- `min_matches`: The parameter n_{min}. At least this number of matches related to a cluster is necessary to create a new hypothesis. This prevents high values of η due to a very small number of matches. A value of 4 is used.
- `considered_matches`: Specifies the number of matches for each training key point considered. A value of 3 is used.
- `neighbourhood_size`: The size of the surrounding of a cluster in which a match is still considered an inlier in pixels. This ensures that features at the edge of a cluster (which might lie just outside) are still considered. A value of 3 is used.

All Figures in Section 1 were created with this basic variant of the algorithm. However, two modifications were implemented, they are discussed in the following two sections.

Figure 16: Recognition results for point cloud 2 with RGB SIFT

2.5.1 Weighted match counting

So far, the matches related to each cluster were counted. However, similar to improved variants of RANSAC (like MSAC) it might be useful to consider the actual distance d of the match and use it as weight. This is done by redefining the metric η in the following way:

$$\eta = \sum_{i=0}^{n_{cluster}} \frac{1}{{d_i}^2}.$$

Unfortunately, this hardly improves the recognition results. Nonetheless, it can be activated by setting the parameter `use_distance_weights` in the main method.

2.5.2 RGB SIFT

Another approach that was used to improve the matching quality was extending the SIFT features to the RGB colour space. Thus, not only the grey version of the scene and the training image was used, but three images in parallel. As a result, the number of matches is drastically increased. This does not only lead to an increased computational complexity, but also to better detection results. For instance, the detection results of point cloud 2 (compare Figure 3 with 16) could be improved notably.

This extension can be enabled by setting the parameter `use_colour_sift` in the main method.

Object	Correct	False negative	False positive
Book	2	0	2
Cookie Box	9	1	0
Cup	8	3	4
Ketchup	8	2	4
Sugar	10	0	3
Sweets	5	5	1
Tea	6	4	1
Total	48	15	15

Table 1: Recognition results with the basic pipeline

Object	Correct	False negative	False positive
Book	2	0	0
Cookie Box	10	0	0
Cup	8	4	5
Ketchup	8	2	2
Sugar	10	0	5
Sweets	9	1	0
Tea	6	4	0
Total	53	11	12

Table 2: Recognition results with the RGB SIFT

3 Evaluation

An overview of the recognition results of the basic pipeline is given in Table 1. The algorithm is able to classify certain objects (like the cookie box or sugar) very well, while struggling with others (like sweets of tea). Reasons for this behaviour might be the texture of the objects. For instance, the cookie box or the book offer clear textures, which can rather easily be distinguished from others. In contrast, the textures of tea, ketchup or sweets are rather fuzzy. Nonetheless, the cup can be detected rather well, despite not showing and distinctive texture at all.

As the weighted match counting approach as described in Section 2.5.1 did not change the recognition performance beyond the deviation between consecutive attempts of the same approach, it was considered unsuccessful and therefore not treated in detail in this report.

The RGB SIFT approach (as described in Section 2.5.2) turned out to be a helpful extension. As shown in Table 2, it increases the recognition accuracy from 77.5 % to 85.5 %. In particular, the detection of sweets was drastically improved. With the basic pipeline, they were frequently classified as ketchup, due to the additional colour information this does not happen any longer. However, the recognition system still suffers from notable problems in distinguishing a cup from sugar. This might be due to the similar colour of the objects and the fact that both do not offer a lot of texture.

Task	Runtime (standard) [ms]	Runtime (RGB SIFT) [ms]
RANSAC	4606	4634
2D Projection (Image)	11	11
Clustering	181	186
2D Projection (Clusters)	32	36
Cluster Classification	12378	29959
Overall	17180	34862

Table 3: Timing of the individual pipeline stages

Task (per training image)	Runtime (standard) [ms]	Runtime (RGB SIFT) [ms]
SIFT Scene	42	146
SIFT Training	38	55
Feature Matching	3	6
Classification	208	437

Table 4: Timing of the steps of the classification stage

The runtimes of the individual steps of the pipeline (both the basic and the RGB SIFT) can be found in Table 3. It becomes apparent, that the RANSAC and the final Classification step are the most time-consuming ones. While the runtime of RANSAC only depends on the number of points and the desired accuracy, the classification step takes longer as the number of object classes and thus training images increases. Furthermore, the effort of the classification step significantly increases as the RGB SIFT approach is used. Some details are depicted in Table 4. As expected, the runtime for the SIFT interest point detection and descriptor calculation is notably increased. As this happens in a loop over all training images, the overall runtime of the classification step deteriorates.