Exercise 6

In the final exercise you will plug different modules and functions you programmed in the previous exercises together to form a complete 3D object recognition pipeline. As this exercise is designed as an *open challenge*, you will not be provided with a code construct or function headers and it's up to you how to implement the algorithm, which will be explained in the following sections.

Dataset

Download files

The 7 objects represented in the dataset are:

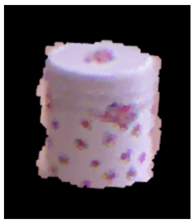
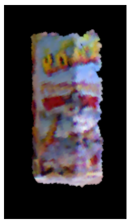
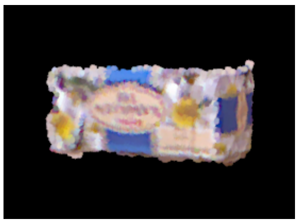
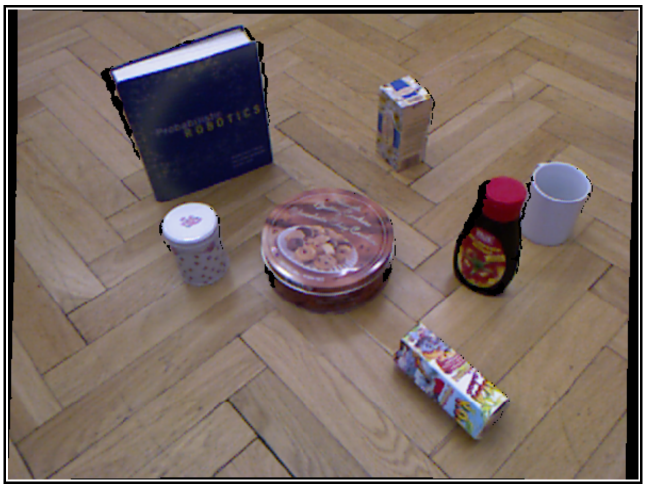
Book	Cup	Cookiebox	Ketchup
			
Sugar bowl	Sweets	Tea	

Figure 1: The 7 different objects to be recognized.

The dataset of 3D point clouds is divided into three sets, namely **training**, **test** and **groundtruth**. The training set contains point cloud files in which the respective object has already been cut out by the clustering algorithm. For each object, many different point clouds are available, such that almost all viewpoints of the objects should be captured. The test set consists of 10 different point clouds containing a setup of the objects. In these point clouds the object recognition algorithm should correctly detect all objects. The groundtruth set is actually not necessary for this exercise, but it might be used to check the recognition results, as it contains the correctly recognized and cut out objects for each test image.

Examples of how to load a point cloud from a file can be seen in the main scripts of the previous exercise. An example of a pointcloud of the test set is shown in Figure 2a.

Also included is a file with parameters for the camera `camera_params.py` used to generate the images of objects on the floor. These parameters are needed to project the 3D pointclouds to 2D images. You only need to use the focal lengths `fx_rgb` , `fy_rgb` , and the coordinates of the optical center `cx_rgb` , `cy_rgb` .



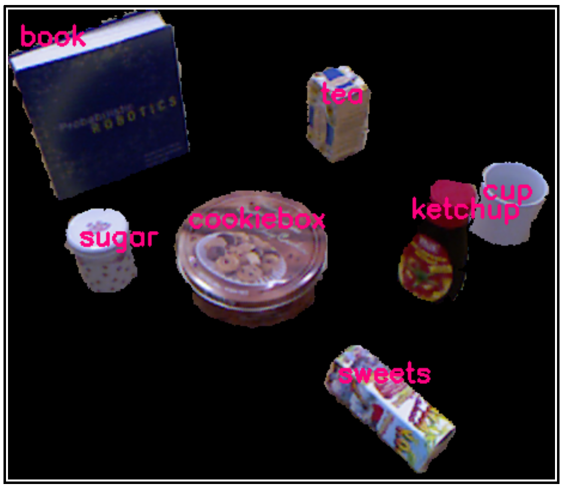


Figure 2a (left): Projected input pointcloud in which objects should be detected.
Figure 2b (right): Possible result of the object recognition pipeline. Each remaining cluster is labelled with the most likely object name

Algorithm (12 points)

The object recognition algorithm works on the remaining pointcloud after removing the ground plane calculated in the last exercise. The remaining points are clustered using a similar approach as you used to cluster the votes in configuration space in exercise 3. The goal of this final step in the pipeline is to be able to assign each cluster of the input point cloud to the corresponding object class. A correct result can be seen in Figure 2b.

The general idea of the method is again based on the principle of matching SIFT descriptors. Matches will be calculated between a test point cloud and all of the separate training point clouds containing an object. As SIFT is a 2D image feature and cannot directly be used on 3D point clouds, point clouds have to be projected to the 2D image plane first. After the matches to all training point clouds have been calculated, the classification of each object cluster is decided according to which training point cloud had the most matches with the respective cluster.

To calculate the SIFT feature points and corresponding descriptors, the same functions as in

exercise 3 can be used. For the necessary pre-processing steps required by the SIFT calculation, refer to exercise 3 as well.
Here is the (suggested) basic sequence of the algorithm:

1. Apply your RANSAC algorithm from exercise 4 on the test pointcloud to eliminate all points corresponding to the ground or the table plane. If your implementation does not work well, you can also use Open3D's segment_plane function.
2. Project all remaining 3D points to the 2D image space using the given camera parameters (focal length and the coordinates of the optical center). For more theoretical information refer to the OpenCV documentation. Implement the projection yourself and don't use the provided functions of OpenCV. Note: You can access the color of each point of your pointcloud with the property .colors. Be aware, that the ordering of the color channels is RGB while the default OpenCV ordering is BGR. You can change the color channels by calling the function cv2.cvtColor or just by reversing the last dimension of your array by slicing `[..., ::-1]`. Many OpenCV functions require a specific datatype, e.g. the error `incorrect depth (!=CV_8U)` means that your array is not in the required datatype of 8 bit unsigned integer (`np.uint8`) with values in the range of 0-255.
3. In 3D space, cluster the remaining points from step 1 as you did in exercise 3. You should probably subsample the point cloud before to speed up the process. An additional post-processing step after the clustering could be the elimination of clusters which do not contain many points, because these clusters might not correspond to objects but to the ground or table. If your Clustering approach from exercise 3 does not work well for these pointclouds, you can use Open3D's DBSCAN clustering function.
4. Project the resulting **cluster numbers** of the 3D points to the 2D image space. This will result in an "image" like Figure 3 (displayed with colormap jet). If you subsampled before clustering you might end up with holes in the projected objects. These holes can be "filled" by applying a simple filter on the image. You can use one of the morphological operators implemented in OpenCV, for more details check OpenCV Morphological Transformations.

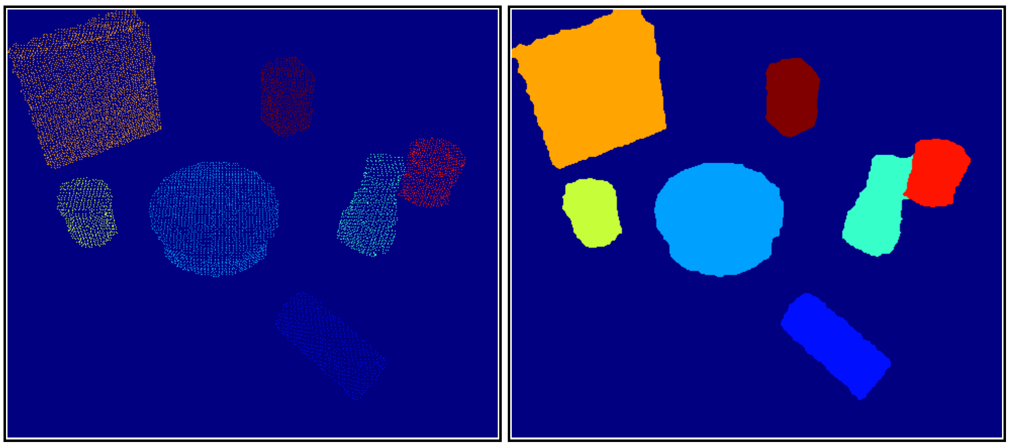


Figure 3: The label image after clustering (left) and after filling the holes (right)

5. Calculate the SIFT descriptors of the scene image. Start a "voting" process with the clusters and training point clouds:

For each training point cloud:

- Project it to 2D image space
- (Maybe crop the projected training image to the object pixels)
- Calculate SIFT descriptors.
- Match the descriptors with the ones calculated for the test image. You can check the main file of exercise 3 to see how to match the descriptors.
- Count how many matches fall into each cluster. One way to check in which cluster a match fell, is to look at the cluster numbers of the adjacent pixels of the matched 2D coordinates in the "label image" you created in step 4.
- Assign each cluster the object class of the training image which had the most matches in the cluster. You might have to introduce some kind of normalization here, otherwise object classes of large objects like the cookiebox will get assigned most of the time because they get many false matches.

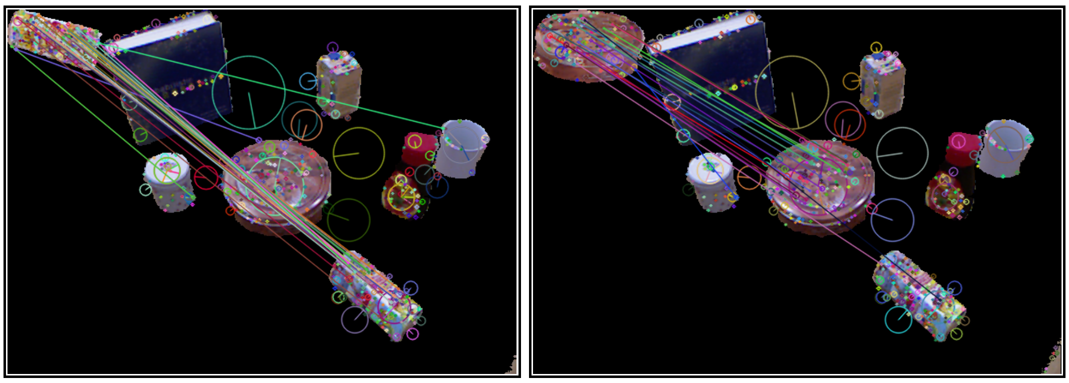


Figure 4: Example matching results for two object pointclouds.

Useful commands

- cv2.cvtColor: Converts an image from one color space to another.
- cv2.putText: Renders the specified text string in the image at a specific location.
- cv2.dilate: Dilates an image by using a specific structuring element.
- cv2.morphologyEx: Performs advanced morphological transformations.
- cv2.applyColorMap: Applies a color map to an image, e.g. for plotting the clustering results.
- np.asarray(pcd.colors) - Get the color information of all points of a pointcloud as a Numpy Array.
- open3d.geometry.PointCloud.cluster_dbscan: Cluster PointCloud using the DBSCAN algorithm, returns a list of point labels, -1 indicates noise according to the algorithm.
- open3d.geometry.PointCloud.segment_plane: Segments a plane in the point cloud using the RANSAC algorithm.

Bonus points

You can make improvements and add additional functionality to get up to 10 bonus points for this exercise. Some examples that could be implemented:

- Improve the recognition rate. You can think of additional or better features (e.g. use color information as well) to also be able to reliably recognize the cup which doesn't have a lot of texture.
- Instead of calculating 2D features on the projected images, implement an approach using 3D features like Point Pair Features (PPF) or Signature of Histograms of OrienTations (SHOT).
- Calculate the object's pose. RANSAC could be used on the counted SIFT matches for the recognized cluster to retrieve the object's correct geometric pose in the scene.
- Improve the clustering. Also the clustering algorithm could benefit from using the color information as an additional cue to separate objects. One approach could be to incorporate an additional "color distance" next to the spatial distance term which is used to determine how "similar" points are.
- Use a Deep Learning model for classification. A possibility would be to use a pre-trained model, remove the last layer and use the features produced by the network instead of the SIFT features. You can also try to retrain the final layer, but due to the small size of our training set you should apply data augmentation to artificially create more training images. Use Keras+Tensorflow for your implementation. You can install the additional libraries in your `mvcr` environment using the .yml files provided for exercise 4 and the command `conda env update --name mvcr --file mvcr_ex4[_gpu].yml`. If you want to train using Google's GPUs, you should be able to write a small script in Google Colab and download the trained model to be used in your pipeline.
- Anything else you can think of which would be a logical extension. If something you do does not work, you can still include it and discuss its failure in the documentation.

You may receive some bonus points for attempts at implementing these the above, or any other ideas you have, even if they do not perform well. **You must explain extensions in your documentation to receive bonus points.**

Documentation (8 points)

- Include images to support your answers. These should be referenced in the answers. The document should include information about what parameter settings were used to generate images so that they can be reproduced.
- Your answers should attempt an explanation of why an effect occurs, not just describe what you see in the supporting images.
- Be precise. "It is better" or similarly vague statements without any additional information

are not sufficient.

Points may be deducted if your report does not satisfy the above points.

1. Show a screenshot of the result for each test point cloud (like the "recognition result" image). You can add text to an array using the `cv2.putText` function. This is required so we can see all your results. (no points for this)
2. Describe your approach. Have a separate section for each step, explaining what you do in that step and what its intended purpose is (e.g. apply clustering so we can create a set of points which we assume are all part of the same object). In each section, list all additional ideas you tried to improve that part of the process, even if they didn't work. Explain why you tried each, and summarise the results. Mention the files and line number ranges where the extensions can be found. You should include images showing what the extension does, if relevant. (3 points)
3. Evaluation of the result and discussion. How many correct detections can you get for each class and overall? You must present these results quantitatively. Include tables and/or graphs. What could be the reasons why some classes are recognised better than others? What parameters have been used and how do they influence the results? If you implemented extensions, you should compare detection performance before and after adding them. You should also include details about the runtime of each stage (excluding RANSAC and clustering, unless you have implemented something different to previous exercises) (5 points)

You must specify any parameter settings you used to generate images that you include.

There is no word limit for this exercise.

You must include instructions for how to run your code. This can be in a separate readme file submitted in the .zip, or as part of your documentation.

Assistance

Please keep to the following hierarchy whenever there are questions or problems:

1. Forum: Use the TUWEL forum to discuss your problems.
2. Email for questions or in-person help requests: machinevision@acin.tuwien.ac.at

Upload

Please upload all files required to make your system work, any code for extensions (even if it does not work well), the helper files from previous exercises, if you made use of them, and your documentation (pdf-format) **as one ZIP-file** via TUWEL.