Computer Vision Project
Summer 2024
Florian Kordon

Sheet 1, starting from April 16th, 2024, due April 29th, 2024, 02:00 p.m.

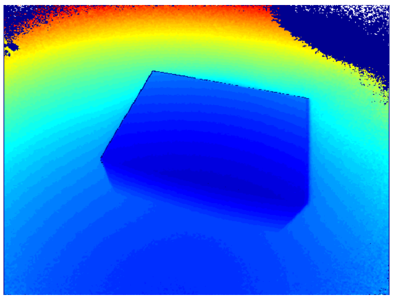
Exercise 1.1: Box Detection

In the first exercise, you will implement a small program that estimates the size of a box from a distance image. As discussed in the lecture, we will use Python as the programming language of your choice. While the slides of the first lecture provide most of the tools, you will need to research how to solve the task best. The following description will serve as a guideline to complete the task.

Feel free to write your own helper functions and try to keep your code structured and easy to understand. You may include your own ideas and extensions as you like.

Have a look at the following libraries, as they contain valuable functions: numpy, scipy, scikit-learn, matplotlib

(a) ToF amplitude image


(b) Distance image

Getting and reading the data Download the example files from StudOn and extract them into your working directory. The four `.mat`-files contain different examples that you can use for testing your implementation. Each example consists of an amplitude image `A`, a distance image `D`, and a point cloud `PC`. Pixels can be accessed via `A[height,width,:]`. Please note that all three representations are registered. This means that the information at the position `(i=50,j=50)` in the amplitude image `A[i,j,:]` corresponds to the distance information at `D[i,j,:]` as well as the 3D point coordinates of the point cloud at `PC[i,j,:]`.

Please begin with loading[1] one of the examples and visualize its content, for example with the command `imshow`[2] or `scatter`[3] using the matplotlib package. This visualization is straightforward for the 2D image data but can be a bit tricky for the 3D point cloud. To circumvent potential issues, please be reminded that the point cloud is structured using the same 2D grid as the 2D images, meaning that you can address individual points in 3D by using the column and row indices `(i,j)`, whereas the `(x,y,z)` coordinates of the 3D point at that position is provided in the last dimension of `PC[i,j,(x:y:z)]`.

Please continue by experimenting with the various filters introduced in the lecture and evaluate if one of them can be useful to improve the input data.

Note that it is valid to subsample the point cloud to improve the runtime (e.g., when using `scatter`). However, you need to ensure you do not lose detection accuracy.

[1] `https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.io.loadmat.html`
[2] `https://matplotlib.org/2.0.0/api/pyplot_api.html#matplotlib.pyplot.imshow`
[3] `https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html`

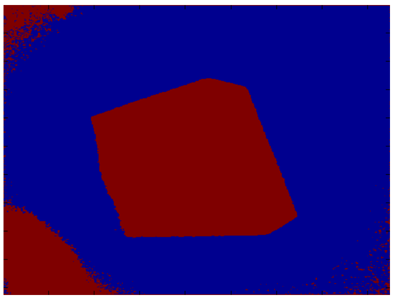
RANSAC

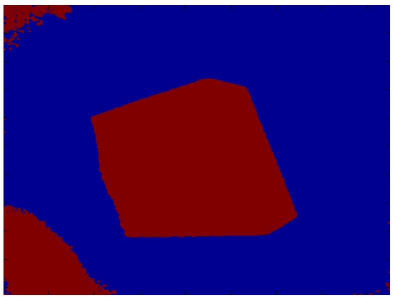
Subtracting single distance measurements from each other to calculate the height of the box is sensitive to noise. A better approach consists of finding two planes that approximate the floor and the top of the box and then calculating the distance between the planes. A simple solution to finding dominant planes in a point cloud is to use RANSAC to find the plane models with the most inliers. In our example, these plane models correspond to the floor and the top of the box.

Use the pseudo-code provided in the slides and the complementary material to implement RANSAC (you should implement it yourself and NOT use any implementation provided in, e.g., scikit-learn). Your implementation should take three parameters: a point cloud containing 3D point vectors, a threshold required to evaluate a model's quality, and a parameter for the maximum number of iterations. Using a parameter/normal representation for the plane models is advised.

$$n_x x + n_y y + n_z z = \mathbf{nx} = d \tag{1}$$

Your code should return the best-fitting model if the maximum number of iterations has passed or all candidate points are within the inlier set. Use your implementation of RANSAC to find all pixels that belong to the floor. Visualize all inliers with a mask image that represents inliers with a value of 1 and outliers with 0. Experiment with different parameters and find a reliable parameter set that finds the floor plane.


(a) Floor mask


(b) Filtered floor mask

Some additional notes on the implementation:

- The point cloud may contain invalid measurements which can be identified by checking if the z-component of a vector is 0. These points should be ignored in your RANSAC implementation.
- Due to the rather high number of points, you should make sure that you use numpy functions efficiently and avoid using loops. Useful functions could be `numpy.cross`, `numpy.all` or `numpy.matmul`/`numpy.dot`'s capability of calculating the dot product of two matrices.

Filtering on the Mask Image

Use morphological operators[4][5] to improve the quality of the floor mask. Evaluate which operators suit the task best. Use the resulting mask to find all pixels/3D points that do not belong to the floor (e.g. the box, background objects, noise, ...).

Finding the Top Plane of the Box

Create a new matrix that contains all 3D-points which are not part of the floor. Use your implementation of RANSAC to find the dominant plane within this set. The example data is captured such that the

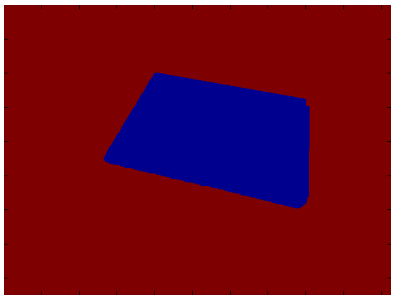
[4] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.ndimage.morphology.binary_closing.html`

[5] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.ndimage.morphology.binary_opening.html`

second largest plane is the top of the box. Create a new mask image that represents the inliers of the box plane.

Pixels which do not truly belong to the box will later create errors in the size estimation of the box. It can be assumed that the largest connected component in the mask delimits the top of the box. Find the largest connected component in the mask with scipy's `label`[6] command. Evaluate if additional preprocessing or filtering steps can improve your result.

Abbildung 3: Box top component



Measuring the Dimensions of the Box

At this point everything that is required to estimate the size of the box is known: a mask that delimits the top plane of the box and plane representations of the top and the floor. Calculate the height of the box by computing the distance between the two planes. One way to find the length and width of the box is subtract the 3D-coordinates of the corners from each other. Analyze the pixels of the box mask to determine the corners.

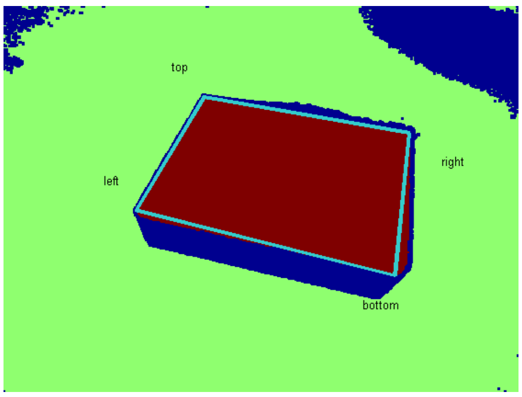
Exercise 1.2: Box Detection - Discussion

Create a simple visualization of your results, either text, figures or both. If you made any extensions or included own ideas in the algorithm visualize these as well.

This simple implementation comes with several weaknesses. Identify some of these drawbacks and make suggestions on how to make the algorithm more robust, more accurate or faster.

Show your implementation and your results to one of the advisors.

Abbildung 4: Visualization of floor, box and box corners.



[6]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.label.html`