Exercise 5

Download files for exercise 5

Starting with this exercise, you will work with *3D pointclouds* recorded with a depth camera instead of 2D images as input data. The goal of this exercise is to find the dominant plane (i.e. the floor) in the given pointclouds, as well as extracting multiple planes from more complex scenes. A common method for that is the RANSAC (Random Sample Consensus) algorithm, which has already been discussed in the lecture (Lecture 9). You will only need to process the spatial information of thepointcloud, color information is not used yet. An example pointcloud of the dataset is shown in Figure 1.

We will use the library Open3D http://www.open3d.org to process the pointclouds. You can find a tutorial on the official site. However, we will mostly use this library for its I/O capabilities and for visualization. For implementing the RANSAC algorithm, we will transform the points to NumPy arrays and perform the calculations on them.

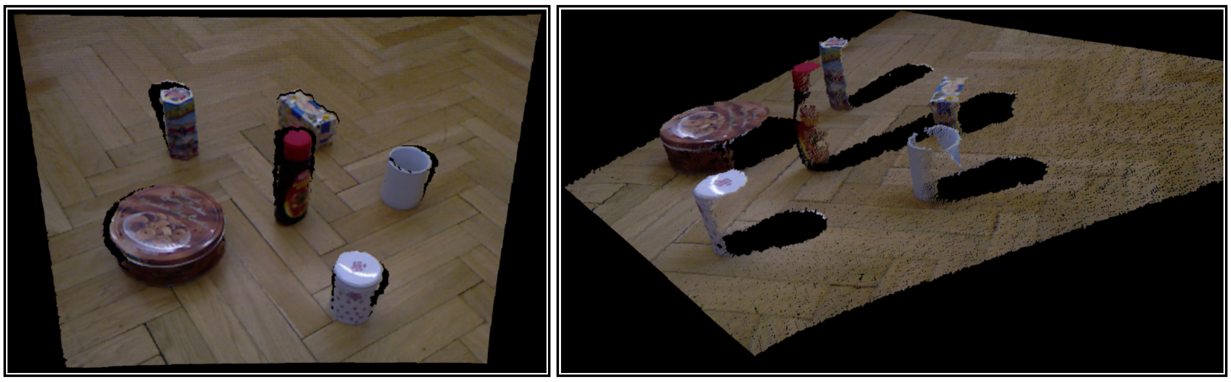

Figure 1: Example pointcloud from the dataset shown from two different viewpoints. Note the difficulties capturing the cup as one object.

Part 1: Implementation (10 points)

In this exercise, the RANSAC algorithm should be fully implemented to detect planes in a pointcloud. We will also explore two different error functions: MSAC and MLESAC. In Figure 2 you can see the result of the RANSAC algorithm with the detected plane colored in red.

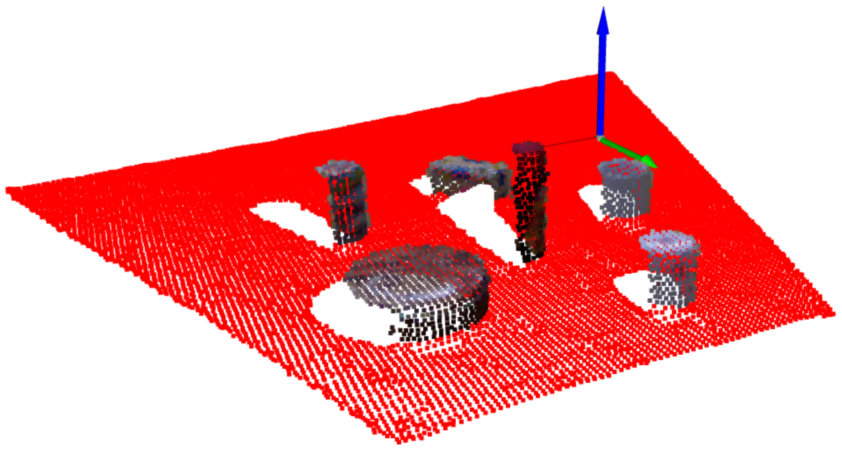

Figure 2: Succesfully detected plane and colored the inliers with red of the down-sampled point cloud.

1. Basic RANSAC (5 points)

Input parameters

- `pcd` : The Open3D pointcloud. The property `pcd.points` holds the points in 3D space. It can be easily converted to a NumPy array with `np.asarray(pcd.points)` with shape `(no_of_points, 3)` . For more information refer to the Open3D documentation.
- `confidence` : Solution Confidence (in percent): Likelihood of all sampled points being inliers.
- `inlier_threshold` : Max. distance of a point from the plane to be considered an inlier (in meters)
- `min_sample_distance` : Minimum distance of all sampled points to each other (in meters). For robustness.

The input parameters for the RANSAC function can be altered in the main script `main.py` . To reduce computation time, the pointclouds are down-sampled before we process them further. Open3D provides two different down-sampling approaches which you can choose from: uniform_down_sample() and voxel_down_sample()

Output parameters:

- `best_plane` : Numpy Array `np.array([a,b,c,d])` holding the coefficients of the calculated plane, defined by the equation `a*x + b*y + c*z + d = 0` .
- `best_inliers` : Boolean (N,) array, containing a True for every inlier and a False for every outlier in p at the corresponding position. E.g. inliers = `[False, True, False, False, True, False, True, True, ...]` means, the points p with column indices 1, 4, 5, 7,... are inliers and therefore part of the

calculated plane.
- `num_iterations` : The number of necessary iterations to achieve the required solution confidence `confidence` .

General RANSAC sequence:

- See lecture slides
- Dynamic adaptation of the number of iterations depending on the number of found inliers (slide 63)
- After finding the largest set of inliers, the plane fit should be refined once more considering all inliers (see below)

You should implement the cost function in `ransac_error` in `error_funcs.py` and use the callable parameter `error_func` in `fit_plane` , similar to how you did it in exercise 3 with the descriptor functions. This allows the cost function to be changed easily using the input parameters, as will be necessary in Step 3.

Refining the fit

After RANSAC has divided the set of points in inliers and outliers, the plane fit can be further improved by finally considering ALL inlier points to calculate the plane parameters a, b, c and d ($ax + by + cz + d = 0$). For that, an overdetermined system of equations has to be solved, which can be done in various ways, such as a least squares approximation, as follows.

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Parameter d is free to choose, such that for e.g. $d = -1$ we get the following simplified system of equations:

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

This system of equations is overdetermined (n equations for 3 unknowns, n = number of inliers). To get the least-squares solution of this system, NumPy offers the function [numpy.linalg.lstsq()]

A plane can be defined such that points on one side of it are in a "positive" space, and others in a "negative" space, depending on the direction of the normal. For our purposes this does not matter. Ensure that all the normals have a positive z value, and make sure the rest of the vector components are kept consistent with this.

Complete **fit_plane.py** and **ransac_error.py**.

2. Multi-plane extraction (2 points)

Sometimes the pointcloud you are dealing with may have more than one plane present. Implement a function which will extract multiple planes from a single cloud. You should apply the function `fit_plane` in a loop. After each iteration, you store the inliers of the plane and remove them from the pointcloud and apply `fit_plane` on the remaining pointcloud. You should stop when the number of points in the extracted plane is smaller than some proportion of the original cloud size. There is a convenient function to remove the inliers from the pointcloud: o3d.geometry.PointCloud.select_by_index.

Complete **filter_planes.py**.

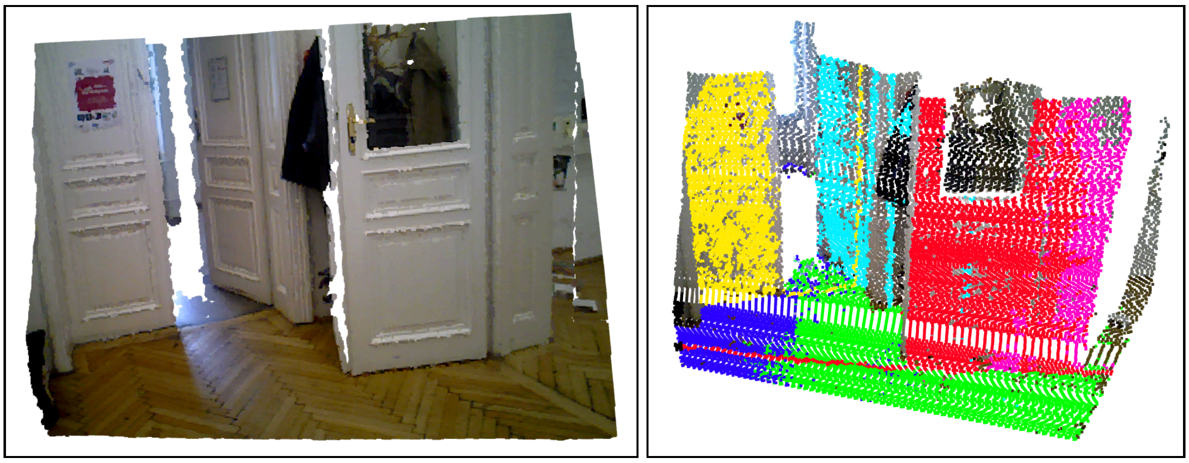

Figure 3a (left): Example pointcloud with multiple planes visible. Figure 3b (right): Example result on down-sampled point cloud with planes in different colors.

3. MSAC (1 point) and MLESAC (2 points)

Use Section 5 of the paper on MLESAC by Torr and Zisserman to implement MSAC and MLESAC. Equation 10 shows the full negative log likelihood function. Note that there appears to be a misplaced bracket in the equation. You can define the error in your function as

$$-L = -\sum_{i=1,\ldots,n} \log\left(\gamma \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{e_i^2}{2\sigma^2}\right) + (1-\gamma)\frac{1}{v}\right) = -\sum_{i=1,\ldots,n} \log(p_i + p_o).$$

You can ignore the squaring of the normalization constant in equation 19. You are not required to use 1.96 sigma as the threshold. The value of `v` should be set to the length of the maximum diagonal of the cloud. For MLESAC, the value σ should be set to `sigma = threshold/2` . This way the threshold works in a similar way compared to the threshold in the other examples because 95% of points should lie within 2σ of the normal distribution. As the paper states, it is sufficient if you do 3 iterations of the expectation maximization algorithm and don't need to check for convergence. You can assume that an inlier is a point which falls within the threshold (or 2σ).

Complete `msac_error` and `mlesac_error` in **`error_funcs.py`** .

Useful commands

- np.full - Return a new array of given shape and type, filled with a certain Value
- np.random.randint - Return random integers
- np.linalg.norm - Return one of eight different matrix norms, or one of an infinite number of vector norms
- np.cross - Return the cross product of two (arrays of) vectors
- np.dot - Return the dot product of two arrays. If both arrays ar 1-D arrays, it is the inner product of vectors.
- numpy.linalg.lstsq - Return the least-squares solution to a linear matrix equation
- np.nonzero - Return the indices of the elements that are non-zero
- np.asarray(pcd.points) - Get the 3D coordinates of all points of a pointcloud as a Numpy Array.
- o3d.geometry.PointCloud.select_by_index Function to select points from input pointcloud into output pointcloud
- o3d.geometry.PointCloud.get_max_bound - Returns max bounds for geometry coordinates.

Forbidden Comments

- o3d.geometry.PointCloud.segment_plane - Segments a plane in the point cloud using the RANSAC algorithm. Of course it is forbidden, but might be nice for comparison.

Part 2: Documentation (6 points)

- Answer each numbered question separately, with the answer labelled according to the question number. For example, question 2 should be numbered 2. You do not need to include the question text.
- 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. Good result (screenshot) with pointcloud number corresponding to the last digit of your matriculation number (e.g. matriculation number 123456 -> pointcloud 6). (0.5 points)
2. For each of the 3 RANSAC parameters (`inlier_margin` , `min_sample_dist` , and `confidence`) a short discussion of how the parameter influences the result (with supporting screenshots). (2 points)
3. Apply multi-plane extraction to the desk, door and kitchen pointclouds. Describe the results and explain issues that you observe. Discuss briefly ways you might solve the issues you describe. Additionally, compare the two different down-sampling approaches and discuss issues that might occur. (1.5 points)
4. Compare RANSAC, MSAC, MLESAC. We provide the script `sac_comparison.py` that can be used to generate data and evaluate the different error functions with multiple metrics. Points are generated around a plane with noise added in z-direction. You can specify the noise parameters, number of points and also how often the tests should be performed. You can change everything in that file, it is mainly thought to give you a good starting point for your experiments. Choose at least two metrics to compare the performance of the different error functions. You can choose from the metrics already implemented in `sac_comparison.py` or add additional metrics (e.g. computation time). Present a few graphs, and briefly describe interesting or surprising things that you see (if any). You are not expected to be rigorous. (2 points)

You must specify the parameters used in all of your screenshots.

Maximum 1500 words, arbitrary number of pages for result images (referenced in the text). Violating the text limit will lead to point deductions!

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 Python files, as well as your documentation (pdf-format) **as one ZIP-file** via TUWEL.