NPM3D - TP5

Aubin TCHOÏ

February 2023

Question 1

The screenshot below shows the "best" segmentation I obtained using CloudCompare.

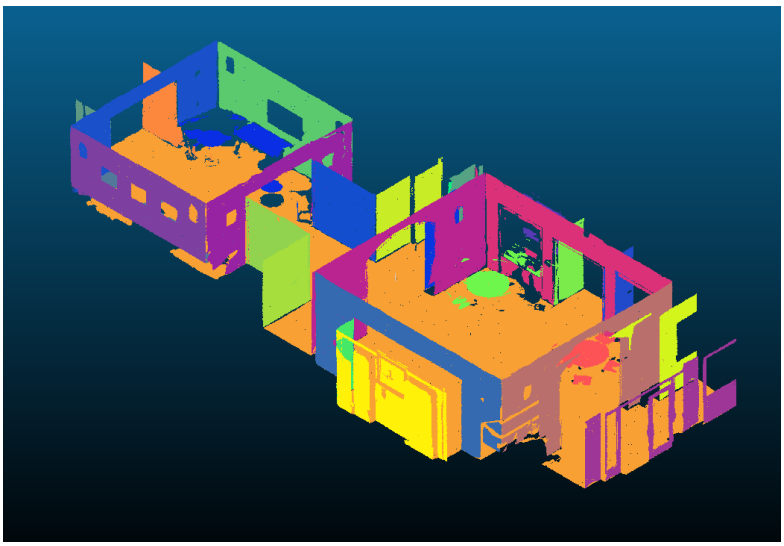


Figure 1: Segmentation obtained using RANSAC Shape Detection plugin

I found that changing the value of the minimum support points per primitive (plane) for a value too small could drastically increase the number of planes generated (2335 planes with a value of 10). I chose a value of 100. I found that increasing the overlooking probability could lead to finding the planes in the wrong order, with smaller planes being found before bigger planes, and I kept the value of 1%. I set a max distance to primitive of 0.1, a max normal deviation of 10° and kept the default sampling resolution of 0.201.

The output of the plugin contains 172 planes, which are correctly sorted by size with the first one being the floor, the second one being a large wall, etc.. Each plane is also a connected component of the same orientation (it can be a plane with a hole but cannot have some curvature) and we observe that past the 50th plane, the planes do not match an actual plane and only group outliers or noise altogether.

Question 2

The result we would like to observe is one where the planes extracted match with planes in the point cloud.

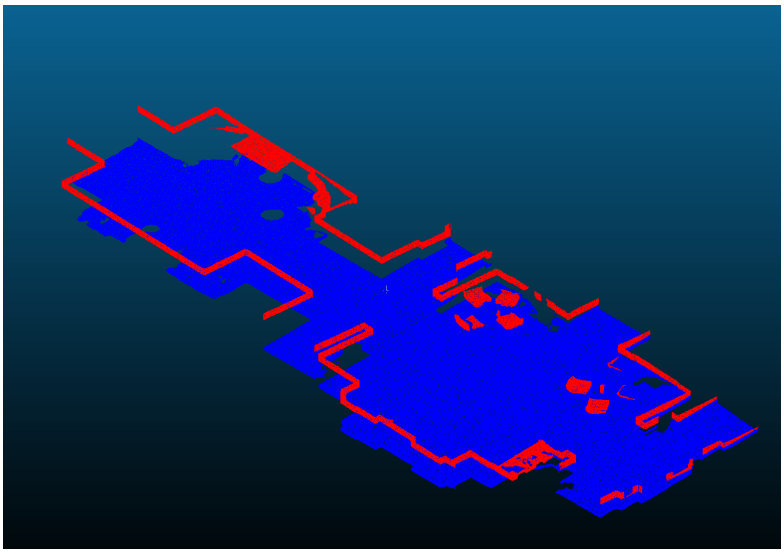
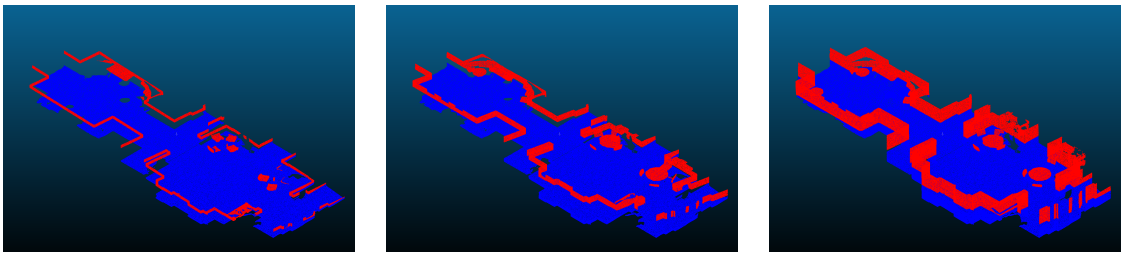


Figure 2: First two planes extracted using RANSAC

Here, the second plane extracted by RANSAC is actually not a plane as it contains a collection of walls directed perpendicularly to the plane being fitted by RANSAC, whose normal is equal to $(0.0090, -0.0126, -0.9999)^\top$, which means the plane is horizontal. There are two issues here: the second plane extracted does not contain points that belong to planes of the same orientation, and it is not a connected component.

The current implementation does not actually try to fit planes, but instead attempts at finding the box of infinite width and length of height 2 times the threshold that fits the largest number of points as shown by the figures below.



a) b) c)

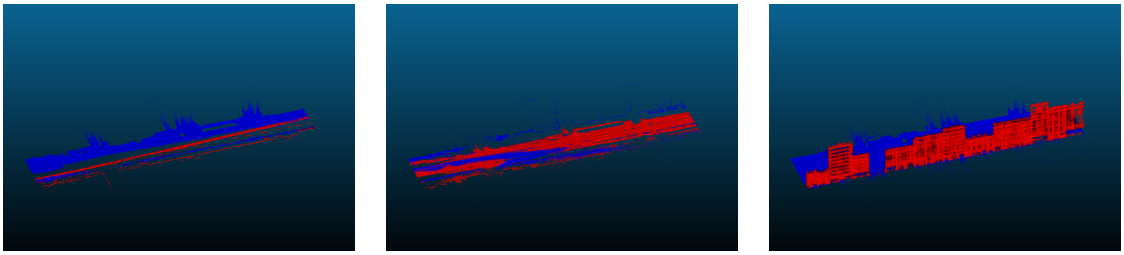
Figure 3: RANSAC with various thresholds a) 0.1 b) 0.2 c) 0.5

We therefore have to include in the voting strategy a measure of similarity between each plane local to a point and the plane to fit using RANSAC.

Question 3

I applied the RANSAC plane detection to the `Notre_Dame_Des_Champs.ply` point cloud introduced in a previous TP. The expected result is to find a first plane matching the road, a second one matching the facades of the buildings opposite from Notre Dame des Champs as they form a planar surface of a significant size, and other ones matching each building.

This point cloud describes a more complex scene with more planes than the `indoor_scan.ply` file, therefore the probability of drawing 3 points in the same plane is smaller. To answer this issue I increased the number of draws from 100 to 500. The point cloud is also noisier, which causes points to be incorrectly excluded from certain planes. I increased the threshold on the maximum distance to a fitted plane until the entirety of an area that could be considered as planar was matched. I found a threshold value of 0.5.



a) b) c)

Figure 4: RANSAC run with threshold equal to a) 0.1 b) 0.2 c) 0.5

When increasing the number of planes to find to 5, the method ran in 166 seconds with the following result:

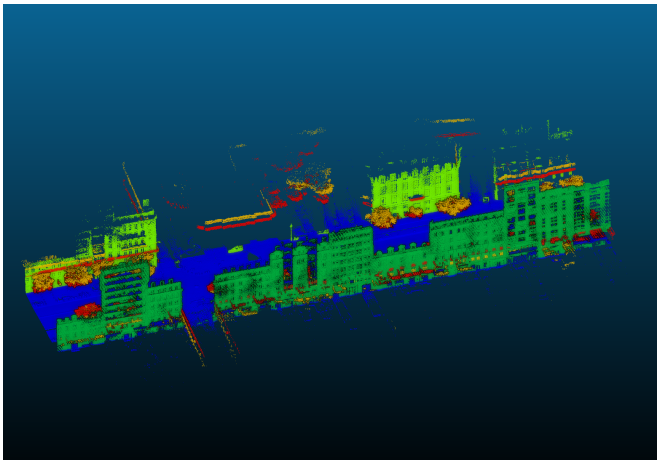


Figure 5: First five planes extracted using RANSAC

The first three planes extracted (blue, dark green, light green) match what was expected (road, building opposite from Notre Dame, other buildings). The issues mentioned previously are highlighted by the presence of vegetation: what RANSAC recognizes as the fourth and fifth planes extracted are not planes as they mostly contain vegetation and small unconnected parts of buildings. If we restrict our search to the first three planes, we get the result below.

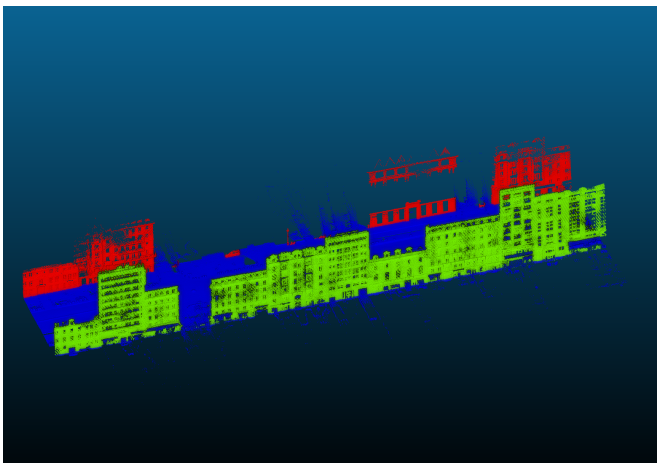
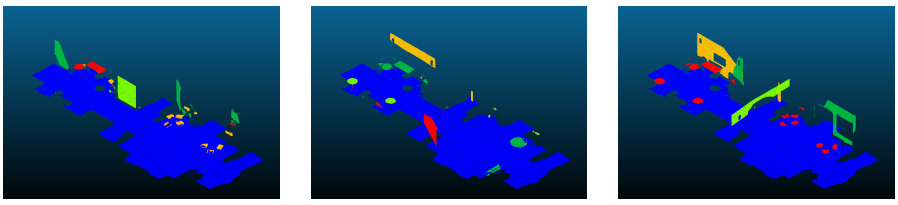


Figure 6: First three planes extracted using RANSAC

Question 4

The method implemented in function `recursive_RANSAC_with_normals` consists in including in the voting strategy of RANSAC a threshold on the angle formed between the normal at a given point (pre-computed before running RANSAC) and the normal of the plane to fit. All the points that have normals that are not aligned with the normal of the plane will be rejected, and therefore the walls will no longer be considered as being part of a horizontal plane.

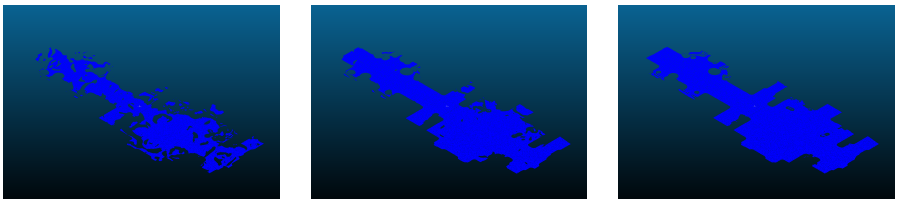
When running this implementation we observe that it displays more sensibility to the randomness of the draws as shown by the figure below. Indeed, there are far less good candidate planes than in the previous implementation, and finding them is harder. We therefore increase the number of draws from 100 to 500.



a) b) c)

Figure 7: Various results obtained with 100 draws

To set the threshold on the angle I chose the lowest value that would still count the floor as a single plane. To do so, I ran iteratively the method with `nb_planes` equal to 1 and increasing values of the maximum angle. I found a value of 0.1 radians.



a) b) c)

Figure 8: Biggest plane extracted with α equal to a) 0.01 b) 0.05 c) 0.1

I kept the value of threshold on the distance to the plane of 0.1 to obtain the following result:

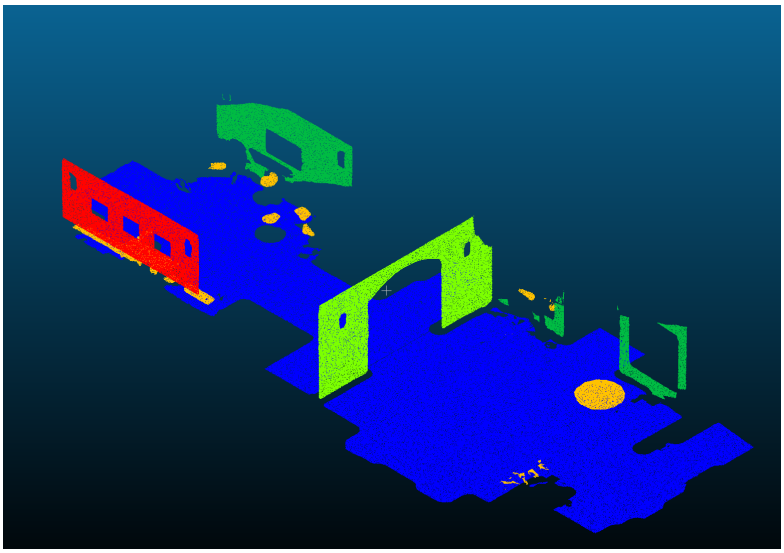


Figure 9: First five planes extracted using RANSAC

Bonus question

Sampling method

The runtime complexity of RANSAC is directly linked to the probability of finding good sample sets. Since shapes are local phenomena, the probability that two points belong to the same shape is higher the smaller the distance between the points. We can thus define a new sampling strategy that exploits this fact to increase the probability of drawing three points that belong to the same plane: a first point is drawn randomly, then the two other points are selected among the nearest neighbors of the first point. We use a kdtree to compute the nearest neighbors efficiently. A new parameter that controls the size of the neighborhood has

to be fine-tuned: in practice I will use a value of 5000 as values that are too low tend to add noise to the normals when the points selected are too close from one another.

What makes this method faster is that we can reduce the number of draws since each one of them is more sample efficient. In practice we will lower by a factor 5, we have no information on the size of each plane to extract and therefore cannot derive a bound on the number of draws required to reach a certain probability to effectively draw three points on the same plane. We will also increase the threshold on the distance to the plane to 0.2, indeed this sampling already adds a certain coherency in the planes generated, and we need to be able to fit very large planes such as the floor using three points that are relatively close from one another.

The initial run without any improvement took on average 158 s (averaged on 10 runs) and spent roughly 46% of its time picking random points and 49% on score evaluation (36% on distance to plane evaluation and 13% on angle between normals computation and comparison). The run with the new sampling method leads to the result below, it took 20 s on average, with now 74% of the time spent on score evaluation.

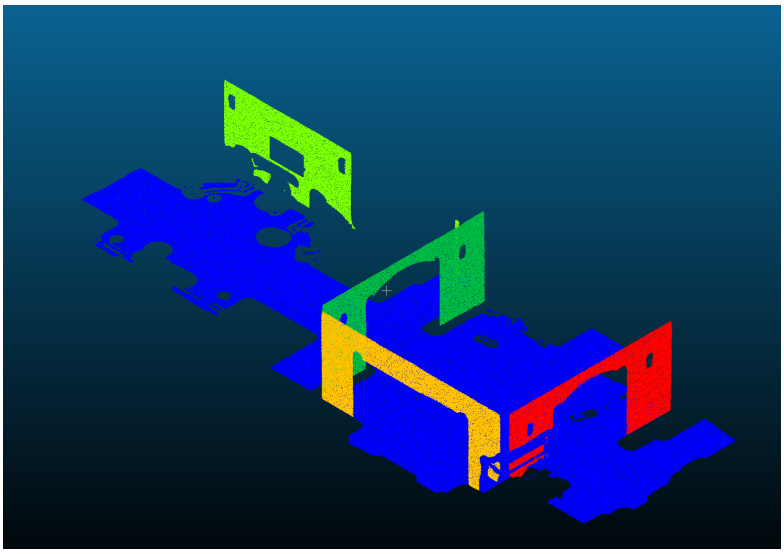


Figure 10: First five planes extracted using RANSAC with a more efficient sampling method

We can see that this sampling method also partially addresses an issue raised previously on the fact that the planes generated do not belong to a same connected component. Indeed, with this sampling method the points selected have a much higher probability of belonging to a common connected component because of their proximity.

Stopping criterion

Another possible improvement would consist in stopping the search for candidate planes before reaching `nb_draws` samples by checking the size of the best candidate found and computing a bound on the probability on the existence of a bigger plane in the point cloud. A possible bound can be obtained by assuming that there is bigger plane of size n, if we denote by N the size of the point cloud, the probability $P(n, t)$ of not detecting it after t draws can be expressed as (the factor 3 comes from the fact that we sample 3 points each time):

$$P(n,t) = \left(1 - \frac{\binom{n}{3}}{\binom{N}{3}}\right)^t \approx \left(1 - (\frac{n}{N})^3\right)^t$$

$P(n,t)$ is decreasing in n, if we set n to be the size of the current best candidate plane after t draws, we can set a threshold as following: if $1 - P(n,t)$ is higher than a certain value it means that if there was another plane as big as the current best one we would most likely have detected it.

In our case we have to set a threshold extremely low (0.6) to trigger the early stopping criterion and it only works for the first plane (floor) because it is much larger than the other ones. We lowered the execution time of the method to 17 s on average for the same results.

Other possible improvements

Another speed up could be obtained by observing that the remaining execution time principally lies in the score evaluation (computing distance from a point to a plane and angles). A possible method would consist in generating all the possible candidate planes and not performing these computations on the whole point cloud but only on subsets of increasing size of it. This would allow to discard candidates based on estimators of their performance on the whole point cloud. I did not implement this method as it requires generating all the candidates at once and is therefore more memory-consuming (more than possible using my laptop) and would prevent the early stopping criterion from being used.

Final result

The screenshot below shows the result obtained when extracting 10 planes, executed in 32 s.

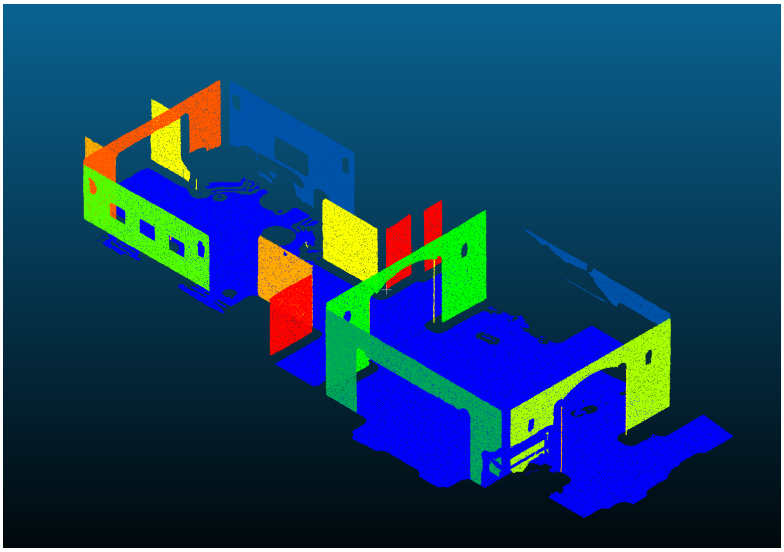


Figure 11: First ten planes extracted using RANSAC

The result obtained is quite similar to the one obtained with the CloudCompare plugin, the main improvement we could do would be to implement a way to only keep connected components as multiple walls that are aligned are matched in a same plane (see the red planes in the middle).