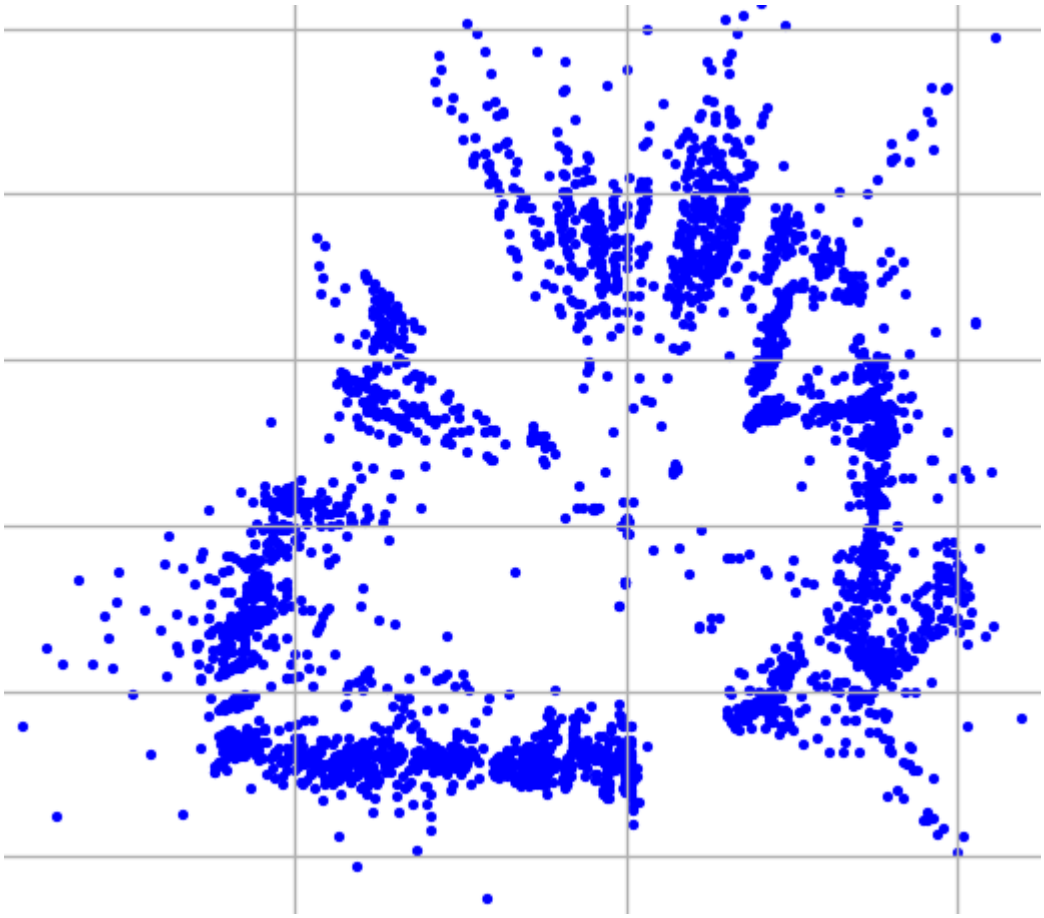
As of last week, we have thought to be basically done.
We had both a scanning procedure and an algorithm to find the exit points that should be sufficient.
That being said, after checking our entire process, we noticed that we output very incorrect exit points (they were inside the room!).
After further investigation we noticed that the lines the RANSAC algorithm outputs, even after messing with the hyperparameters, do not look like a rectangle in the slightest!
Since the RANSAC algorithm seems to have no bugs (remember we have debugged it quite a bunch last week!) we concluded that we need to investigate how our data looks like after the scanning.

Visualization:

one of our goals for this week was to create a visualization for our data and our algorithm. We have created a small python script that given a txt file of the points, plots them on the screen. After seeing what the output of the scripts is, we have come to see a very worrying problem. There is simply too much noise, unscanned data, and inaccurate data!



Because of this very worrying problem we have decided to make some changes to both the simulator's scanning algorithm and the algorithm to find the exit points.

Simulator:

The original scanning algorithm of turning clockwise a bit and then moving around a bit, seemed to create too much noise and inaccurate data, that even to a human eye (after filtering noise) can't conclude both the precise shape of the room (assuming it's a rectangle) and where are the exits.
Because of this, we needed urgently a better scanning algorithm,.
Here's a summary of the evolution of our scanning algorithm:

1) Initially, we used the scanning algorithm in runSimulator.cpp:
    1) Move a bit in each direction.
    2) Rotate a bit.
    3) Return to step 1 until the total rotation is a full circle.
2) We started off by tweaking some numbers – the rotation angle, the stopping condition etc...
3) Added a retry option: if after an iteration of moving + rotating the ORB-SLAM tracker got lost, retry with the same angle.
4) Added a hard retry option: the above is dubbed soft-retry, a hard-retry repeats the scans of the last several scanned angels and occurs after several soft-retries.
5) Added an initial “fast circular scan” – this scan stays in the same place and only rotates; it helps with familiarization of the area before the more rigorous scans. This was our algorithm at the beginning of the week. Even after all these changes, we still consistently got “track lost”.
6) Believing the problems arose from staying in roughly the same position, we devised a new algorithm:
    1) Repeat X times:
    2) Go to a random location, “fast circular scan” from there.
7) Added a retry option: since commands that weren’t rotation commands could fail, we wrote a “reverse command” function, we apply it if we get lost.
8) Added the initial scan as described in step 5.
9) Added a random rotation command after each iteration, this allows us to use more than two axis in our movements.
10) Modified “fact circular scan” to work in two parts: clockwise and anticlockwise, that means the if we get lost during a “fact circular scan”, we’ll still learn some data.

11) Made the two parts of the “fact circular scan” to overlap a bit as insurance when one of them fails.
12) Change to the initial scan which makes it more like version 1:instead of just rotating, go in a specific pattern which maximizes visibility:
    1) Go backwards – this deals with possible obstacles in the way.
    2) Move to the right and look to the right, return to center.
    3) Move to the left and look to the left, return to center.
    4) Go forwards – canceling out the initial backwards move.
13) This is our new algorithm, not only does it work much better (in giving a room looking scan), but it also gives much more points due to the larger and randomer number of movements.

To help us analyze the performance of the algorithms, we added visualization (finally), however that seemed to just fill our screens and make it harder to see what’s happening in later iterations, we decided to instead opt for logging: we currently log:

- In each iteration, no matter the part, we log the current location and iteration number.
- In “fast circular scan”, we log every rotation done.
- In the main part of the new algorithm: we log the command, success type (“fail”: track lost after applying command, “success part”: track not lost after applying command but lost after “fast circular scan”, “success full”: track not lost).

Algorithm:

As there is a lot of noise, we decided to create an initial noise filtering algorithm, that will make our RANSAC far more accurate. We have dubbed this algorithm Iterative-Pseudo-Clustering (or IPCL for short). The algorithm iteratively filters the data with an algorithm called Pseudo-Clustering (or PCL for short), PCL simply removes points from the data that have too little points close to it, there is also a reverse version (indicated by a flag, called RPCL) which removes points that have too many points close to it. IPCL seems to filter the data well enough that we immediately move to the algorithm that finds the room (RANSAC).

We had some problems with finding the right hyperparameters for some functions but believe we found appropriate values that scale well.
That being said, after running the algorithm it seems that RANSAC returns one incorrect line, which is on a couple very dense clusters. Because of that, we decided, after running the algorithm, to devise some scoring algorithms for the points close to the chosen lines, and run them on the 5 lines RANSAC found (the 4 correct ones and 1 incorrect one):

1) sum of all distances for each 2 pair of points that are close to the line (prioritizing long dense lines)
2) max distance of any 2 points close to the line (prioritizing long lines)
3) sum of all cross products for each triplet of points that are close to the line (prioritizing line linearity)

We thought of another way: instead of running both PCL and RPCL in the beginning, we only run PCL and run RPCL once per line.

All those methods work to an extent but we're not sure yet which one fits best (probably the last one).

Next week (aka tomorrow):

We do not have any crazy plans in mind; choose the best of the above, a bit of refactoring, uploading to github, and writing a detailed final report.