Simulator & ORB-SLAM

Due to sone timing issues, we only had (roughly) a day to combine the new implementation of the exit finder with the existing navigation code. In that time, we ran into several linking issues since we wanted to preserve the existing code, to overcome this we decided to take a snapshot of the current VM state and override the existing RoomExit code with our code, however, due to the timing issues, this will have to wait for the upcoming week.

Even though we had some linking issues, we still managed to integrate the code with a relatively small amount of changes, of course, because of the linking issues we couldn't run it but the compiler doesn't detect any issues with the code so hopefully, once the linking issues are resolved, this will work after only a handful of attempts.

Until that, we ran several tests, we found out that at the initial configurations, ORB-SLAM loses track (the current frame has a "Track lost" status), to overcome this we found several solutions, all with their own advantages and disadvantages:

1) Slow down the movement speed & add pauses between the scans - this makes the transition into new frames smoother and thus easier for ORB-SLAM to handle, however it does make the overall code slower.
2) Retry the scan – this is essentially the same as adding a pause whose length depends on the amount of retries, since attempting a rescan is relatively swift, we decided to use this method as its drawback only occurs when there is a failure and even then, it is relatively minor.
3) Go back to the starting point and slowly rotate the camera so that the angels match – this should theoretically work because when we scan, after each iteration we go back to the same starting point, this is essentially a more sophisticated retry.

We found that combining solutions 1 and 2 fits our needs, the combination allows for only a small speed decrease and a small amount of retries, there are still some frames where the track is lost (especially if the field of view is partially blocked) but those have become so few we can ignore them. There are undoubtedly other solutions, but these seem to suffice.

Exit-finder

This week we have implemented the part in the algorithm that finds the quadrilateral the room represents and from there finding the exit point with a cluster finding algorithm (DBSACN - Density-Based Spatial Clustering of Applications with Noise) (more on the specifics of the algorithm below).
Because of that, as of this week, the code of the algorithm to find the exit of the room has been completely implemented (although needs checking and refactoring) and can

be viewed on our GitHub page.
As a reminder for the exact algorithm, we implemented:

1) Remove y coordinate of point (it gives us no information and generally useless)
2) Find 4 lines that represent walls of room using the RANSAC algorithm, and remove all points that reside on walls
3) Find quadrilateral that represents the room with the four lines (we just go over all possibilities of intersection points of lines to achieve this)
4) Remove points inside quadrilateral
5) Find point clusters with DBSCAN of remainder points
6) Remove noise from dataset

The remaining points should be where the exit points are.

While writing the code we've encountered many difficulties such as:

1) formatting functions inside the class so that writing the final main function (getExitPoints) would be very simple to implement and refactor in the future if needed
2) implementing the dbscan algorithm in a non-messy way (easier said than done)
3) writing the code in an environment we can't test it, because of that the code must have been written in a formatted way from the get-go for fixing issues that might occur later

Additionally, in the code there are some minor issues that need to be fixed, though it still might (with medium probability) work in the simulator in its current state as well.

General plans until the due date:

Color coding: yellow: requires the simulator, cyan: doesn't require the simulator.

The following are general goals for a continuation of the project, we won't do it all, more on that further below. Each section is ordered from most important to least important.

1) Get the simulator to work with our exit-finder code
2) Purge the unhelpful stuff
3) Visualization
4) Tests
5) Use RGBD in ORB-SLAM instead of monocular
1) Make a final report that includes details we dismissed from the weekly reports
2) Refactor the current code
3) Add our code to GitHub
4) Write some helper bash scripts

5) Implement a trivial exit algorithm which returns the furthest point
6) Functionality for random start point & angle size

This week we'll focus on:

1->3

4->5->6

The reasoning for that is that some tasks require us to finish other tasks beforehand.

Additionally, we'll Liam what next can we do given the new time limit (also on Dan's zoom).

WE WOULD LIKE TO MENTION THAT EVEN IF DIDN'T GET A NEW SUBMITTION DATE WE WOULD STILL HAVE FINISHED ON TIME. THOUGH THE FINAL RESULT WOULD HAVE LOOKED LESS POLISHED AND REFINED.