Meeting with Liam:

In our zoom meeting with Liam, and in some following discussions we had between ourselves, we decided on the following:

1) The final definition of the project's framework: We will work solely in the simulator, according to Liam, we don't have nearly enough time to use actual drones or cameras, not to speak of TOF cameras.
2) The goal of the project: using ORB-SLAM data, map the room and get TOF data (i.e. get the distance to each sampled point), find an exit point and navigate there.
3) This will be done in the following steps:
   a) Sample points in the room by rotating the camera in place.
   b) If more data is needed move a bit and repeat.
   c) Use an algorithm (discussed further down) to construct the walls of the room.
   d) For each point, calculate its distance from the camera.
   e) Use the above data to fins an exit point.
   f) navigate to the exit point.

Simulator & ORB-SLAM:

We did two major things this week regarding the Simulator & ORB-SLAM, we got familiar with it, with the source code, with how things work, with the things we can change and with the things we can't. We also wrote some code that manages to rotate the camera in the direction of a given target angle and move towards it (the last step). Though, when trying to run this with the current position obtained by simulator.getCurrentLocationSlam() we ran into some problems, we managed to track down the source of the problem but haven't managed to solve it just yet (This is one of the things we'll work on the upcoming week). Additionally, it seems that sometimes ORB-SLAM fails to initialize the current frame, this was probably caused by one of our changes so we will be able to easily revert this when the time comes using the original source code in the GitHub repository. Liam gave us the distance calculation code and the sampling points by rotation code appears in runSimulator.cpp. That means that all we have left to do is implement steps c and e and integrate the above into a single source file. Using the knowledge and experience we gained this week, we believe we will manage to complete the above tasks on time.

Wall-and-exit-room-finding algorithms:

First, we have some current assumptions for the algorithm:

1) The room is in the shape of a convex quadrilateral.
2) The room was scanned completely.

3) The point cloud we have gotten from the room scan has some points outside the quadrilateral which represents the exit' location.

The algorithm itself has 3 major steps:

1) Convert the data to workable 2d.
2) Determine the shape of the room.
3) Determine where the exits are.

The first step:

Notice that the y (height) coordinate of point cloud is irrelevant, as the height of the points in the point cloud does not give us any important information. Because of that we can easily convert our problem to one in a 2d plane in one swift iteration.

The second step:

To find the shape of the room we first find the lines where the walls of the room lie on. To find the lines we use an algorithm known as RANSAC (Random Sample Consensus). The algorithm samples 2 points and determines how many points are close to it (by some defined range), we iterate multiple times, and return the line that counted the largest amount of close point. We repeat this scheme 4 times, and after each time remove the points that are close to the line. We then find the intersection points that create a convex quadrilateral (There can only be one). This convex quadrilateral represents the room.

The third step:

We first remove all points that reside inside the quadrilateral. From here on we only need an algorithm that detects clusters. There are multiple ways to determine clusters, but we have thought of a way that uses the TOF data:

Since exit points will be further away from the camera relative to the walls, we can use TOF data to find points that are far away. Points that are far away and are not inside the convex quadrilateral (i.e. the room) are gathered into clusters defined by the distances between the points in the cluster.

As for the actual implementation of the algorithm, we've managed by now to code the algorithm up to the RANSAC stage.

Our plans for the upcoming week:

1) Integrate the existing codes into a single source file.

2) Finish the Wall-finding-algorithm's implementation.

3) Continue thinking on how to find the exit point (using the depth parameter).

4) Fix any current problems with the simulator.

In what do we need help:

We’d like to have a precise due date and expectations of the result of our project this semester.