Individual Lab Report 10

By Clare Cui

Team B: Arcus

Clare Cui

Maitreya Naik

Angad Sidhu

Logan Wan

6 April 2017

Individual Progress

Continuing with the visualization work I did for the last progress review, my job over the past two weeks was to get the mapping and visualization pipelines connected. Although I had originally said that I would be working on observation synchronizer, this role was reassigned to Logan, who had been working more with the sensor simulations. Logan took care of getting the colorized point cloud points from the RGB camera information, so I started out by pulling his work from the repository and following the chain of functions that was called from sending a point cloud to the DenseVoxelMap package.

From my previous work, I had accomplished maybe 25% of the total work that needed to be done to visualize everything. Logan's point cloud was immediately published inside MapUpdater, but it needed to be fed into DenseVoxelMap. To do this, I had to create another function to handle RGB color information using sensor_msgs::PointCloud2 that would transform the point cloud from the LiDAR frame to the body frame of the UAV. In fact, almost every function that was used after sending the point cloud to DenseVoxelMap needed to be rewritten to include RGB information, so I became very familiar with changing sensor_msgs::PointCloud function calls to their equivalents in pcl::PointCloud and converting to and from sensor_msgs::PointCloud2. The process of visualizing the occupied point cloud in ROS involves transforming the point cloud into the global frame, making sure it is within the bounds of the simulated environment, updating the occupancy status of the grid voxels, and then publishing the occupancy grid point cloud.

The occupancy grid map colorized with the RGB camera data can be seen in Figure 1. As the UAV explores the space, the map gets updated with cells that are seen as occupied. Since the camera has a limited field of view, colors are assigned for a small subsection of the voxel grid map each time it is updated. Occupied voxels that have not been viewed by the camera are a default white color. It appears that this simulated UAV flies conservatively, staying within a small area and not turning around to view the whole map, so only a few voxels ever get colored. Additionally, it can be noticed that several of the colored voxels are not actually conforming to the contours of the pit, particularly the dark green voxels in the center of the image. The reason for this is due to hard-coding a fabricated camera intrinsics matrix, which transforms the points away from where they actually are. We will update this with a yaml file that has the correct camera intrinsic parameters when we begin integrating what we have achieved in simulation with real bagged data.

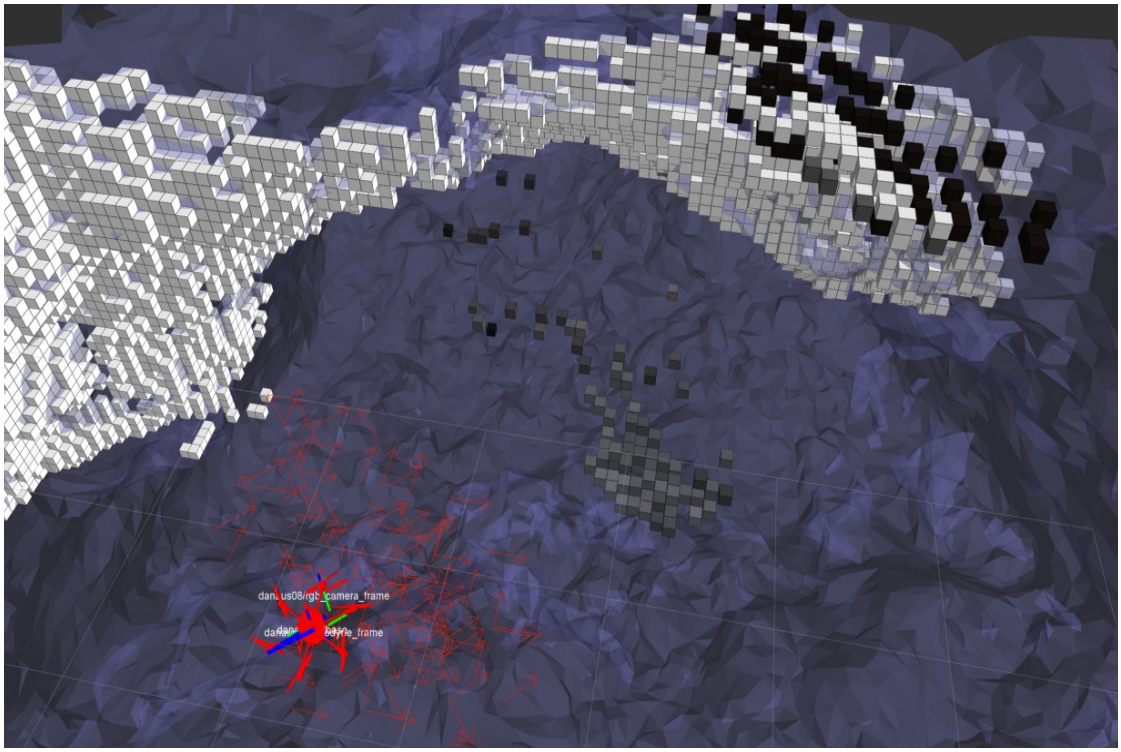

Figure 1: *Occupied dense voxel map in simulation created by a LiDAR with RGB information from camera.*

Challenges

I experienced significant challenges this week debugging issues with the code. The easier problems to debug were the errors that the compiler gave, which were usually due to syntax or improper use of a function. However, when I was first able to successfully build the simulation stack, the drone disappeared after flying for a brief moment, and I could not visualize any of the sensor topics. I worked with Angad to see what was wrong, and we basically depended on print statements to see where the simulation was breaking. There were two critical parts that were preventing the simulation from running. One was that, in pit_exploration_planner, I had mistakenly changed a subscriber from the incremental_pointcloud topic to the occupied_RGB topic. This was a continuation of the line of thought that everything needed to be changed to accommodate the RGB cloud. However, this was unnecessary as the planner was just checking for collisions at this stage and did not need any RGB color information. Additionally, the RGB point cloud message was not the same type as the incremental point cloud, so this was also causing an issue. After discovering this, we were able to corner the next major issue, which was that I had been trying to send a point cloud to the DenseVoxelMap inside a loop that was looping over individual points. So, essentially, I had been trying to register extremely small point clouds that were getting bigger iteratively, instead of just sending one large point cloud

with all of the points. Once this was changed, I was able to get the simulation to work once more fairly easily. From this experience, I learned that it is important to know exactly what it is you are changing before you do anything with it – which has been a challenge in and of itself for me as I have been getting familiarized with C++ – and how to debug more efficiently.

Teamwork

Logan Wan: Logan worked on reading over and understanding the observation synchronizer, synced up the sensor messages, and helped me with some of my simulation issues when they first started.

Angad Sidhu: Angad determined the bug with the trajectory generator to be a time synchronization issue and helped me greatly with the simulation issues I was experiencing.

Maitreya Naik: Maitreya brought up the sandbox on the team laptop and continued looking through code for the motion manager, trajectory generator, and action library.

Future Plans

In the next two weeks, I am planning to work on getting mapping in Rviz up with real bagged data. We are also hoping to go to Lafarge Quarry to collect more data to help prepare for SVE. If I achieve mapping with bagged data, I will also try to work on simulating column obstacles to test the trajectory generation.