Harikrishnan Suresh

Team C: Fly Sense
Teammates: Shivang Baveja, Nicholas Crispie, Joao Fonseca, Sai Nihar Tadichetty
ILR04
Nov 10, 2017

Individual Progress

Following my work on octomapping algorithm and its optimization, I tested the algorithm on a LIDAR dataset provided by Near Earth Autonomy. However, the map generated was not accurate and updated very slowly. This raised serious questions about the robustness of the algorithm and whether it would work for our case with an actual flying vehicle. Since the mapping and obstacle segmentation is a very critical aspect of our project, I decided to test another algorithm (grid mapping) developed at the AIR lab, CMU.

The grid mapping code was not open source, and did not have any online documentation. So, it took me a few days to understand the underlying algorithm and its implementation. After figuring out all the dependencies and obtaining different support packages from the lab's bitbucket repository, I was able to develop a working version of the code.

Since the grid mapping visualizer code provided by AIR lab did not work, I worked in collaboration with Nick to store the published obstacle lists and visualize the data offline (not real time in ROS but in python). The data when visualized did not give the expected output. The obstacle list data was found to grow sparser and sparser with time, which contradicted the whole "obstacle list" term. The same problem was conveyed to our mentor who provided few clarifications on the code. The output getting published was not a complete list of obstacles at every time step, but a list of "updates" (which is why it was naturally getting sparser over time). The same has been shown in Figure 1 and 2. The actual output of interest for us was stored in a shared memory, which could be accessed by another node separately. Following the suggestions received by him on how to obtain the relevant data from shared memory, I started working on a new node that would access all the cells in the memory location of grid map and return the cell occupancy state.

Along with this, I also created a package for "flysense mapping" which would take in raw point cloud data, tf and odometry information and publish the topics required by the grid mapping algorithm, specifically the point cloud hits/misses, world frame, drone frame and sensor frame.

I also got 2 data sets from NEA - one with registered point cloud data and the other with odometry information. The point cloud data was being published as a topic and seen in the terminal, but not working on Rviz. After consulting with one of the robotics engineers, I learnt that the data must be converted from double to float to visualize in Rviz, else a position transform won't be received. I wrote a small code that would perform the required conversion so that the data can be viewed in Rviz.

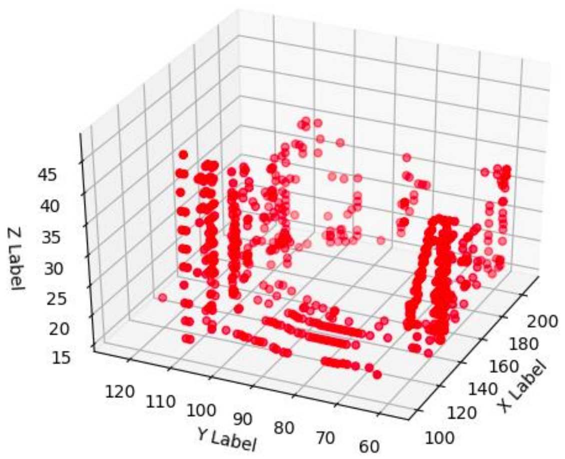
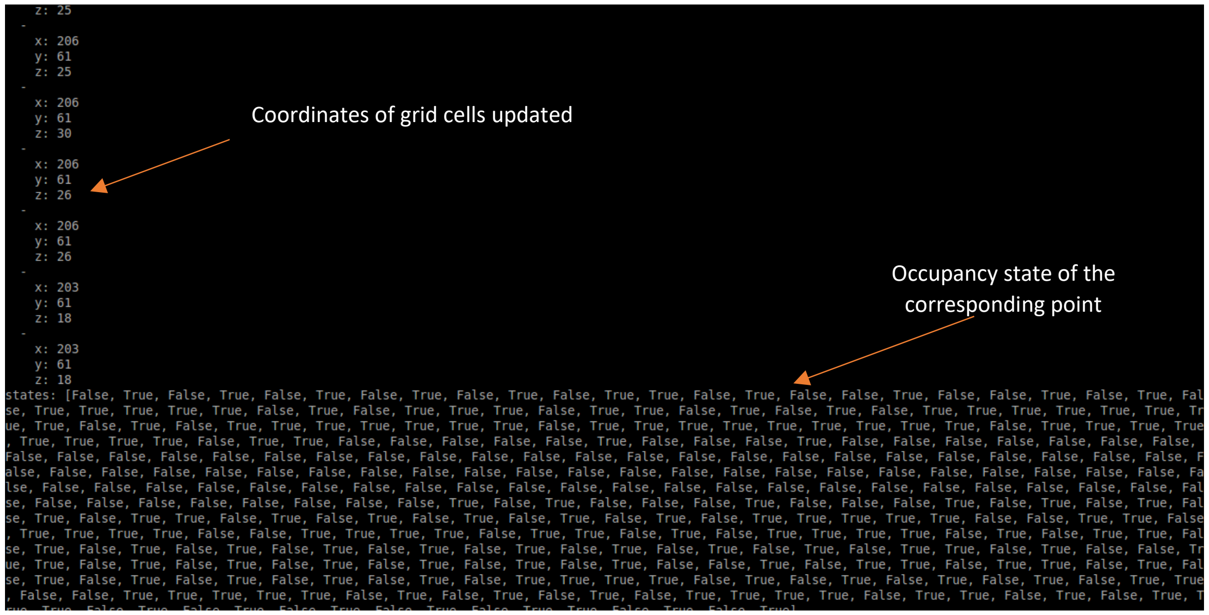


Figure 1: Obstacle list published by the code (visualized offline in python)


Coordinates of grid cells updated
Occupancy state of the corresponding point

```
    z: 25
  -
    x: 206
    y: 61
    z: 25
  -
    x: 206
    y: 61
    z: 30
  -
    x: 206
    y: 61
    z: 26
  -
    x: 206
    y: 61
    z: 26
  -
    x: 203
    y: 61
    z: 18
  -
    x: 203
    y: 61
    z: 18
states: [False, True, False, True, False, True, False, True, False, True, False, True, True, False, True, False, False, True, False, False, True, False, True, Fal
se, True, True, True, True, True, False, True, False, True, True, True, False, True, True, False, True, False, True, False, True, True, True, True, True, True, Tr
ue, True, False, True, False, True, True, True, True, True, True, True, False, True, True, True, True, True, True, True, True, False, True, True, True, True
, True, True, True, True, False, True, True, False, False, False, False, False, True, False, False, False, True, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fal
se, False, False, False, False, False, False, False, False, True, False, True, False, False, False, True, False, False, False, True, False, True, False, True, Fal
se, True, False, True, True, False, True, False, True, False, True, False, True, False, True, False, True, True, True, True, True, False, False, True, True, False
, True, True, True, True, False, False, True, True, True, True, False, True, True, False, True, False, True, True, True, True, False, True, False, True, True, Fal
se, True, False, True, False, True, False, True, False, True, False, True, False, True, False, True, False, True, False, True, True, False, True, False, False, Tr
ue, True, False, True, False, True, False, True, False, True, False, True, False, True, False, True, True, False, True, True, False, True, False, True, Fal
se, True, False, True, False, True, False, True, False, True, False, True, True, True, True, False, True, False, True, False, True, False, True, False, True, True
, False, False, True, True, True, True, True, False, True, False, True, False, True, False, True, False, True, True, True, False, True, True, False, True, False, True, T
rue, True, False, True, False, True, False, True, False, True, False, True, True, False, True, False, True, False, True]
```

Figure 2: Terminal output for obstacle list topic

Challenges faced

1. Understanding a third-party code without any online documentation was really time consuming for me. I also found it difficult to understand the governing algorithm, but was facilitated by a few valuable inputs from our mentor.
2. Even though the main code was working, I was not able to visualize the output. During the weekly meeting with our mentor, I was provided with a new package was used solely for visualization purpose.

3. The grid mapping algorithm does not work on simple publisher/subscriber model but uses shared memory with locks and semaphores. As I am relatively new to such level of programming, it took me some time to interpret various portions of the code and debug.

Team work

Nick and Shivang assisted me few days before the progress review to get the grid mapping code to work. Nick created a small code in Python to visualize the output of grid mapping code as the air lab visualizer did not work.

Shivang worked on developing a complete software architecture for our project. He also implemented a point cloud filter in ROS based on PCL CropBox which would allow only the point cloud hits within the dynamic window to enter the mapping node. He also coordinated with Nihar to establish communication between the AR interface and Onboard computer. He also worked on retrieving pose information from the DJI matrice using ROS and DJI SDK.

Nick completed the design of Power Distribution Board and FVE testing platform. He also coordinated with Shivang to perform few ground tests with the DJI matrice and acquire data from the sensors.

Joao was involved in developing a dynamic window for the quadcopter, which will be very critical to the point cloud filter and obstacle segmentation. He developed a C++ code implementation for the same.

Nihar successfully deployed an upgraded version of the HUD with artificial horizon, attitude variables and scrolls containing dummy data in accordance with the final version. He also set up a GUI that displays the incoming messages from the Onboard computer.

Future work

Since the mapping algorithm is not completed as per schedule and the challenges with the AIR lab code are still not solved, a decision on the final mapping algorithm to be implemented and its achievable quality will be taken. If the AIR Lab code does not work, I will be proceeding with octomapping algorithm. The preprocessing node will then contain few more Point Cloud Library codes running in a pipeline to improve the efficiency of octomapping. Considering this delay and its dependencies with some of the other subsystems, I will be investing all my time to get a working version of obstacle mapping by the next progress review. It will be shown working on the Onboard computer.

Our team will be working on getting all the subsystems individually working. We will be working on the first version of sound warnings and speech recognition. We will also work on getting the next version of HUD with live data from the DJI. Once the obstacle mapping is complete, it will be integrated with the AR interface and shown as Bird's Eye View.