Individual Lab Report 8

Progress Review 9

Pulkit Goyal

March 1, 2018

Team F - Falcon Eye
Pulkit Goyal
Pratibha Tripathi
Yuchi Wang
Rahul Ramkrishnan
Danendra singh

Individual Progress

I mainly contributed to segmenting out the relevant obstacles in front of the Husky from the point cloud in realtime and then avoiding those obstacles using Husky's control Stack. Rahul worked on this along with me.

1. Segmenting out the relevant obstacles out of the entire point cloud of Lidar

I primarily contributed to segmenting out the localized obstacles from velodyne Lidar's point cloud along with Rahul. We used PCL to process the Velodyne data cloud. We were using this onto a saved pcap file, but in this progress review we implemented this onto a real time point cloud.

2. Removing outliers based on condition

As mentioned in previous ILR (i.e.ILR07), using the entire point cloud for segmenting out the obstacles was computationally very expensive. As we have i5 Mini-pc on-board Husky robot, which has limited computational power so we decided to reduce the point cloud before processing it to find obstacles. To start we reduced the size of point cloud by removing all the points which are beyond certain distance from the robot. We evaluated both the methods available in point cloud library which is outlier removal using conditional or radius outlier removal. Finally, we used Removing Outliers tutorial with a Condition tutorial from point cloud library to achieve this.

The useful and relevant points in the cloud were filtered out based on two condition, the field of view from 0 to 210 degrees in front of Lidar/Robot and points in vicinity of 10 meters around the robot. We removed all the point beyond these conditions as they were not helping us with the motion planning of the robot.

When we implemented the same approach onto the point cloud we were getting from the velodyne in real-time, we faced the following issues:

3. The rate of getting the cloud from Lidar was greater than processing time of single point cloud

We used ROS subscriber to subscribe to the point cloud published by velodyne driver velodyne/pointcloud. As velodyne has a rotation rate of 5-20 Hz. We were sure that we are running obstacle segmentation is fast enough to keep up with the rate at which velodyne driver is publishing the pointcloud. But the node started mi-behaving. Then I decided to use multi-thread function of the subscriber node. But 2-3 hours into the implementation I realized that multiple threaded application means i'll have to take care of shared memory with help of mutex which can make the problem complex. Pondering over some simple solution I realized that we can drop some point clouds alternately, while processing one of the point clouds and move the vehicle in little bit slower speed. After much of a experimentation we chose 2 as number of clouds we will drop while processing other data. This number was good with the speed with which we wanted to move the ground robot and also we could successfully segment the obstacles from the point cloud of moving vehicle. I designed a subscriber which drops 2 clouds and process the third one.

4. The segmented area was not matching the area physically present in front of Lidar

In order to segment out the required area from the entire point cloud, we were required to ignore certain point from the entire point cloud. Lidar throws out points w.r.t. axis which is centered at the Lidar Sensor. It gives points in spherical coordinate (r,w, alpha) system, which are then converted to X, Y, Z coordinate system by PCL. The axis of Velodyne Puck are defined as shown in Fig. 1 below.

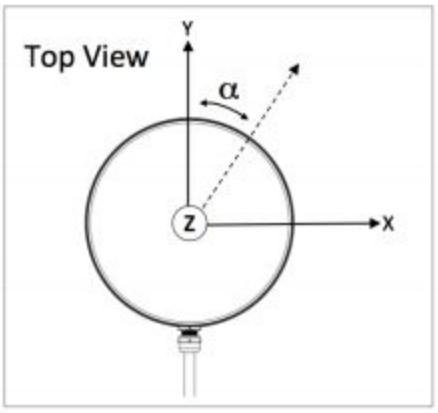

Fig. 1 Coordinate system given in Velodyne Manual

It gives data as far as 100m from the sensor, this data was clipped down to 10 meter as we wanted to detect in vicinity of robot we also wanted the sensor to have a field of view of 0 to 180 degrees. When we ran this on the live data it was segmenting out the obstacles in the axis which was shifted by 90 degrees than the actual axis. As shown in fig 2, we wanted data in +x+y and -X+Y axis but the data we were getting was in -Y-X and +Y-X. We spent a lot of time debugging different transformations in the code believing the axis direction given in the Velodyne manual. After referring to different forums we realized that the data given by the ROS Lidar driver is inverted by 90 degrees w.r.t. To the axis given in the manual.

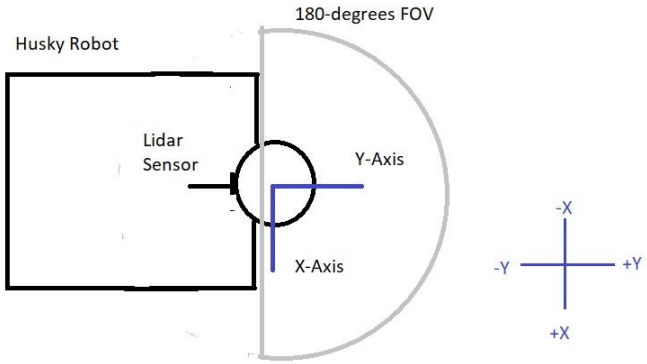

Fig2. Axis as defined in the manual

5. Getting Nan data from point cloud data

After resolving the axis misalignment problem we tested the robot by moving it using the remote control checking if it's able to segment out the obstacles properly in real time from the point cloud while moving. We noticed that it's not publishing the obstacles at the rate at which it should given the frequency of point cloud. As per calculation, if obstacles are placed in FOV of the Lidar it should publish obstacles at the rate of 2 per second as per the implementation done in point 1. This is when we started visually reading the data from the Velodyne on the terminal, this is when we realized it was throwing a lot of NAN in place of x,y,z values in the point cloud data. Looking on the internet we realized that most of the time velodyne publishes just NAN data instead of point cloud.

6. Control stack of the Husky, Reactive approach

Current control stack of Husky works with GPS and IMU. It continuously keeps on checking the difference of its global goal's position and orientation w.r.t. Values from GPS and IMU. So according to the control stack robot moves certain x,y, theta for certain time step and then checks the difference of its current position and orientation using the on-board sensors w.r.t. To global goal and then decides the next step.

According to reactive approach, as soon as we detected obstacle in the robot's FOV we overrode the control stack and made the robot move to a Pseudo goal orientation and position which takes the robot away from the obstacle. Moving towards the Pseudo goal position will involve moving robot +90 or -90 degrees(depending on position of obstacle in front of it) and then moving it forward equal to the length of the obstacle. While moving in short time steps robot keeps on checking if the FOV in front of it is obstacle-free, if it's not it will repeat the same steps. After executing this motion control is given back to the normal control stack until a obstacle is encountered.

7. Overall Software Stack

We subscribed to the point cloud data published by velodyne driver. This node is responsible for dropping some packets and segmenting out the obstacles which are then published on a topic. This topic containing segmented obstacles, which is maximum and minimum X and Y values of the obstacles, is subscribed by the control stack.

Challenges

1. We lost one of our drones as it got stuck on the tree, we invested a lot of time trying to bring it down.
2. Shift in axis by 90 as explained above, consumed a lot of time to debug.
3. Getting NAN values from the velodyne data as explained above.
4. It's definitely challenging chasing different timelines, feels like we need to manufacture time in order to cover up everything.

Teamwork

1. Danendra and Pratibha worked on setting up the IMU with the control stack of the Husky robot.
2. Rahul diligently worked on incorporating obstacle avoidance in Husky's control stack along with me.
3. Yuchi worked on incorporating exploration algorithm in drone's control stack along with working on placing a drone on the tree in the process.

Future Plans

We specifically plan to work on the following tasks. Integrating the different modules to work together which involves the following:

1. Bebop sending the goal location to husky over the WiFi network.
2. Developing the Husky's control stack to move progressively on short term goals given by Bebop.
3. Husky avoiding obstacles while moving to the short term goals.