Autonomous Vehicle Project Report

Wanting JIN and Xirui LIU

January 17, 2019

Tools: Matlab, KITTI data, KITTI development kit

1 Needed Data

$Lidardata(velodyne\_points)$: the coordinates of the detecting object in lidar frame ${}^{Velo}T_{obs}$
$GPS/IMUdata(oxts)$: the GPS data in global frame ${}^{G}T_{IMU}$
$Calib\_imu\_to\_velo$: transformation matrix from IMU to VELO ${}^{IMU}T_{Velo}$
$Tracklet\_labels$: labeled 3D location and rotation
$Image$: picture taken at each timestamp

2 Data Processing

Listing 1: Get pose; tracklet; lidar data from raw sensor information

```
file_path='data/2011_09_26_drive_0046_sync/';
oxts = loadOxtsliteData(file_path);
pose = convertOxtsToPose(oxts);
tracklets = readTracklets([file_path '/tracklet_labels.xml'])
for i=1:size(pose,2)
    fid = fopen(sprintf('%s/velodyne_points/data/%010d.bin'
    ,file_path,i-1),'rb');
    velo = fread(fid,[4 inf],'single')';
    velo = velo(1:20:end,:); % take only 1/20 of the total data
    fclose(fid);
    scan = velo(:,1:3);
end
```

3 Build Occupancy Grid

We build the function **myMap=occGridMapping(myMap,local_occs,pose_robot,param, T_IMU2velo)** to build the occupancy grid map and update the map once the new measurement come in. *myMap* is the occupancy map calculated in last time iteration. *local_occs* is the lidar data which contains the coordinates of the obstacle in lidar frame. *pose_robot* is the pose data gathered by IMU expressed in global frame. *param* contains param about the value of *param.origin*, which is the robot origin coordinates in the map, *param.reslo* is the resolution of the map (1 meters contains how many grids)*param.lofree* and *param.looccu* will introduce later.

As the measurement contains many noises, we will calculate the state of one grid according to all the past measurements. From what we study in class we have got that:

$$l\left(m_i|z_{1:t}, x_{1:t}\right) = l\left(m_i|z_t, x_t\right) + l\left(m_i|z_{1:t-1}, x_{1:t-1}\right) - l\left(m_i\right) \tag{1}$$

where $l(x) = log\dfrac{p(x)}{1-p(x)}$, $p(x)$ stands for the possibility that a grid is occupied, by using the form of $l(x)$, the only item contains the current measurement is $l\left(m_i|z_t, x_t\right))$. We use $l(x)$ to represent the state of a grid, then we can update the state by using $S^+ = S^- + lomeas$, where $lomeas$ comes from the measurement result as $lomeans = lofree$ if $z_t = 1$, $lomeans = looccu$ if $z_t = 0$. In this way, we can update the state of the map by only computing the sum, which is highly efficient.
We can use the matlab function *(i,j)=(ceil(x/r),ceil(y/r)* to transfer the coordinates of the real world to grid map. Once we detect an object at (x_o, y_o), we take the measurement of that grid as $z_t = 0$ and the grids lying on the line between the occupied grid and the robot are free with $z_t = 1$. We use the Bresenham's algorithm to get the set of free grids, which is shown in figure 1.

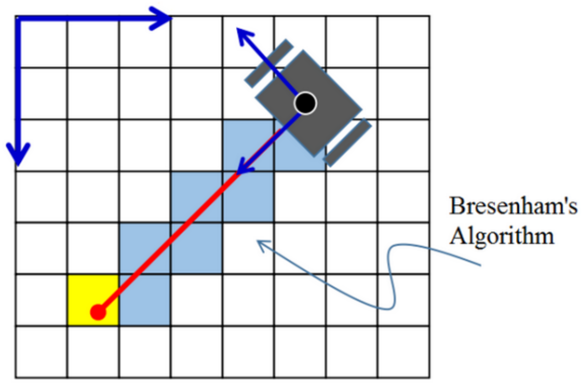

Figure 1: Bresenham method

Once we got the occupied set and the free set of the grids, we need to transfer it coordinates in to global frame by using:

$${}^{G}T_{obs} = {}^{G}T_{IMU} * {}^{IMU}T_{Velo} * {}^{Velo}T_{obs} \tag{2}$$

4 Add label box and car trajectory

we build function **draw_tracklets(frame, tracklets, pose,origion,resol)** and **drawCarTra (frame,pose,origion,resol),T_IMU2velo** to draw the box for tracklets and robot as well as robot trajectory. The tracklet xml data stores poses relative to the first frame where the tracklet appeared. We first transmit this coordinates to the velodyne coordinates by calculating the transformation between current frame and the first frame. Finally, we can get the coordinate of the detected object in global frame by using Equation 2. We calculate the accordingly coordinates in grid map by:

$$cor\_grid = cor\_global * param.reslo + param.origion \tag{3}$$

Then we can draw the box around the corner of the object. By doing the similar process, we can transfer the robot pose into grid map and draw the box and trajectory.
The result after several time intervals is:

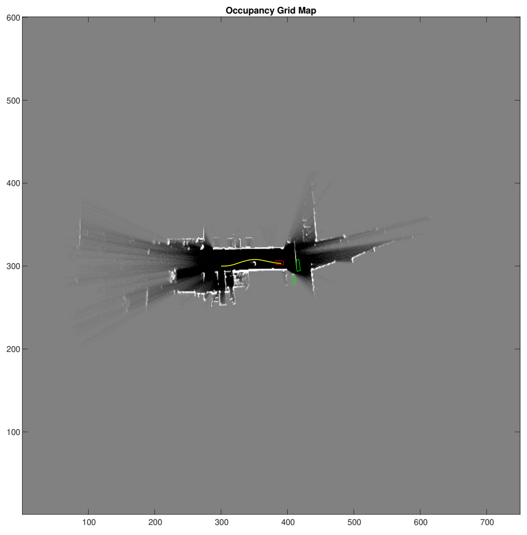

Figure 2: Occupancy grid map after several iterations

5 Conclusion

We can add the scene image and the instantaneous 3D point cloud, to compare the result. A movie is attached to see the result.
Something to improve:
We need to initially set the size for the occupancy grid map. Sometimes the lidar detection could go beyond the range of the matrix if this function is applied to a long navigation scenario. We can improve the function by changing the size of the map according to the size of the scenario or moving the origin of the map with respect to the robot position.