



Summer Internship Program 2020-21

MapLite :autonomous navigation without
detailed prior maps

By

Awies Mohammad Mulla

Abstract

Most autonomous vehicles currently rely on high-definition prior maps in order to localize and navigate in their environment. In this project we aim to localize the vehicle using only onboard sensors and sparse topometric map which is publicly online. We tested this algorithm on indoor (Ati) and outdoor (KITTI) datasets to generate a pre-driven trajectory. The performance of the algorithm can be quantified by level of similarity between generated trajectory and groundtruth.

Introduction

Localization is an integral part of autonomous driving, where the vehicle has to estimate its current pose on the map. It is also a common process of Simultaneous Localization and Mapping.

Over the years, huge amount of study has been done to solve the problem of localization and SLAM as a whole. Some of the works uses geostationary satellites to pinpoint the location of the vehicle. Although compared to initial times, this approach has become quite accurate populated areas, there are still remote places where using this method is not good option due to bad signal. Other works mention combining readings from the odometry sensors with LiDAR and/or visual sensors (like stereo cameras) using algorithms like Kalman filters, Particle filters, etc to give final estimate of vehicle pose. The mentioned algorithms have been largely evolved over past decade and are used extensively. For large numbers of states in a system Kalman filters are usually the first choice, and Particle filters are used for lower number of states due their low runtime. But while working with these algorithms, we often require maps with accurately positioned landmarks.

Generating High definition metric maps consumes huge amount of time and resources. So, it is inefficient to generate such a map for a remote area with low population density and no appropriate landmarks. Hence, the maplite is a suitable option in such situations as it uses a topometric map and road segmentation for localization.

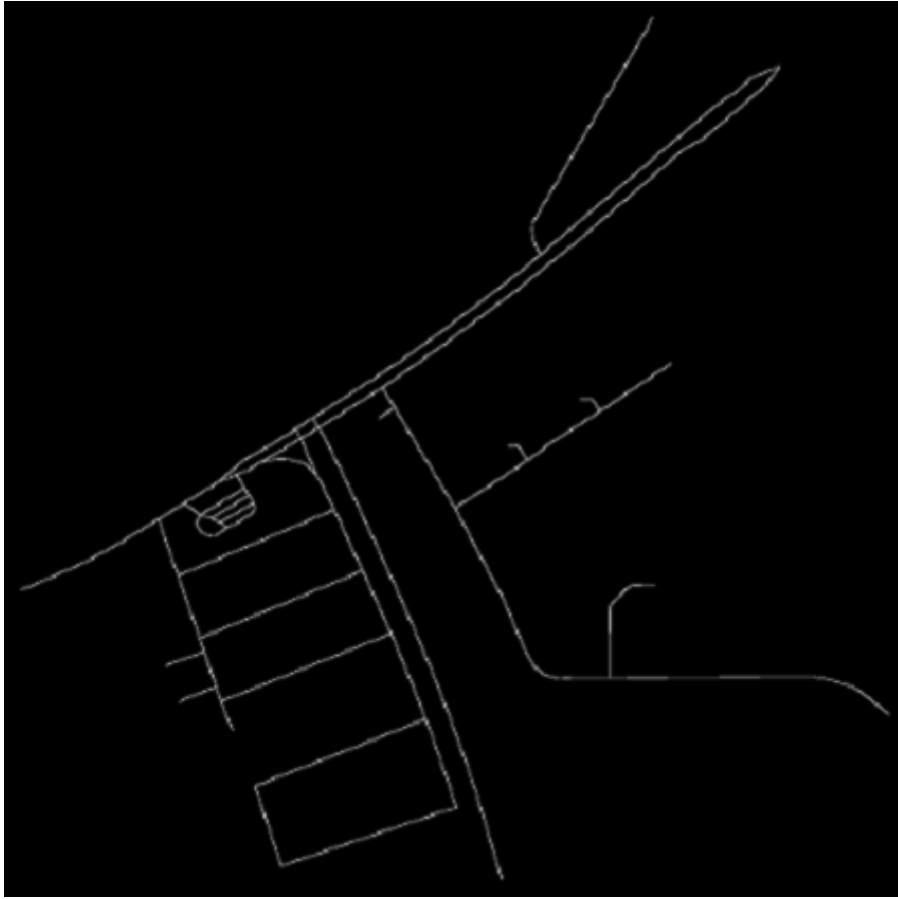
Goal of this project is to localize the vehicle in real-time (given initial pose) using, LiDAR, IMU, wheel encoder and a topometric map of the location.

Maplite

This algorithm aims to localize the the vehicle using odometry sensors, LiDAR, and topometric map of the location. The topometric maps are simple graph-like data structures. Hence, for each map M

$$M = \{V, E\}$$

where, each vertex describes a way point and each edge describes a road segment. However, while the connectivity of the network can be assumed to be relatively correct, the same cannot be said for either the relative or global accuracy of the specific waypoints. Compared to high definition metric precise maps, less time and effort is required to generate topometric map. They also require far less storage capacity. For testing the algorithm on KITTI dataset we have used the map available on openstreetmap.org for the coordinates mentioned in datasets. An example of topometric map is shown below



Algorithm

The algorithm is mainly carried out in two steps:

1. Estimating the prior distribution or Odometry-based prediction
2. Correction using LiDAR Observation likelihood

Odometry-based prediction

We estimate a prior distribution in this step using an Extended Kalman Filter given initial pose of the vehicle, input provided and measurements from wheel encoder and inertial measurement unit (IMU). The implementation of the Extended Kalman Filter is explained below:

Pose of vehicle at time t is \mathbf{x}_t and input provided is u_t . Covariance matrix at time t is P_t and noise by sensors reading the input (wheel encoders) is indicated as σ_u .

$$\begin{aligned}\mathbf{x}_t &= [x_t, y_t, \theta_t] \\ u_t &= [v_t, \omega_t] \\ P_0 &= \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \\ \sigma_u &= \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}\end{aligned}$$

EKF Prediction step

$$\begin{aligned}\hat{\mathbf{x}}_{t+1} &= \mathbf{x}_t + \begin{bmatrix} v_t \cos \theta_t dt \\ v_t \sin \theta_t dt \\ \omega_t dt \end{bmatrix} \\ Q &= G \sigma_u G^T \\ \hat{P}_{t+1} &= F P_t F^T + Q\end{aligned}$$

where, $F = \begin{bmatrix} 1 & 0 & -v_t \cos \theta_t dt \\ 0 & 1 & v_t \sin \theta_t dt \\ 0 & 0 & 1 \end{bmatrix}$ and $G = \begin{bmatrix} \cos \theta_t dt & 0 \\ \sin \theta_t dt & 0 \\ 0 & dt \end{bmatrix}$ (F is jacobian of $\hat{\mathbf{x}}_{t+1}$ w.r.t. states)

Q = process noise, here $\sigma_v^2 = 0.1$ and $\sigma_\omega^2 = 0.01$

EKF Correction step

Linear and angular velocities from the IMU (accelerations obtained from IMU) is obtained in following manner

$$z_t = \begin{bmatrix} v_t^x + a_t^x dt \\ v_t^y + a_t^y dt \\ \omega_t \end{bmatrix}$$

Estimated velocities as the function of states is given by,

$$h(\hat{\mathbf{x}}_{t+1}) = \begin{bmatrix} \hat{v}_{t+1} \cos \hat{\theta}_{t+1} \\ \hat{v}_{t+1} \sin \hat{\theta}_{t+1} \\ \hat{\omega}_{t+1} \end{bmatrix}$$

With measurement noise $R = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$

Hence, the correction step is as follows,

$$\begin{aligned}\mathbf{x}'_{t+1} &= \hat{\mathbf{x}}_{t+1} + K(z_t - h(\hat{\mathbf{x}}_{t+1})) \\ K &= \hat{P}_{t+1} H^T (H \hat{P}_{t+1} H^T + R)^{-1} \\ P_{t+1} &= (I_{3 \times 3} - KH) \hat{P}_{t+1}\end{aligned} \tag{1}$$

where, H is jacobian of $h(\hat{\mathbf{x}}_{t+1})$ evaluated at the predicted state ($\hat{\mathbf{x}}_{t+1}$)

$$H = \begin{bmatrix} 0 & 0 & -\hat{v}_{t+1} \sin \hat{\theta}_{t+1} \\ 0 & 0 & \hat{v}_{t+1} \cos \hat{\theta}_{t+1} \\ 0 & 0 & 0 \end{bmatrix}$$

Sampling

Now, sample k points in a circle with radius r_{var} and center (x_t, y_t) . At each of these points consider l distinct orientations in the interval $[\theta - \delta\theta, \theta + \delta\theta]$. Assign weights to each of these points according to following expression

$$(x^i, y^i) \in \{(x_t^1, y_t^1), (x_t^2, y_t^2), (x_t^3, y_t^3), \dots, (x_t^k, y_t^k)\} \tag{2}$$

$$\theta^j \in \{\theta_t^1, \theta_t^2, \theta_t^3, \dots, \theta_t^l\} \tag{3}$$

$$w_{ij} = \exp\left(\frac{||\hat{\mathbf{x}}_{t+1} - \mathbf{x}_j^i||}{b}\right) \tag{4}$$

where, b is a scale factor obtained by parameter tuning. Parameters r_{var} and $\delta\theta$ are to be set by the user. $\delta\theta$ can be either set according to $P_{3,3}$ or be chosen depending on traction present at location. r_{var} can be either set to $\max(P_{1,1}, P_{2,2})$ or $12 - 15\%$ of the road width (r_w). r_{var} can be reduced at the end of each iteration (according to (5)) and be increased again at beginning of next timestep (as uncertainty increases as vehicle moves).

$$r'_{var} = |ln(\frac{sc'}{sc})| \tag{5}$$

where sc' and sc are scores of sampled point and prior estimated mean respectively (mentioned in (6))

LiDAR Observation Likelihood

Each LiDAR scan Z_t^L consist of n 3-tuples $\mathbf{z}_i = (x_i^L, y_i^L, l_i)$ where $x_i^L, y_i^L \in \mathfrak{R}$ give the position of each measured LiDAR point in the sensor frame and $l_i \in \{0, 1\}$ represent classification label indicating whether the point is on road or off-road. We define signed distance function $f_D(z_i, x_i^L, M)$ which represents the distance of point z_i to the nearest point on any edge in the topological map M (given the location of vehicle, \mathbf{x}_t). The probability of observing each point z_i at location \mathbf{x}_t is expressed in terms of sigmoid function

$$P(z_i|\mathbf{x}_t) = \begin{cases} \frac{1}{1 + \exp(f_D(z_i, x_i^L, M) - r_w)} & l_i = 1 \\ 1 - \frac{1}{1 + \exp(f_D(z_i, x_i^L, M) - r_w)} & l_i = 0 \end{cases}$$

where r_w represents likelihood of finding points labeled road, far from map M which contains road centerlines. For example at road center $f_D(z, x) = 0$, $P(z_i) \approx 1$ if $l_i = 1$, $P(z_i) \approx 0$ if $l_i = 0$ while far from the road, the converse is true. In other words, here r_w can be considered as half road width. Depending on the location (type of road or lane count) r_w can varied or kept constant.

Signed distances for all possible (x_i^L, y_i^L) in map M are calculated and stored as a look up table. This is done to improve the time complexity of the algorithm so that it can be used in real-time.

Finally, we calculate score of the current point (sampled point from (4)) using following expression

$$sc = \sum_i P(z_i|\mathbf{x}_t^i) \quad (6)$$

Fusion of Odometry prediction and LiDAR Likelihood

From (2) and (3) we have $(k \times l)$ possible poses

(x, y)	θ
(x^1, y^1)	$\theta^1 \ \theta^2 \ \theta^3 \ \dots \ \theta^l$
(x^2, y^2)	$\theta^1 \ \theta^2 \ \theta^3 \ \dots \ \theta^l$
(x^3, y^3)	$\theta^1 \ \theta^2 \ \theta^3 \ \dots \ \theta^l$
\vdots	\vdots
(x^k, y^k)	$\theta^1 \ \theta^2 \ \theta^3 \ \dots \ \theta^l$

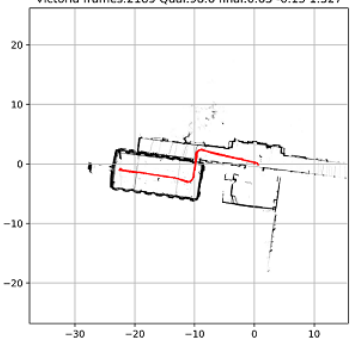
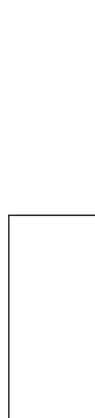
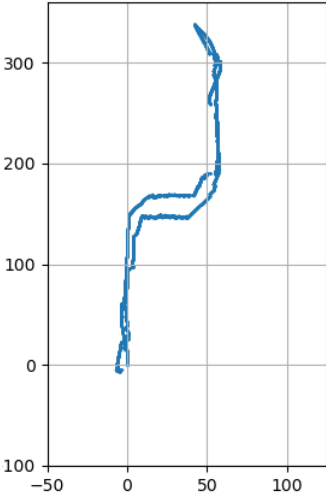
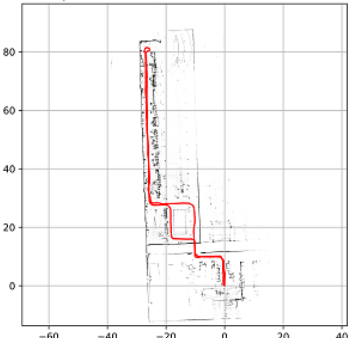
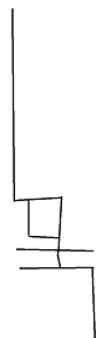
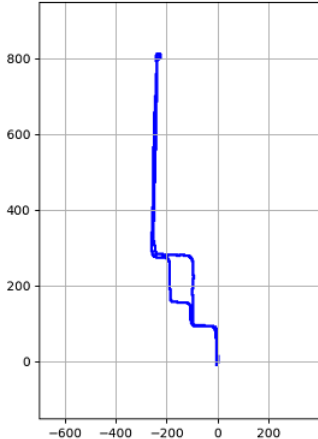
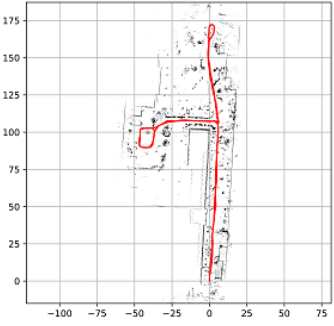
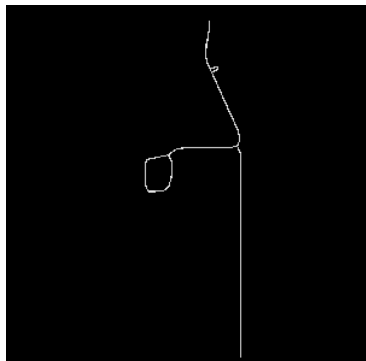
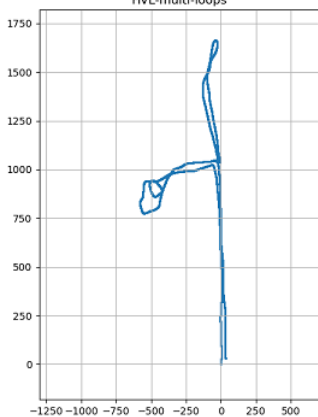
Since, LiDAR scans are obtained w.r.t. sensor frame, we now consider all the above possible poses for sensor and calculate the score (from (6)) for all the above poses. Now, select the pose with maximum score say \mathbf{x}_j^i has score sc with odometry weight w_{ij} (from (4)) and score of estimated mean pose (1) be sc' . We calculate the corrected pose as weighted mean of \mathbf{x}_j^i and \mathbf{x}'_{t+1}

$$\mathbf{x}_{t+1} = \frac{(w_{ij} \cdot sc \cdot \mathbf{x}_j^i) + (sc' \cdot \mathbf{x}'_{t+1})}{(w_{ij} \cdot sc) + sc'} \quad (7)$$

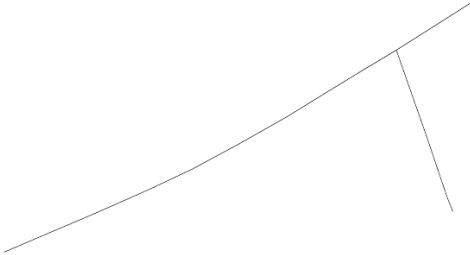
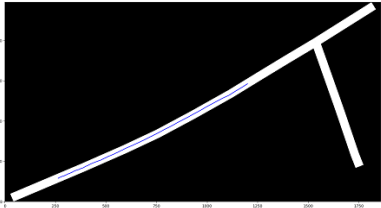
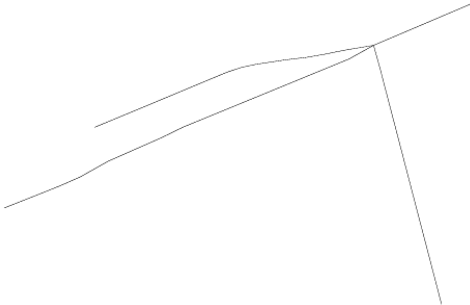
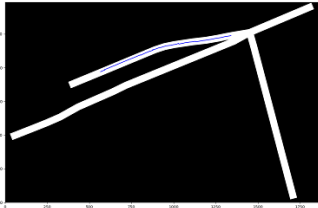
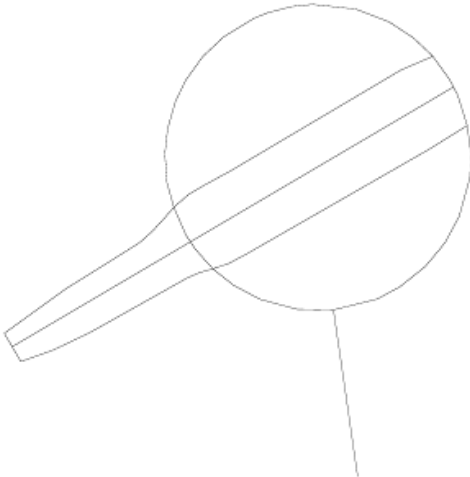
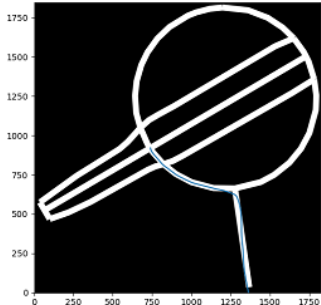
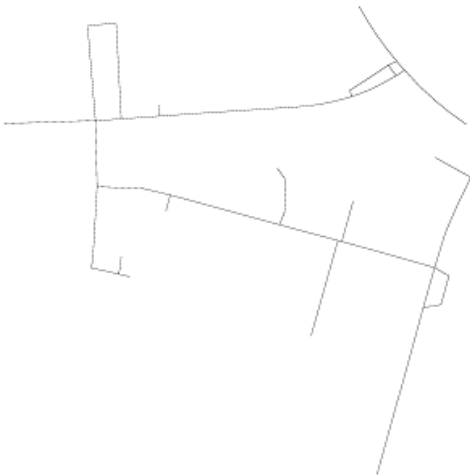
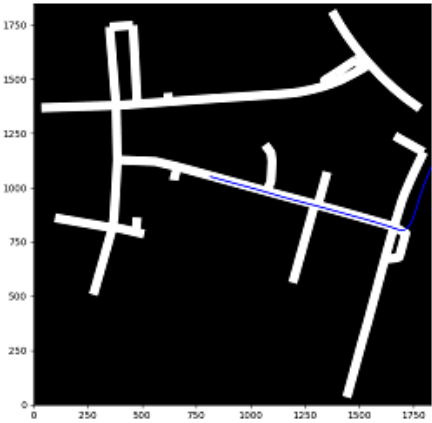
Results and analysis

Mentioned algorithm was tested on 3 Ati datasets and 8 KITTI datasets. Results are shown and explained below:

Ati Datasets:

Dataset name	Original path	Topometric map used	Trajectory generated
Hospital	<p>Victoria frames:2189 Qual:98.6 final:0.63 -0.15 1.327</p>  A 2D plot showing a red path on a grayscale map of a hospital layout. The path starts at the left, moves right, then up and right, ending at the top right. The axes range from -30 to 10 on the x-axis and -20 to 20 on the y-axis.	 A simplified topometric map of the hospital layout, showing the main corridors and rooms as black lines on a white background.	<p>Hospital</p>  A 2D plot showing a blue trajectory on a grid. The trajectory starts at (0,0), moves up to y=100, then right to x=50, then up to y=300, and finally right to x=100. The axes range from -50 to 100 on the x-axis and 0 to 300 on the y-axis.
Factory	<p>factory frames:5355 Qual:98.6 final:-0.43 0.21 -1.563</p>  A 2D plot showing a red path on a grayscale map of a factory layout. The path starts at the bottom, moves up, then right, then up, then right, ending at the top right. The axes range from -60 to 40 on the x-axis and 0 to 80 on the y-axis.	 A simplified topometric map of the factory layout, showing the main corridors and rooms as black lines on a white background.	<p>Factory</p>  A 2D plot showing a blue trajectory on a grid. The trajectory starts at (0,0), moves up to y=100, then right to x=200, then up to y=800, and finally right to x=400. The axes range from -600 to 200 on the x-axis and 0 to 800 on the y-axis.
HVL-multi-loops	<p>hvl-multi-loops frames:12193 Qual:98.6 final:0.08 0.28 -1.563</p>  A 2D plot showing a red path on a grayscale map of a complex layout. The path starts at the bottom, moves up, then right, then up, then right, ending at the top right. The axes range from -100 to 75 on the x-axis and 0 to 175 on the y-axis.	 A simplified topometric map of the HVL-multi-loops layout, showing the main corridors and rooms as black lines on a white background.	<p>HVL-multi-loops</p>  A 2D plot showing a blue trajectory on a grid. The trajectory starts at (0,0), moves up to y=1000, then right to x=250, then up to y=1750, and finally right to x=500. The axes range from -1250 to 500 on the x-axis and 0 to 1750 on the y-axis.

KITTI Datasets:

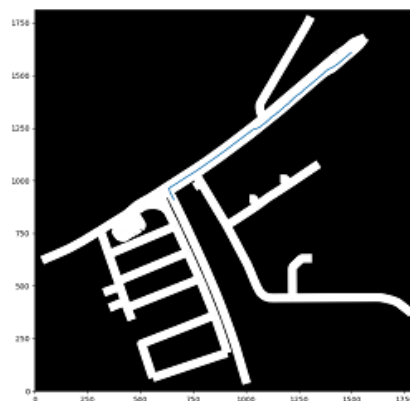
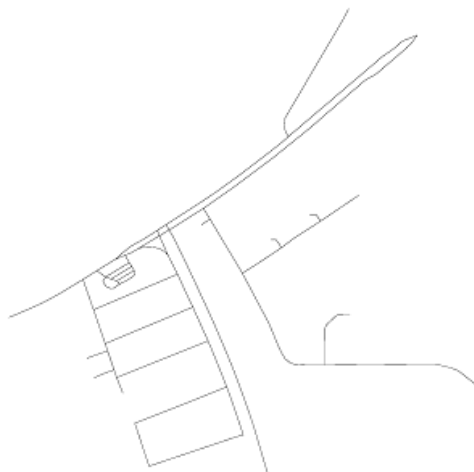
Drive number (2011_09_26_no.)	Topometric map used	Trajectory generated
0001		
0002		
0005		
0009		

Drive number
(2011_09_26_no.)

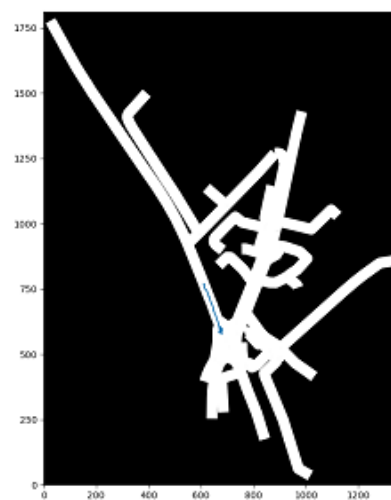
Topometric map used

Trajectory generated

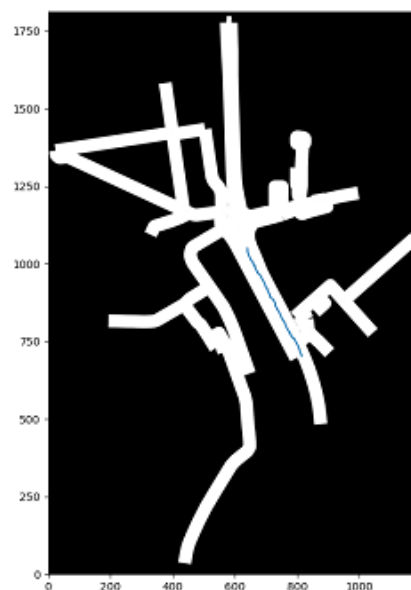
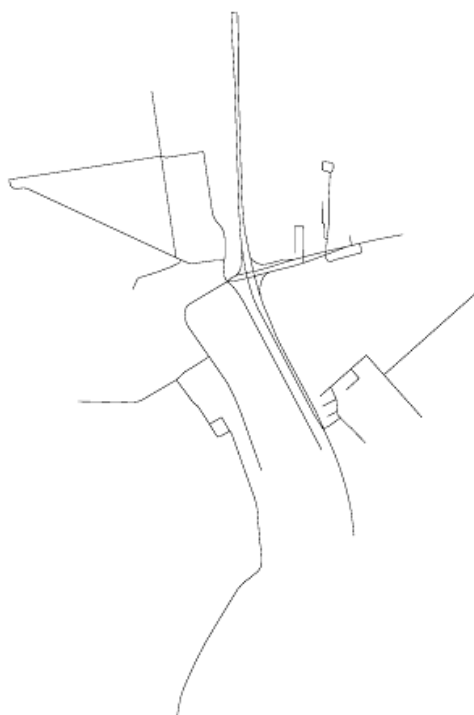
0014



0051



0056

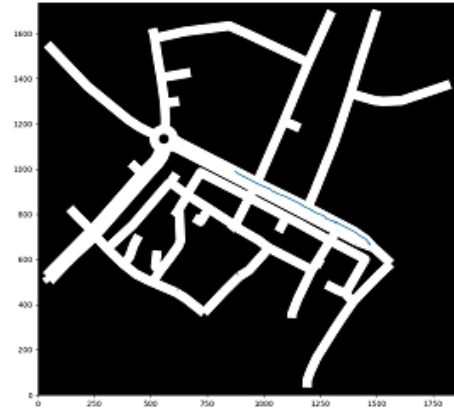


Drive number
(2011_09_26_no.)

Topometric map used

Trajectory generated

0059



- The maplite algorithm was tested on 3 Ati datasets provided by ARTPARK and 8 KITTI datasets available online. Among these, all of the Ati datasets generated trajectory which was 80 – 90% close to the groundtruth. The algorithm was able to generate the trajectory to 7 out of 8 KITTI datasets with about 95% accuracy. The runtime for each iteration is about 90ms.
- The process models (odometry prediction model) used for KITTI and Ati datasets were different, hence there was considerable difference in results. Final velocities from KITTI datasets were used to estimate the pose in odometry prediction step. Whereas, process model mentioned in algorithm was used for Ati dataset. So, the current LiDAR Observation Likelihood can be used with other process model to achieve better results.
- Current model is sensitive to tunable b parameter which acts like weight for odometry prediction pose. The appropriate value for b can be chosen by considering r_{var} (from (2)), $\delta\theta$ (from (3)) and accuracy of the process model.
- The labelling of lidar points as on road or off road is currently done by assuming that only the line of path of vehicle with road width $2r_w$ is the only road. While using this assumption we are unable to classify all the possible road points at an intersection of roads and complete 180 deg turn about a point. This leads to late correction of pose by the model and risk of running into obstacles.
- If the location where the vehicle is driven has good traction and we have considerably accurate odometry prediction model we drop out the sampling of points and only consider sampling of θ . To consider this we should have a concrete model for segmentation of roads, so that error can be minimized. This can reduce the runtime of algorithm to about 50ms.

Future work

- The algorithm can be made robust to b by improving the process model. This can be done by using Odometry-based prediction step to estimate the $[v, \omega]$. In this approach the error generated by noise from IMU can be reduced compared to current model where $[x, y, \theta]$ is being estimated.
- We can use deep learning models similar to rangenet, polarnet, etc for segmentation of road surface to not only navigate vehicle on drivable surface but also classify the LiDAR points as on road or off road. As mentioned in analysis current approach for classification is not concrete and prone to large errors at intersections.
- We can combine this algorithm with trajectory planning and obstacle detection algorithms to autonomously pilot the vehicle at desired destinations. Trajectory planning algorithms like A* can be used with other cost functions to generate trajectory for smooth rides. Some of the cost functions which can be used for trajectory optimization are mentioned below:

Our goal is to calculate optimum spline trajectory which starts at vehicle's position and ends at x_g local goal point of the trajectory. This optimization utilizes a three-part cost function

1. Road distance cost

$$J_d(q) = \frac{1}{k} \sum_{i=1}^k d(q_i)^2 \quad (8)$$

where q is a candidate spline with k waypoints and $d(q_i)$ is the signed distance function to the road points (points on trajectory between vehicle position and local goal point).

2. Relative path length cost

$$J_l(q) = \frac{\sum_{i=2}^k d(q_i, q_{i-1})}{d(x_0, x_g)} \quad (9)$$

where x_0 is current vehicle location.

3. Minimize the maximum curvature of the trajectories

$$J_\kappa(q) = \max \frac{\|\sigma' \times \sigma''\|}{\|\sigma'\|^3} \quad (10)$$

where σ is spline representation of q and, σ', σ'' are first and second derivatives.

So, optimal obtained by solving

$$q_{opt} = \underset{q \in \chi}{\operatorname{argmin}} [J_d(q), J_l(q), J_\kappa(q)] W_q^T \quad (11)$$

where χ represents global trajectory and W_q is a weight vector for tuning the relative importance of the cost function terms.

References

1. Prof. Cyrill Stachniss' lectures on SLAM algorithms. [\[link\]](#)
2. [MapLite : autonomous navigation in rural environments without detailed prior maps](#) by Teddy Ort.