2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Robust LiDAR Feature Localization for Autonomous Vehicles Using Geometric Fingerprinting on Open Datasets

Nicolai Steinke[1], Claas-Norman Ritter[1], Daniel Goehring[1] and Raúl Rojas[1]

***Abstract*— Localization is a key task for autonomous vehicles. It is often solved with GNSS but due to multipath the performance is often not sufficient. Feature localization systems using LiDAR can deliver an accurate localization but the creation of the necessary feature maps is an effortful task. With digitization of urban planning processes a lot of street level data is being generated and increasingly becomes openly available. We propose a novel feature localization system which utilizes geometric fingerprinting to robustly associate features to a feature map generated from this open data from the city of Berlin. With this association, we perform a precise localization of a vehicle in areas spanning over several square kilometers using an optional IMU, the vehicle's CAN-odometry and an initial pose estimate. We evaluated our system with our autonomous car in real world scenarios and achieved a centimeter precision localization accuracy outperforming a high-cost GNSS. The source code will be published at `https://github.com/dcmlr/fingerprint-localization`**

I. INTRODUCTION

A precise self-localization is a necessary preposition for robots in order to operate autonomously. GNS systems alone are not sufficiently accurate especially when it comes to autonomous driving in dense urban traffic conditions. Feature localization algorithms using LiDAR sensors have proven their ability to provide localization estimates with centimeter level precision. Most current algorithms rely on feature maps, which are self-acquired using slam algorithms. Generating precise large scale feature maps with slam algorithms is an effortful task. Following the open data strategy of the European Union (directive 2003/98/EC), a lot of geographic data is openly available and could be used for the localization of autonomous vehicles. Drawbacks of these datasets lie in the facts that they are often several years old and the accuracy is not comparable to self-acquired data. We present a feature matching algorithm using geometric fingerprints which robustly matches the geographic data from the openly available datasets for the city of Berlin [10] to the perception of a LiDAR sensor in real time, and is capable of providing a centimeter level localization using these matchings. The Berlin dataset offers a lot of different feature types to select from. We decided to use pole-like features (trees, traffic lights and traffic signs), walls and corners of buildings because these features are easily detectable with a LiDAR, very common in urban environments, do not change over time, and they are robust in respect to seasons.

This work was supported by the Bundesministerium für Verkehr und digitale Infrastruktur (BMVI) - Automatisiertes und Vernetztes Fahren auf digitalen Testfeldern in Deutschland.

[1]Dahlem Center for Machine Learning and Robotics (DCMLR), Department of Mathematics and Computer Science, Freie Universität Berlin, Germany, {nicolai.steinke | cn.ritter | daniel.goehring | raul.rojas}@fu-berlin.de

II. RELATED WORK

Self-localization approaches have a long history in robotics and a variety of solutions exist depending on the problem domain and the available data. Markov localization, especially Monte-Carlo Localization methods have proven their strengths in many applications and for all kinds of sensors, e.g. in museum environments [14], in office spaces [13], for soccer playing robots [6], in outdoor environments with pole features [7], [9], [12], [16], on highways using traffic signs and lane markings [5], in tunnels of mines [15] and for wheeled robots [3]. LiDAR sensors proved to be capable to achieve highly precise localization results using feature localization algorithms [2], [7], [9], [11], [16]. In recent publications LiDAR SLAM approaches were improved to incorporate global state estimations from GNSS sensors to counter drift issues [11] and it was shown how current Monte-Carlo Localization methods can be improved using machine learning [2]. Algorithms that use semantic features in outdoor environments often rely on pole features, which have proven to be robust and easily detectable as shown by Schaefer [7], who proposed a pole localization system and evaluated its long term stability over 15 months. Research by Sefati [9] and Weng [16] showed that pole features are robust and stable features in challenging urban scenarios with many dynamic objects and occlusions. Pole feature localization algorithms usually rely on feature maps and utilize particle filters in order to match the observed features to the map [7], [9], [12], [16]. A different approach is the extraction and matching of geometric patterns of feature locations to a feature map. Brenner [1] showed that the local patterns of pole features in urban road scenes are sufficiently unique to be used as fingerprints for a matching algorithm. This was demonstrated in a subsequent article where Schlichting [8] reduced the rate of erroneous matchings of pole features from 83% to 11% compared to a nearest neighbor baseline. The experiments were performed using a low-cost LiDAR attached to a vehicle. The feature maps are mostly created by applying the same pole detector that is used for the localization task to registered 3d point clouds [7], [9], [16]. This limits the prospects of replicating the performance with another detector or sensor on the same map and it exposes economic limitations on the localization systems since the point clouds have to be acquired with a LiDAR equipped vehicle and processed for every area beforehand.

III. FEATURE DETECTION AND TRACKING

A. General feature detection considerations

We selected the following feature types for our implementation: pole-like structures, building and fence corners, walls and fences, because they are common and easily detectable with a 64-beam Velodyne HDL-64E LiDAR. Pole-like structures consist mostly of trees, traffic signs, traffic lights and street lights. Our tests showed that there are places where there are not enough pole features to perform an accurate localization. In these cases the addition of building corners and walls improved the localization results. In order to maintain a high execution performance we tried to keep the feature detection algorithms as fast and simple as possible. The main goal of the detector is to detect as many features as possible even if this leads to a higher false positive rate. The feature tracker and the fingerprint recognition should be able to filter the false positives. The LiDAR produces point cloud data which is divided in 64 rings, one for each laser beam. Due to the mounting of the lasers, the rings stack on top of each other. The detection algorithms are scan line algorithms, which use 4-8 of these rings as scan lines per point cloud. The point clouds are merged 360° point clouds, which are generated at a 10 Hz frequency.

B. Detection of pole features

The pole feature detection scan line algorithm works as follows: Assuming the car sits on a perfectly even and empty plane with no objects around it, every laser of the sensor hits the ground somewhere around the car because all lasers have a downward angle. The resulting point trace over time for each laser is a circle on the plane. If there is any pole-like feature some points are missing from the circle because they are on the pole. Considering only one laser, the pole is visible in the resulting point cloud as a small number of points which are significantly closer to the car than the points around it. The principle is visualized in Fig. 1, where the three points in the center are closer to the laser origin than the ground points on the sides. This is the working model of our scan line pole feature detector: find some points spanning a horizontal distance less than one meter which are at least one meter closer to the car than the points around it. If we see the pole feature in the majority of the rings processed by the detector (we used eight rings), the detector will return it as a detected pole feature. For the center point estimation, we assume that the poles are circular or rectangular in shape, which is true for the majority of poles seen on the streets of Berlin. We take the two most outer points of the pole on each side and combine them to a line. Then we take the center point on the surface and the closest point to it from the line formed by the selected outer points to get the direction vector. The center point is then placed at $\frac{3}{2}$ times the pole width along the direction vector (we usually measure about $\frac{2}{3}$ of the pole width). Note: If the pole is convex the direction vector will point away from the car, if it is concave, it will point towards the car. The center point estimate is done for every ring and the mean of the estimates is used as final pole feature position. This detector is much simpler as the ones proposed in other works [7], [9], [16] and produces many false positives, but it has a very good execution performance (on average 9.1 ms per point cloud on a 2013 Intel Xeon E3-1221) and the fingerprint recognition step filters the false positives.

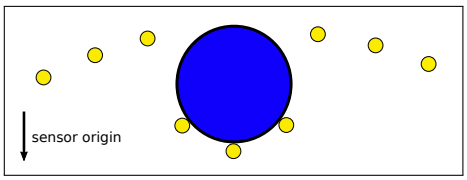


Fig. 1. Schematic representation of a pole feature (blue) with laser intersection points (yellow).

C. Detection of wall and corner features

We also use a laser scan ring based scan line algorithm for the detection of wall features. The algorithm adds points consecutively and tests if they form a line by selecting two random points and checking for every new point if the distance to the line and the last added point falls below a threshold (we used 0.25 m for the maximum line distance and 0.5 m distance from the last point). If a new point lies outside the inlier distances, a new line segment starts. The collected points of the last line segment are checked for their total length and if it is longer than a threshold of five meters it is considered to be a wall hypotheses and a RANSAC line fit is performed. With the corner feature detection we mainly aim for the detection of building corners. Thus we define a corner feature as two walls meeting in an angle between $\pm 80°$ to $\pm 100°$.

D. Feature tracking

We use a simple, nearest-neighbor based tracking approach for pole and corner features, which are treated as point features in the feature tracker. The tracking algorithm is executed for every point cloud measurement from the LiDAR which operates with a frequency of about 10 Hz. The tracking on wall features is performed in similar fashion as the tracking of pole and corner features with the difference that not only the distance but also the angle is considered.

IV. FEATURE FINGERPRINTING

In order to localize the vehicle, the detected features must be associated with stored features in the feature map. Since the data we are using is already several years old (2015) and not error-free, the algorithm is required to be robust against missing and changing features in the map. We developed a novel fingerprint matching algorithm, which is accurate, efficient and robust against map inaccuracies. The algorithm identifies a feature by its type and its characteristic geometric fingerprints (Fig. 2). A feature's fingerprint vector consists of all pairwise distances and angles to all other features in a predefined radius (we used 50 m). The angle is measured clockwise in respect to the grid north of the chosen Cartesian coordinate system. This means that a localization is only possible if the north direction is known with at least some

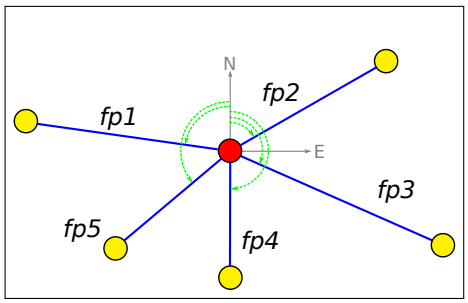


Fig. 2. Fingerprints (angles w.r.t the north vector are green dashed arrows, distances are blue lines) of the centered point feature (red, others are yellow).

degree of accuracy (less than $\pm 10°$) for the initial pose estimate. This might seem like a major drawback, but in our experience even low cost GNSS-receivers can determine the north vector with sufficient accuracy in a moving vehicle. This limitation could be overcome by matching only the distance field of the fingerprints until a stable matching is established with the cost of longer initialization time since more fingerprints would need to be collected until a valid matching can be found.

A. Geometric fingerprints of point features

The fingerprint vector of feature *a* consists of tuples of the Euclidean distance and the angle of the feature *a* to a feature *b* and all other features in the fingerprinting range (1).

$$\vec{f_a} = \{(d(a,b), \alpha(a,b)), ...\} \tag{1}$$

The calculation of the angle between two point features p_1 and p_2 is shown in (2). The resulting angle $\alpha(p_1, p_2)$ is in the range $[-\pi, \pi]$ where the UTM north vector is defined as zero.

$$\alpha(p_1, p_2) = \cos^{-1}\left(\frac{p_{1_y} - p_{2_y}}{d(p_1, p_2)}\right) \text{sgn}(p_{1_x} - p_{2_x}) \tag{2}$$

B. Geometric fingerprints of edge features

We found that the algorithm would not perform well in very narrow roads where pole features are sparse and often too close to the walls of the buildings to be consistently detected. Therefore we added the possibility to use corners and walls of buildings to improve the feature density in very narrow streets. We decided to use these features because they are frequently present in dense urban environments, easily detectable and rarely occluded for a LiDAR mounted on a car's roof. However, walls can only be used to improve the localization laterally since they are usually only partially observed. The fingerprints must allow to identify a wall even if only a fraction of it has been detected. Hence we treat an edge feature for fingerprinting as an infinite line and define the distance between an edge and a point feature as the smallest Euclidean distance between the two (Fig. 3).

$$\vec{l_1} = \frac{\begin{pmatrix} e_{1_{x2}} - e_{1_{x1}} \\ e_{1_{y2}} - e_{1_{y1}} \end{pmatrix}}{|e_1|} \tag{3}$$

$$proj = \langle \vec{l_1}, \vec{p_1} \rangle \tag{4}$$

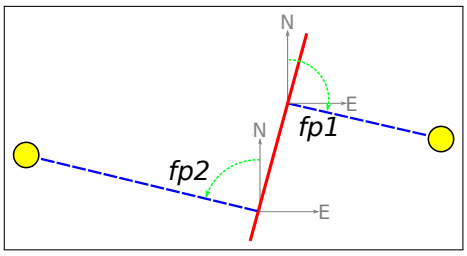


Fig. 3. Two fingerprints (blue dashed lines and green arrows) of the red edge feature (other features are yellow).

$$c(p_1, e_1) = proj \cdot \vec{l_1} + \begin{pmatrix} e_{1_{x1}} \\ e_{1_{y1}} \end{pmatrix} \tag{5}$$

$$d(p_1, e_1) = \sqrt{(p_{1_x} - c(p_1, e_1)_x)^2 + (p_{1_y} - c(p_1, e_1)_y)^2} \tag{6}$$

Equation 6 defines the distance function *d* for an edge defined by two points $e_1 = ((e_{1_{x1}}, e_{1_{y1}}), (e_{1_{x2}}, e_{1_{y2}}))$ and a point feature $p_1 = (p_{1_x}, p_{1_y})$. The closest point for p_1 on the edge e_1 is defined by the function $c(p_1, e_1)$, which is shown in (5). Note that the multiplication $proj \cdot \vec{l_1}$ in (5) is a scalar multiplication of the direction vector l_1 since $proj$ is the scalar result of the inner product in (4).

$$\vec{r} = (c(p_1, e_1)_x - p_{1_x}, c(p_1, e_1)_y - p_{1_y}) \tag{7}$$

Equation 7 shows the calculation of the rejection vector $\vec{r}$ from the point feature $p_1 = (p_{1_x}, p_{1_y})$ and the edge feature $e_1 = ((e_{x_1}, e_{y_1}), (e_{x_2}, e_{y_2}))$. The rejection vector points from the closest point on the edge e_1 to the point p_1.

$$\alpha(p_1, e_1) = \cos^{-1}\left(\frac{r_y}{|\vec{r}|}\right) \text{sgn}(r_x) \tag{8}$$

Equation 8 defines the calculation of the angle from a point feature p_1 to an edge feature e_1. The angle is measured between the vector from the closest point on the edge e_1 to p_1 with respect to the y-direction (north) vector.

$$\alpha(e_1, e_2) = \cos^{-1}\left(\frac{\vec{l_1} \cdot \vec{l_2}}{|\vec{l_1}||\vec{l_2}|}\right) \text{sgn}(l_{1_x}) \text{sgn}(l_{2_x}) \tag{9}$$

The angle between the edges e_1 and e_2 is defined in (9). The direction vector $\vec{l_2}$ of the edge e_2 is defined analogous to $\vec{l_1}$ (3). As in (2) the sign of the x-coordinates of the directional vectors determine the sign of the resulting angle.

$$\alpha(e_1) = \cos^{-1}\left(\frac{l_{1_y}}{|\vec{l_1}|}\right) \text{sgn}(l_{1_x}) \tag{10}$$

The angle of an edge feature is calculated by the inverse cosine of the y-coordinate of the edge vector in relation to the length of that vector (10). This is basically the same calculation as performed for the rejection vector for edge to point angles (8). We define the angle of an edge feature against the map north vector as the angle value of a fingerprint against any other edge feature (9).

$$d(e_1, e_2) = \alpha(e_1, e_2) \tag{11}$$

The distance between the two edge features e_1 and e_2 is calculated as angle between them (11). Since edge features

are considered with infinite length, the Euclidean distance between two edges is zero at one point as long as they are not perfectly parallel. Hence we decided to use the angle difference in this case, which causes the distance and angle fingerprint fields to be dependent only on the edge angle. This weakens the uniqueness of a fingerprint. However, we found that only the angle of our detected walls was robustly detectable under almost all circumstances.

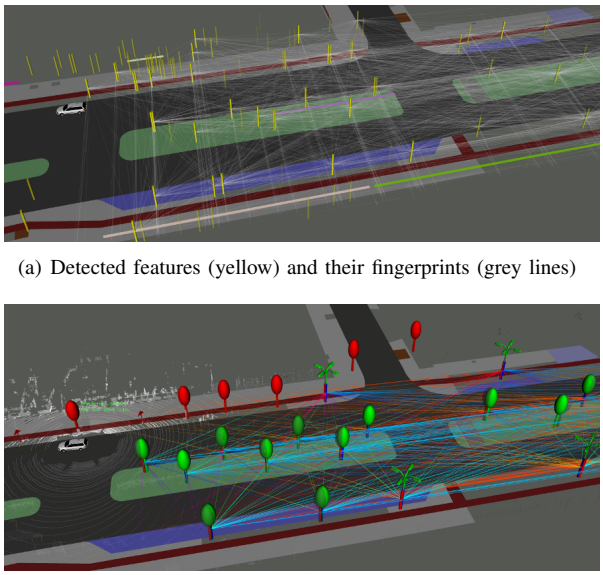
C. Apriori map creation and storage

We retrieved the feature positions with a geographic information system (GIS) application which is capable of accessing the web feature services (WFS) operated by the city of Berlin (available at [10]). We collected the information about street trees, traffic lights, traffic signs, street lamps and building models. The building models contain the buildings outlines as polygons which can be used to compute the wall features. We defined an angle threshold value of $\epsilon = \frac{\pi}{10}$ for two polygon edges to be considered to form the same wall segment and a minimal wall length of 5 m. If two building outline polygon edges meet at an absolute angle in $[\frac{\pi}{2} - \epsilon, \frac{\pi}{2} + \epsilon]$, it is considered to be a building edge. With the resulting feature set M we compute the angle and distance fingerprints values for all feature pairs $(a, b) \in M \times M$, where the distance $d(a, b) < 50\ m$. For storage purposes we use an SQLite database with the Spatialite extension for spatial data and indexing. The fingerprints are stored in a table along with the position, an identifier and the type (pole, corner, wall) of the feature they are belonging to. A spatial index (implemented as R*-tree in Spatialite) on the feature position provides fast searching and filtering capabilities. The maps we generated used 33.75 MiB of storage space per km^2 on average.

V. FEATURE RECOGNITION AND LOCALIZATION

The recognition of the fingerprinted features works as follows: The fingerprints of all detected features are calculated locally on the fly and then compared with the ones stored in the feature map database. An uncertainty threshold value is used to counter measurement inaccuracies. From this comparison a list of features, sorted by the count of matching fingerprints, is received. This principle is visualized in Fig. 4, which shows the pre- and post-matching state: The locally calculated fingerprints (grey lines) of the detected features (yellow cylinders and colored line segments) are compared against the fingerprints in the database and in case of matching fingerprints (colored lines) a feature association is established (green). Matchings with an additional red and blue cylinder were used for the localization. Some features have a grey outline next to them, which indicates a positional offset between the detected feature and the one stored in the map.

Equation 13 represents the point feature recognition query using relational algebra against the fingerprint table proposed in sec. IV-C. It is calculated for every feature and filters all stored fingerprints by the feature position ($d(f_x, f_y, q_x, q_y < r)$), where the distance d of the database feature with the coordinates f_x, f_y to the coordinates of the query feature



(a) Detected features (yellow) and their fingerprints (grey lines)

(b) Matched and not matched features (green and red) of the feature map and matched fingerprints (colored lines)

Fig. 4. Visualization of the feature matching process

q_x, q_y must be smaller than the search radius r. The second filter matches the feature type (point or edge). The fingerprint matching is represented in the term p (12): For every tuple consisting of a distance value q_d and an angle value q_α of the query feature q a filter condition is generated, which holds for tuples having a distance and angle difference lower than the corresponding tolerance values ϵ_d and ϵ_α. These values can be adjusted at runtime in order to compensate for changed measurement conditions. The selection operator σ (13) returns the count of the matching fingerprints, the corresponding identifier and the position for every map feature in the search radius r.

$$p = \bigvee_{i=0}^{n} (|q_{di} - f_d| < \epsilon_d \wedge |q_{\alpha i} - f_\alpha| < \epsilon_\alpha) \tag{12}$$

$$\sigma_{d(f,q)<r \wedge f_t = t \wedge p}(\Sigma(id), id, x, y) \tag{13}$$

The relational algebra representation of the recognition of the edge features is shown in (14). The query is mostly identical to (13) with the exception of an additional term $d(f_p, q_p) < \epsilon_{oR}$, which enforces the position of the other features of the fingerprints to be closer together than ϵ_{oR} and the selection of the start and end point of the edge. This term was introduced to avoid multiple matching of edge to point fingerprints. In the case of edge to edge matching it is omitted. Generally the feature with the most matching fingerprints is correctly associated with the corresponding detected feature, but in order to improve the localization result we use a median filter to reduce the variance of the matched feature's positional offsets.

$$\sigma_{d(f,q)<r \wedge f_t = t \wedge p \wedge (d(f_p,q_p)<\epsilon_{oR})}(\Sigma(id), id, x_1, y_1, x_2, y_2)) \tag{14}$$

In order to maintain a low computation time, a reduction in the search radius r for the features in the map is nec-

essary. The search radius should not be larger than 2 to 5 m depending on the localization uncertainty and mapping accuracy. Nonetheless in case of a very high positional uncertainty, this threshold can be arbitrarily large with higher computational load. In order to find the transformation between matched features, we used the PCL singular value decomposition implementation of the standard point-to-point error metric, which is also used by the ICP algorithm (TransformationEstimationSVD) [4]. We use the transformation estimate to improve our existing localization, which is calculated from an initial position, the wheel odometry and an IMU utilizing an extended Kalman filter. We used the open source implementation of the robot localization node module (`http://wiki.ros.org/robot_localization`) of the Robot Operating System (ROS) and fused the data with the wheel odometry and IMU data of the car.

VI. EXPERIMENTS AND EVALUATION

The evaluation focuses on the robustness of the feature matching algorithm and compares the global localization accuracy with a GNSS in different scenarios. We performed one experiment to evaluate the robustness and another two in order to evaluate the accuracy based on a repeatability measure. The first experiment was performed in a challenging urban scenario (Fig. 5) with bigger and smaller residential roads with a length of about 16 km, which were driven according to the speed limits (between 30 km/h and 50 km/h). The feature map was spanning over an area of about 35 km^2, contained 33 665 features and was created from unmodified openly available data. The feature matching needed an average of 35.6 ms on a 2013 Intel Xeon E3-1221 CPU (4 cores) and was executed with 10 Hz frequency. The implementation uses four threads for the database matching and caches matchings for up to one second depending on the count of matching fingerprints. It is purely CPU based and no GPU is used. The database file is stored on a consumer grade SSD-drive. The coloring of the trajectory (Fig. 5) represents the count of matched features in that position. While it mostly stays in a sufficient range for a stable and accurate localization, it drops to lower levels in some very narrow residential streets. This can be explained by the lower overall feature count due to the narrowness of the street and the lack of fences in the used data set for the feature map. This leads to a much lower observable feature count in some areas. Despite these problems, the visible feature density is almost anywhere high enough in order to maintain a consistent matching. Only one full loss of matchings occurred (red part of the trajectory of Fig. 5) which was caused by a construction site and road closure, which occluded the LiDAR and forced us to do a u-turn. However a stable matching was regained very quickly and without any intervention. Excluding the initial pose estimate, there was no fusion of GNSS data and the localization was based only on the LiDAR feature localization, IMU and wheel odometry. The system's robustness is underlined by the fact that does not get "lost" even after driving through several underpasses (Fig. 5 center), which obstruct

Fig. 5. Matching robustness evaluation in a challenging urban area. Satellite imagery: [10]

the LiDAR and limit the feature visibility. The evaluation of the localization accuracy is challenging because of the lack of a precise ground truth. Even our high cost Applanix GNSS is not able to provide a localization accuracy that is precise enough in order to use it as ground truth. The system is theoretically capable of delivering a position estimate within an accuracy of centimeters. However, in practise the position estimation error can reach up to a meter in a moving vehicle due to multipath reflections, limited GPS visibility and low quality of the RTK-Signal. Therefore we chose a repeatability measure for the evaluation of the localization accuracy and perform a comparison between the high cost GNSS fused with the IMU and wheel odometry and the feature localization using different sensor setups. For the repeatability measure the car is driven in a closed-loop track for several laps. The lateral localization error is calculated as distance from the average trajectory of all laps for every trajectory point. The disadvantage of this method is that the driver of the test vehicle cannot drive perfectly on the lane center, which introduces an additional error in the localization accuracy. This error is more pronounced at the u-turn maneuvers which are excluded from the evaluation for this reason. Additionally we perform a spatial filtering with a distance of 16 cm between two consecutive points to avoid a bias towards slowly driven and stopping parts of the trajectory. Two repeatability experiments were executed and evaluated for accuracy. The first took place in a parking lot, which exhibits near perfect conditions for both localization methods. It is a big open space, which minimizes multipath reflections for the GNSS and there are a lot well visible poles which benefit the feature localization. A line pattern on the street helps the driver to repeat the same trajectory with high precision. One lap is about 450 m long and was driven with an average speed of about 25 km/h with a maximum of 50 km/h. We drove 13 laps with a total distance of 5.85 km.

TABLE I

LOCALIZATION ACCURACY EVALUATION

	parking lot						urban street					
	IMU				w/o IMU		IMU				w/o IMU	
lat. error (m)	GNSS	HDL64	Lux	LIO-SAM	HDL64	Lux	GNSS	HDL64	Lux	LIO-SAM	HDL64	Lux
max	0.391	0.193	0.268	1.794	0.417	0.307	1.516	0.297	0.426	2.056	0.710	0.714
avg	0.051	0.028	0.029	0.059	0.031	0.033	0.460	0.073	0.077	0.190	0.091	0.088
σ	0.053	0.026	0.030	0.087	0.039	0.038	0.309	0.054	0.059	0.202	0.074	0.077
rmse	0.074	0.039	0.041	0.105	0.050	0.051	0.554	0.091	0.097	0.277	0.117	0.118
feat. reject. rate	-	0.736	0.890	-	0.845	0.840	-	0.858	0.671	-	0.836	0.633

The second experiment replicates a more realistic traffic scenario where the car was driven repeatedly in normal traffic conditions on a public road. Fig. 6 shows the trajectories of both localization methods and the content of the feature map with underlying aerial imagery. We tested the feature localization with two different LiDAR sensor setups with and without fusing the IMU data, the wheel odometry was always used. The first LiDAR setup was using Velodyne HDL-64E, which has 64 beams with a vertical resolution of about $0.4°$ and an horizontal resolution of $0.08°$. The feature detection and tracking was performed as stated in sec. III. The second setup was an array of six Ibeo Lux 4L LiDARs, which are mounted in the bumpers of the car. These LiDARs have 4 beams and resolutions of $0.8°$ vertical and $0.25°$ horizontal with an FoV of $110° \times 3.2°$. The laser firing direction is almost parallel to the ground, which results in a higher effective detection range compared to the HDL-64E, but the low mounting height leads to a higher chance of occlusion. The Ibeo Lux sensors are combined with a proprietary Ibeo fusion and tracking module that has access to the vehicle's CAN odometry data and fuses the point clouds of the sensors and offers an obstacle detection, tracking and classification. We replaced our feature detection and tracking algorithm with the proprietary Ibeo module for the experiments with the Ibeo sensors. Since the module is not capable of classifying poles, walls or corners, we resorted to feed all detected static objects as pole features to the fingerprint matching module. Additionally to the lateral localization error, the total count of tracked features along with the count of successfully matched features and the rejection rate is given in table I. For reference we provide the results of the LIO-SAM [11] method performed on the same datasets along with our results. LIO-SAM used the Velodyne HDL-64E, GNSS and IMU data but not CAN odometry data for the localization. We also set the downsample rate parameter to 2 as suggested by the authors for the Kitty dataset which uses the same LiDAR model. Unfortunately, the LiDAR and IMU sensor clocks are not perfectly synchronized in our dataset, which might lead to a slightly degraded performance. Nonetheless the results are comparable since all methods operated under the same conditions. The results show that all methods are capable of localizing the vehicle with a high degree of accuracy in the parking lot scenario. Compared to the parking lot the localization methods show a worse performance in the more challenging urban scenario, especially the GNSS, which suffers from multipath reflections. Regarding the sensor setups, the Velodyne HDL-64E paired with the IMU delivers the best results but nonetheless the results of the Ibeo Lux sensors are impressive considering the lower resolution. For a further evaluation of the impact of the resolution on the accuracy an ablation study was performed (table II), where only every second or fourth beam of the Velodyne HDL-64E was used, which halves or quarters the vertical resolution. In order to maximize the impact of the reduced resolution, the IMU was not fused for the ablation study. The ablation study shows that a lower vertical resolution leads to a worse localization performance. With every second beam used, the accuracy is still high in both scenarios but if only every fourth beam is used (16 beams total) the detection range suffers, which leads to less detected features and a significantly reduced localization accuracy. Due to the lower feature count, the matching runtime improved to 33.8 ms for 32 beams and 21.8 ms for 16 beams (measured on an Intel Xeon E3-1221). Depending on the scenario and sensor, the rejection rate can increase or decrease but we could not determine a simple causal relationship between sensor resolution and the rejection rate. Further investigation is needed to find the factors which fully explain the rejection rate's behavior.

VII. CONCLUSIONS

We presented a novel feature matching and localization approach using geometric fingerprinting. The results of the experiments provide proof of the good performance in different scenarios. The fingerprinting algorithm solves the association problem robustly and with high accuracy (avg. 7.3 cm lateral error) in real world scenarios. The initial pose estimate does not need to be highly accurate, which makes this localization technique independent of expensive GNSS receivers. The feature maps can be generated from freely

TABLE II

ABLATION STUDY ON THE VERTICAL LIDAR RESOLUTION WITHOUT IMU

	parking lot		urban street	
lat. error (m)	32 beams	16 beams	32 beams	16 beams
max	0.749	0.638	0.533	1.055
avg	0.039	0.044	0.095	0.127
σ	0.060	0.066	0.082	0.125
rmse	0.071	0.079	0.126	0.178
feat. reject. rate	0.870	0.646	0.876	0.794

available data and do not need to be perfectly accurate or up to date. Missing or additional features have no significant impact on the matching performance. The execution time amounts to 44.7 ms per 360° point cloud on an Intel Xeon E3-1221 (9.1 ms feature detection, 35.6 ms matching). The feature maps can be large scale, spanning several square kilometers and containing tens of thousands of features. In this article we exclusively used LiDAR sensors but the algorithm itself is not bound to any type of sensor. Hence it is possible to use other sensor types, like stereo-cameras or mono-cameras combined with a depth estimation. Since the features are represented as edges and vertices, lane lines and curbs can easily be added for additional accuracy benefits.

REFERENCES

[1] C. Brenner, "Global localization of vehicles using local pole patterns". In Proc. of Joint Pattern Recognit. Symp.", 5748, pp. 61—70, Sep. 2009.

[2] X. Chen, T. Läbe, L. Nardi, J. Behley and C. Stachniss, "Learning an Overlap-based Observation Model for 3D LiDAR Localization", In Proc. of IEEE/RSJ Int. Conf. on Intell. Robots and Syst., pp. 4602-4608, Oct. 2020.

[3] P. Elinas, and J. Little. "σMCL: Monte-Carlo Localization for Mobile Robots with Stereo Vision". In Proc. Robot.: Sci. Syst., pp. 373-380, June 2005.

[4] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu and S. Behnke, "Registration with the point cloud library: A modular framework for aligning in 3-D". IEEE Robot. & Automat. Mag., vol. 22, no. 4, pp. 110-124, Dec. 2015.

[5] W. C. Ma et al., "Exploiting sparse semantic HD maps for self-driving vehicle localization". In Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., pp. 5304-5311, Aug. 2019.

[6] T. Röfer and M. Jüngel, "Vision-Based Fast and Reactive Monte-Carlo Localization". In Proc. IEEE Int. Conf. Robot. and Automat., pp. 856-861, Sept. 2003.

[7] A. Schäfer, D. Büscher, J. Vertens, L. Luft and W. Burgard, "Long-Term Urban Vehicle Localization Using Pole Landmarks Extracted from 3-D Lidar Scans". In Proc. Eur. Conf. Mobile Robots, pp. 1-7, Oct. 2019.

[8] A. Schlichting and C. Brenner, "Localization using automotive laser scanners and local pattern matching". In Proc. IEEE Intell. Veh. Symp., pp. 414–419, June 2014.

[9] M. Sefati, M. Daum, B. Sondermann, K. D. Kreiskother and A. Kampker, "Improving vehicle localization using semantic and pole-like landmarks". In Proc. IEEE Intell. Veh. Symp., pp. 13–19, June 2017.

[10] Geoportal Berlin / "Digitale farbige Orthophotos 2019 (DOP20RGB)", "Straßenbefahrung 2014". dl-de/by-2-0, online, https://www.stadtentwicklung.berlin.de/geoinformation/

[11] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping". In Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., pp. 5135-5142, Oct. 2020.

[12] R. Spangenberg, D. Göhring and R. Rojas, "Pole-Based Localization for Autonomous Vehicles in Urban Scenarios". In Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., pp. 2161-2166, Oct. 2016.

[13] C. Stachniss and W. Burgard, "Mobile Robot Mapping and Localization in Non-Static Environments". In Proc. 20th Nat. Conf. on Artif. Intell., pp. 1324-1329, July 2005.

[14] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte and D. Schulte, "MINERVA: A Second-Generation Museum Tour-Guide Robot". In Proc. IEEE Int. Conf. Robot. Automat., pp. 1999-2005, May 1999.

[15] Y. Wang, L. Huang and W. Yang, "A Novel Real-Time Coal Miner Localization and Tracking System Based on Self-Organized Sensor Networks". EURASIP J. on Wireless Commun. Netw., 2010, pp. 1-14, July 2010.

[16] L. Weng, M. Yang, L. Guo, B. Wang and C. Wang, "Pole-based realtime localization for autonomous driving in congested urban scenarios". In Proc. IEEE Int. Conf. Real-time Comput. Robot., pp. 96-101, Aug. 2018.

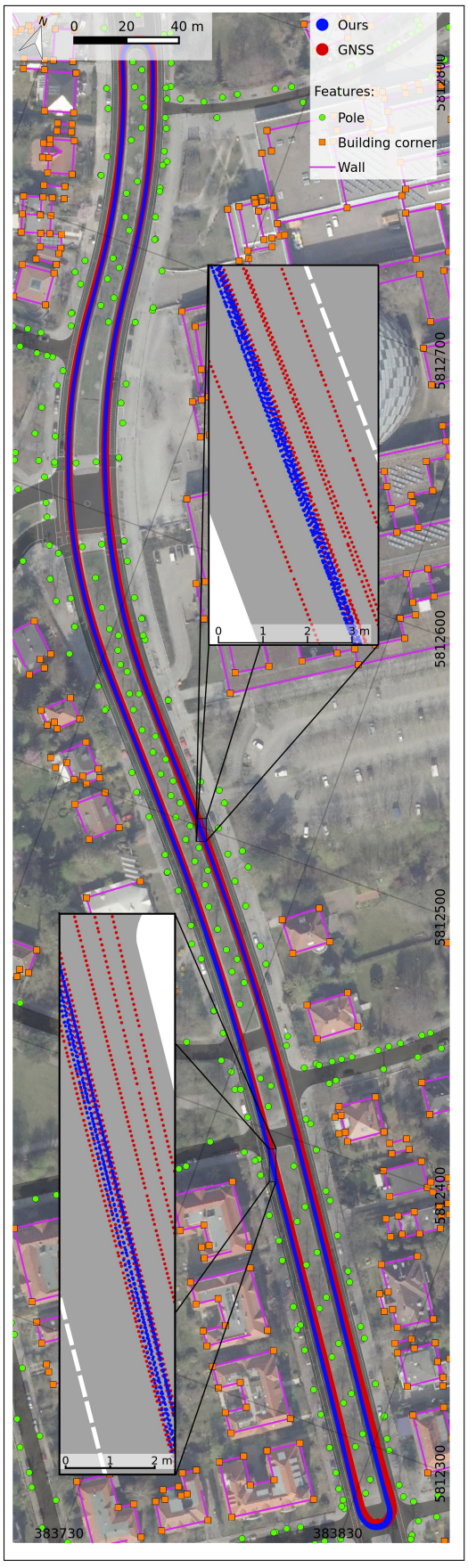


Fig. 6. Comparison of trajectory with feature localization and GNSS localization. (Coordinates are in UTM zone 33N, data and satellite imagery: [10])