TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Radar SLAM for Autonomous Driving

Authors	Felix Bergmann, Frithjof Winkelmann, Hans Schmiedel, Michael Seegerer
Mentor(s)	Dr. Georg Kuschk Astyx GmbH / Cruise
Co-Mentor	M.Sc. Fabian Wagner
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2020

Abstract

Simultaneous Localization and Mapping (SLAM) is a problem from robotics concerned with mapping the environment while simultaneously localizing the robot in the environment. It constructs a map of the environment that can be used for further action planning. It has applications in many robotics problems, for example autonomous driving. Many types of sensors can be used, for example cameras, lidar or radar sensors. While cameras and lidar often perform poorly in bad weather environments, radar sensors produces good measurements even in extreme weather conditions. This makes them a good alternative to cameras and lidar.
Using radar sensors poses some challenges, since their output is usually more noisy and contains fewer data points than lidar scanners.
In this project we explore different solutions to SLAM for radar data in the context of autonomous driving.

Contents

1 Introduction

1.1 Motivation for the Project

Deploying autonomous drivings requires a proper knowledge of the environment and accurate localization within this environment. The perception of the environment is normally provided by highly accurate lidar and camera sensor systems. These systems can suffer greatly from visibility impairments usually caused by bad weather conditions. Radar sensors don't suffer as much from these problems but generally have a lower resolution, are more noisy, and usually feature only 2 geometrical dimensions.

Astyx GmbH/Cruise is working to solve this problem by building high resolution 3D geometrical radar sensors which can capture many more data points per scan. For reference, the only other public automotive radar dataset has around 100 points per scan [1], while the Astyx GmbH/Cruise sensor delivers approximately 1000 points [2] and a typical lidar sensor has about ≈ 100000.

This project investigates wether the radar sensors by Astyx GmbH/Cruise can be used to improve SLAM systems.

1.2 Goal of the project

The goal of our project is to evaluate classical and deep learning based approaches for co-registering multiple $3D$ radar point clouds and computing $6DoF$ relative poses between them in the context of simultaneous localization and mapping (SLAM).
The main tasks carried out are the following:

- Literature research (sparse point cloud approaches vs camera-SLAM & Lidar-SLAM)
- Evaluate point cloud based classical algorithms on radar point clouds
- Evaluate state-of-the-art neural network approaches on radar point clouds
- Comparison against public camera/Lidar approaches using the existing multi-sensor setup
- Improvements via segmentation of static/dynamic objects

The project is carried out using a radar sensors developed by Astyx GmbH / Cruise.

1.3 The SLAM Problem

Simultaneous Localization and Mapping (SLAM) is a problem from robotics, which is concerned with mapping the environment of a robot while simultaneously estimating its state in the environment. Commonly the map is represented by the location of interesting keypoints in the environment. The state, or pose, is usually described by the trajectory of the robot, which is a sequence of locations, but can also include other things like for example the speed.

In a normal SLAM application, the sensor measurements come in as the robot operates. Since the map and state of the robot are needed for further action planning, this means that SLAM algorithms typically have to run in real time. Also note that the estimation of states at an earlier time can and should be updated as new measurements come in. This aspect is typically referred to as **loop closing**. It is typically necessary because the robot accumulates error in its state estimates stemming from the measurement noise.
The tasks the system therefore needs to solve are:

- Estimation of the current state of the robot: Given the state of the robot at the previous time step x_{k-1} and the measurement from the current time step z_k, estimate its new state x_k
- Loop closing: Given all measurements and state estimates up to the current time step, find loops and use those to minimize the accumulated error of all state estimates.

These two steps are typically called the front end and back end of SLAM [3]:

- The front end takes the sensor data as input and is responsible for detecting and extracting relevant features within this data. It also associates matching features in consecutive scans. The front end typically outputs the new state x_k. When the state consists of the position the front end simply extracts the relative pose between consecutive frames.
- The back end then takes all the information of the front end to deduce the robots position and to update its interior map representation. It associates data between the current measurement and much earlier scans.

This separation of SLAM systems is depicted in Figure 1.

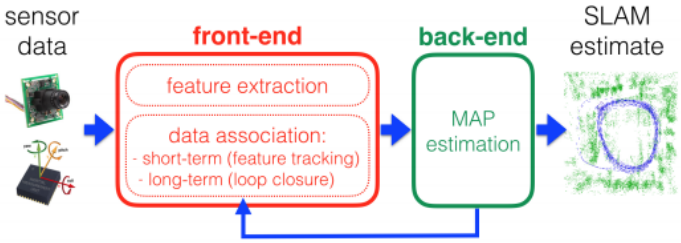


Figure 1: Separation of front end and back end by [3].

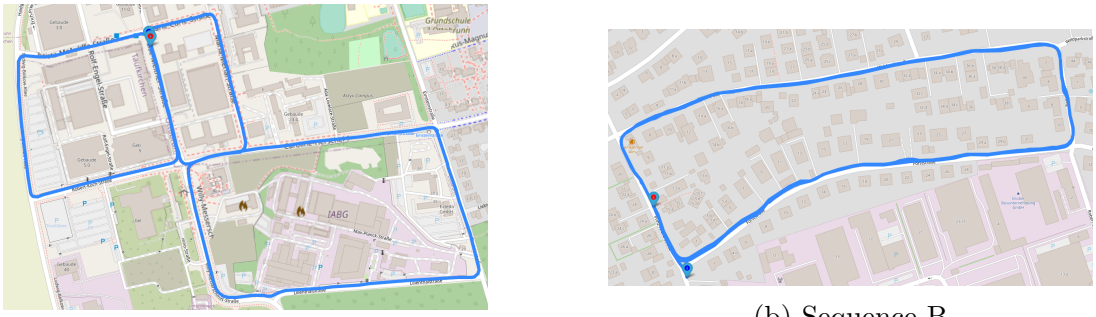
2 Datasets

2.1 Astyx/Cruise Dataset

We use a dataset from Astyx/Cruise [2]. This data includes

- Point clouds from a radar sensor, where each point cloud contains about 1000 points
- Point clouds from a lidar sensor (Velodyne VLP-16)
- GPS data with an accuracy of $0.5m$ in position and $3.14°$ in orientation
- CAN bus data of the ego vehicle itself like the current yaw rate and the velocity.

This data was captured at a rate of 10Hz for two sequences of consecutive driving spanning about 2000 and 4000 frames respectively.
The GPS trajectory of the sequences can be seen in Figure 2.



(a) Sequence A

(b) Sequence B

Figure 2: GPS Tracks from the two sequences in the Astyx dataset

Astyx GmbH/Cruise also has a public dataset called Astyx HiRes2019 Dataset. It only contains 546 frames with the same frequency of 10 Hz but includes camera data for each frame at 2048 × 618 resolution. The other sensors specifications are the same.

The radar sensors produce a cloud of $5D$ points consisting of $[x, y, z, V_r, mag]$ at each frame. Additionally to the $3D$ geometric position $[x, y, z]$ of each point they yield a relative radial velocity $V_r \in [-5.2m/s, 5.2m/s]$ and a feedback magnitude $m_i \in \mathbb{R}^+$. The feedback magnitude corresponds to the reflection strength of the detected object. This depends heavily on the material of the object and reflection angle. For example, metal usually generates a much higher magnitude than human flesh. Velocity values v outside the range $[-5.2m/s, 5.2m/s]$ are mapped to the range using modulo: $v_i' = (v_i + 5.2 \ mod \ 10.4) - 5.2$ due to the doppler ambiguity. The radial velocity itself is determined by analyzing how the detection's motion has altered the frequency of the returned signal.

A visual comparison of example point clouds from the Astyx HiRes2019 Dataset's lidar and radar sensor can be seen in Figure 3 along with a camera picture for interpretation of the scene. Here the lower resolution of the radar data can easily be observed, along with the generally different shapes formed by the obtained detections. Particularly, in

the left lidar point cloud, the car coming from the other direction (see camera picture) has a vague recognizable outline while in the right radar point cloud, the car is almost indistinguishable from other detected shapes.

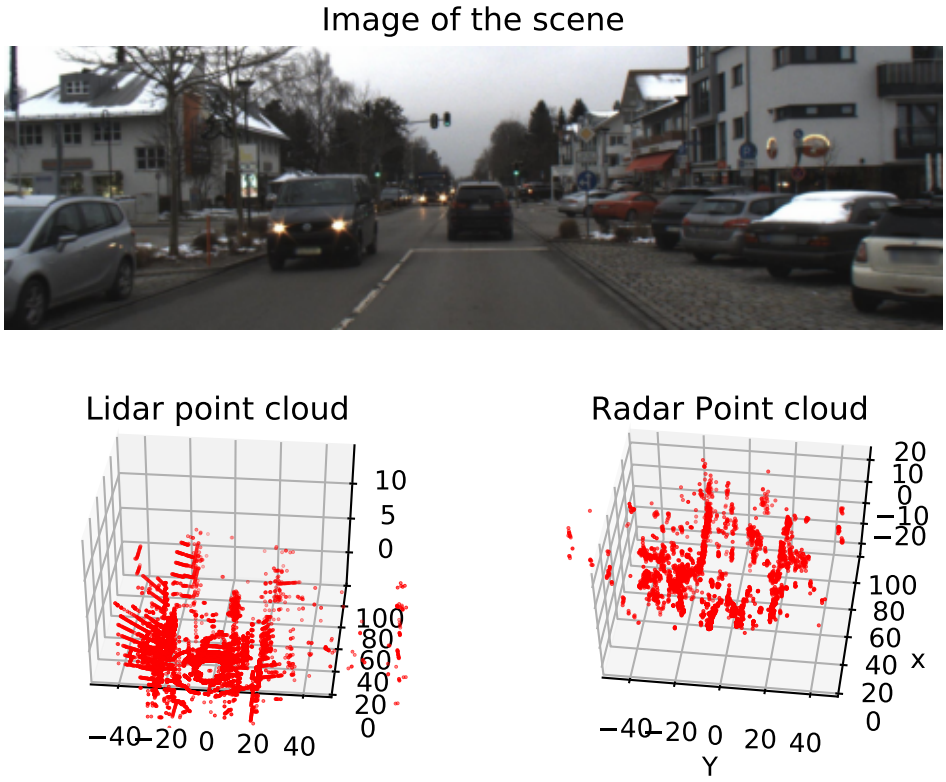


Figure 3: Comparison of the radar and lidar point cloud of the same scene. All axis scales are in meters. The obtained detection points are plotted in red.

2.2 Kitti Dataset

Since the Astyx Dataset contains no accurate ground truth information on the vehicles position and orientation, we also use the Kitti Odometry Dataset [4] in some of our approaches.
This dataset consists of sensor information captured from a driving vehicle in a total of 21 sequences. The following sensors are available:

- Lidar Data from a Velodyne HDL-64E
- Stereo images from two front facing cameras
- Accurate location (accurate to 0.01m) and orientation (accurate to 0.03°) from a OXTS RT 3003 localization system

This data was also captured at 10Hz and thus is suitable for comparison to the Astyx GmbH/Cruise dataset.

3 Related literature

Here we give an overview of the literature on SLAM. We mostly focus on Radar SLAM and give short references to visual and lidar SLAM methods.

3.1 Classical Approaches

Compared to lidar or camera based SLAM approaches, there are very few attempts at solving the SLAM problem using radar sensors in a similar measurement setup as in our datasets [2]. Furthermore, due to the sparse density, medium range and high amount of noise of radar point clouds, landmark based approaches, where real-world static landmarks are used as reference points for orientation, are not well suited for the radar datasets in general. The most promising approach for our use case was the one described in [5]. Here, the authors also had a single front-facing radar sensor setup. They proposed a graph-based SLAM variant using *Iterative Closest Point (ICP)* algorithms [6] to match single consecutive radar scans together. Graph-based SLAM methods basically model the problem as a graph. In simplified terms, nodes in the graph represent the positions of the robot and the scan at that position, and the edges describe the translation and rotation between the nodes.

Another similar radar based graph SLAM approach is proposed in [7], where the system consists of three major parts: Pose Tracking, Local Mapping and Loop Closure. Pose Tracking includes the extraction of features from the radar scans and the detection of similarities between them to estimate the pose transformations. Local Mapping depicts these transformations on a map and Loop Closure finds and eradicates loops. The authors also tested their approach against the common Camera and LIDAR based methods ORB-SLAM2, LOAM and SuMa for difficult weather conditions. Except for some minor problems with snow, Radar SLAM was able to function in all weathers while cameras and LIDAR sensors were seriously hindered by fog, rain and snow.
A problem we presumed to have with this approach was that feature extraction and scan matching are applied on single scans. However, due to sparsity of our radar scans, this wouldn't give us satisfying results. Instead, we combine multiple scans to submaps beforehand. This idea stems from [5], as they had the same problem with sparse radar scans.

Contrary to graph SLAM, the occupancy grid map approach models its map as a grid/matrix of small cells. Every cell contains the information of how likely it is that this cell is occupied. By assuming the position of the robot is known, each radar scan updates the occupation values of corresponding cells. This technique is used for example in [8]: The paper describes the use of a 360° microwave radar sensor to build occupancy grids for every scan. Together with scan matching, which estimates the transformations between scans/grids, odometer sensors and global map updating, this yields a high resolution SLAM system called R-SLAM [8]. The approach looks promising, however, our Radar point cloud only contains a 110° front faced view.

Prevalent problems, that SLAM systems have to face, are described in [3] as Robustness, Scalability and Map representation. Robustness analyses how well the system performs

if we consider algorithmic or hardware failures. Algorithmic failures, for example, can be false detections of loops, incorrect data association because of outliers or poor sensor data because of unpleasant environments. Hardware failures, on the other hand, are malfunctioning or aged sensors which then adversely affect the overall sensor measurement. Scalability describes the problem of worse performance and increasing memory due to the growing map size. While indoor environments can easily be processed, outdoor usage will probably need some kind of parallelization or sparcification. The latter describes techniques to use fewer but more useful data points, which could then yield to undetected loops. The aspect of map representation focuses on how to model the environment such that we get an acceptable trade-off between performance, memory consumption and accuracy.

3.2 Deep learning based approaches

Both the front end and back end part of SLAM can be solved with deep learning based approaches. We will first focus on the front end - the odometry.

3.2.1 Deep odometry on point clouds

For the front end, a deep network typically models a function that maps from the previous position and current observation to the current position. Since we can compute the new position $x_k = P_{k-1,k}\ x_{k-1}$, in most cases the relative pose is estimated (here $P_{i,j}$ refers to the matrix representing the relative pose between frame i and frame j). In this case the function $h'(z_{k-1}, z_k) = P_{k-1,k}$ is modelled by some deep network, where z_i is the observation at frame i.
It is known that relative pose estimation from images is possible using convolutional networks [9, 10, 11]. This is typically called deep visual odometry (DVO).
Since we are working on radar point clouds, we will focus on architectures that can work with raw point clouds. Given the formulation of the front end of SLAM as a pose estimation problem, we can try to estimate the pose between two point clouds. This is also known as point cloud registration, where a relative pose from a source point cloud S to a target point cloud T is sought. There are two common ways of solving this problem:

- Approaches based on Iterative Closest Points (ICP): In the classical ICP algorithm, each point is assigned the corresponding closest point in the target point cloud. Then the distance between those points is minimized (further explained in Section 4.2.3).
  Instead of using the minimal distance, one can use a deep network to find point correspondences. This can be done either by generating the correspondences by the network itself, or have the network map points to a feature space and then finding correspondences by minimizing the distance in feature space.

- Direct methods: These methods directly predict the relative pose of two point clouds using a deep network.

We will now quickly summarize the most important papers for both ICP based approaches and direct methods.

ICP based approaches:
The paper on Fast Convolutional Geometric Features (FCGF) [12] and 3DMatch [13] propose deep networks based on 3D convolutions that compute features for each point in a point cloud. They then show that these features can be used to register point clouds. D3Feat [14] proposes a deep network that simultaneously detects keypoints in point clouds and computes useful features for keypoint matching.

Direct methods:
PointNetLK [15] uses a reformulation of the Lucas-Kanade algorithm [16] and PointNet [17] to compute the pose between two point clouds. PCRNet and iPCRNet [18] are extensions of this work that replace the pose improvement step of Lucas-Kanade with another deep network. Deep Closest Point [19] is an extension of the classical Closest Point algorithm, which replaces the normal closest point approximation by a transformer architecture [20]. This produces a "soft" point correspondence which is used to compute the transformation between point clouds.

Direct methods are usually easier to implement and more efficient in training and inference. Thus we will focus on these in our experiments.
There are also approaches that can be seen as a hybrid between deep visual odometry and odometry on point clouds [11, 21, 22, 23]. They map the point cloud to a 2D panoramic image and then use a convolutional network to estimate the pose. Since these approaches only work for dense point clouds, we will not explore them any furhter in this project.
Worth mentioning are also approaches based on scene flow [24, 25]. They extend the definition of scene flow to 3D point clouds. For each point in the source point cloud they try to estimate a translation (called flow) so that the point clouds match. These methods have the advantage that they allow filtering out moving objects and thus can estimate the transformation between the point clouds in a more stable manner.

3.2.2 Deep learning approaches for loop closing

Altough some of the deep odometry approaches employ recurrent networks to model temporal dependencies in the measurement sequences and thus view the problem as a sequence to sequence modeling problem, they can not be viewed as SLAM. This comes from the fact that they only estimate a pose for each frame once, while in a real SLAM systems, the estimates for previous states might be refined when new observations come in and a loop is detected.
There are two deep learning based approaches for loop closing:

- Use a deep network to detect loop closures. This can either happen by computing features for each frame and then detecting loop closures in the feature space or by directly estimating the probability of a loop closure between two frames.

- End to end based approaches: Here SLAM is viewed as a sequence to sequence learning problem. A deep network takes in the sequence of observations Z and estimates a sequence of states X. When a new observation is added to the observation sequence, the network receives the whole sequence again to estimate a new sequence of states.

In [26], the Scan Context image [27] is used as the input to a convolutional network to compute the place index - a unique identifier of a grid cell in the map. They show that this apporach can reliably detect loop closures after being trained on training data from a single day.
In [28], the authors propose to model SLAM as a sequence to sequence learning problem. They consider images generated from a video game engine and first use a convolutional network to estimate relative poses between frames. They then use a Transformer based architecture to model the graph optimization that happens in classical SLAM algorithms. They call this approach Neural Graph Optimization. They show that this neural graph optimization is indeed able to improve the estimated poses coming from a visual odometry. Due to time constraints and the lack of ground truth loop closures we could not evaluate deep learning approaches on loop closing for radar data.

4 Methodology

In the following we will outline our approach to the problem. We will first describe how we obtained ground truth information for Astyx/Cruise data, and will then explain classical and deep learning based approaches for radar SLAM.

4.1 Ground truth

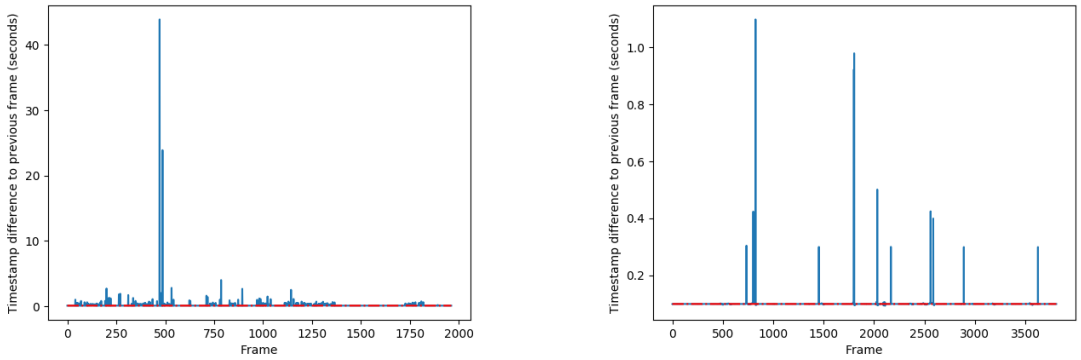
While we have some ground truth information of the position and orientation of the vehicle at each frame from the GPS, we suspect that this is not accurate enough for usage as frame to frame ground truth information. Thus we need to extract ground truth pose information from the data in some other way. We use the available lidar data to try to extract ground truth poses using Lidar SLAM algorithms, as they are usually very exact and have existing implementations. We evaluate the following approaches

- Surfel-based Mapping (SuMa) [29]: Uses Iteratively Closest Points (ICP) to get frame-to frame pose estimation (odometry) and then uses a surfel map for efficient loop closing.
- LOAM: Lidar Odometry and Mapping in Real-time [30]: Extracts feature points at edges from Lidar scans and uses those feature points for odometry and loop closing.

These approaches are well tested on the KITTI dataset [4] and return good results. Unfortunately, we find that on the Astyx data they fail, either during point cloud registration or during loop closing. We suspect that this is due to missing frames in the data: When information is captured from different sensors, one has to make sure that the measurements roughly happen at the same time. When this is not the case, for example when sensors are not synchronized properly or one sensor is not working properly, the frame of this measurement has to be dropped. This happens in both sequences from the Astyx data, as can be seen in Figure 4. We suspect that this is the reason why the Lidar SLAM algorithms are not working properly. We where thus only able to obtain ground truth pose estimate for sequence B for the first 2278 frames, since after that a loop closing error occurs, rendering the poses unusable.
Due to the missing frames we solely rely on the GPS data for estimating the relative poses between consecutive scans. Estimated relatives poses based on the CAN bus data suffer from enormous drift and are thus not usable.

4.2 Classical approaches

As mentioned in chapter 3, the experiments in [5] had a similar measurement setup and a well defined graph-based SLAM approach. The system structure of this approach also served as an inspiration for the algorithm structure for our classical approach. Our structure is illustrated in Figure 5.



(a) Sequence A (b) Sequence B

Figure 4: Difference of the timestamp of the current frame to the last frame for both sequences. At 10Hz, this should have a value of $0.1s$ (indicated by the red line) when no frames are dropped.

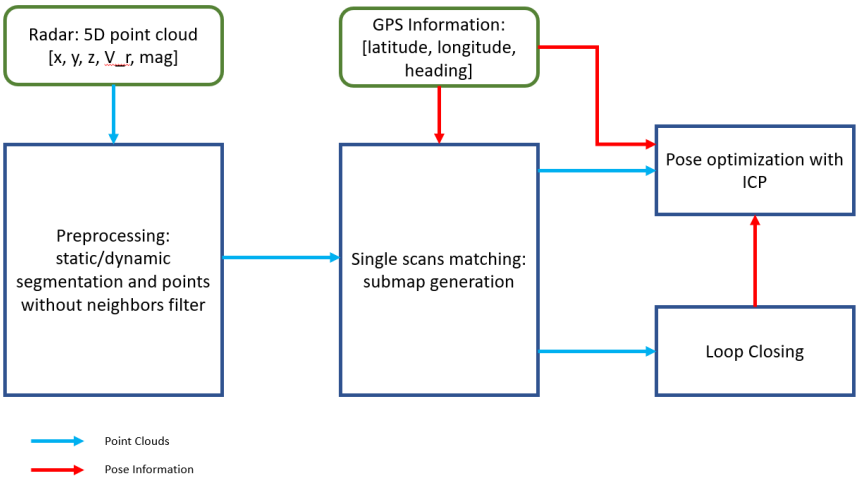


Figure 5: System architecture of our classical *SLAM* approach. Design inspired by [5].

4.2.1 Radar Preprocessing

The Radar sensor measurements are provided as a 5D point cloud dataset with 3D pose information, the relative radial velocity V_r of the detection point, and the feedback magnitude of detection.

STATIC TARGETS FILTERING:

For pose estimation we are interested only in static points. Therefore we try to discard any kind of moving points from the point cloud. This is made by possible by the Doppler shift phenomenon in waves, which for radar makes relative radial velocity measurable for each detection. To classify the radar point into stationary and non-stationary we used the

approach discussed by Kellner *et. all* [31]. Here random sample consensus, or RANSAC, [32] is used which can generally eliminates outliers and clutter from data.

The main idea in this context is that static points all have a calculable theoretical radial velocity based on their position to the car which can be compared to the measured radial velocity. Moving objects would then have a greatly different measured radial velocity than the one they would have if they were static.

To obtain this theoretical radial velocity, a curve is fitted to a random batch of the data usually containing both static and moving targets. Since most points are static, the fitted curve gives the relation between positions and radial velocity for static objects. However, it is slightly inaccurate due to the few moving points also being considered in the curve fitting. The algorithm then checks how many of the points in the whole frame are a good fit to this curve to evaluate its accuracy. This whole process is done iteratively until the curve with the most inliers is found.

The fitting is done using scikit-learn's [33] polynomial fitting with degree parameter 2 for a parabola and taking the point's direction of arrival (see Figure 6) as input. The minimization carried out is described in Equation (1).

$$\begin{aligned} &\min_{a,b,c\in\mathbb{R}} a\cdot\theta^2 + b\cdot\theta + c - V_r \\ &\text{with } \theta : \text{Direction of arrival [rad]}, \quad V_r : \text{Radial velocity [m/s]} \end{aligned} \tag{1}$$

Points that fit this curve are all static targets while the others are considered outliers - the moving targets. This is illustrated in Figure 7, where the same scene as in Figure 3 was used.

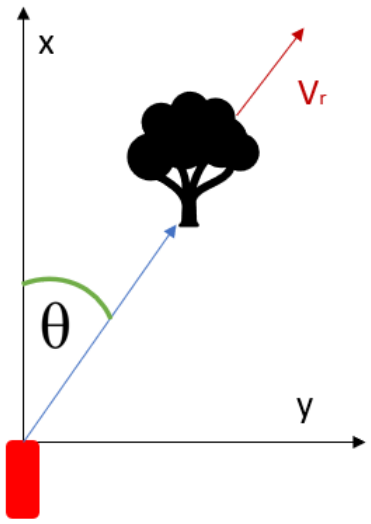


Figure 6: Illustration showing the angle of arrival θ of a detected object and its radial velocity V_r relative to the ego vehicle (here in red)

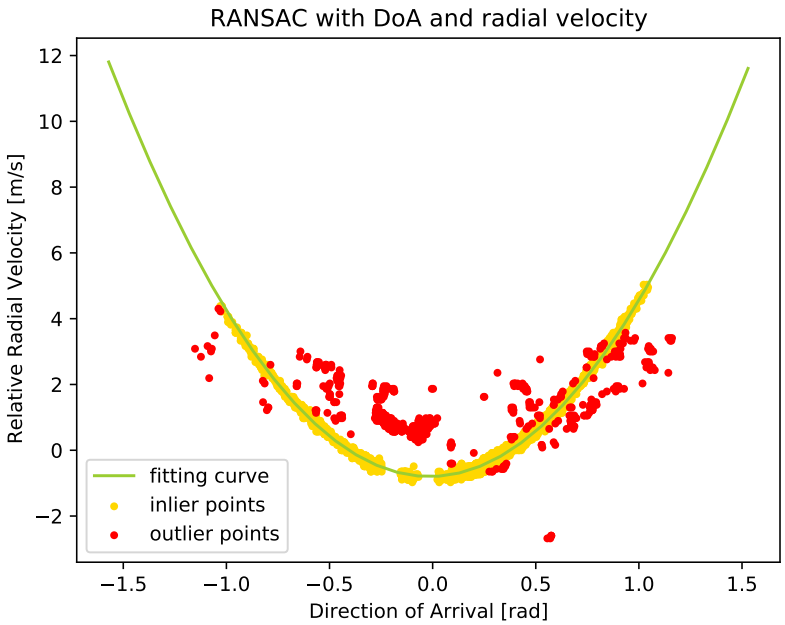


Figure 7: Application of RANSAC for filtering static targets by fitting a polynom. The relative velocity of the single radar points (yellow & red) are obtained with the Doppler effect. The resulting fitting curve for static targets is represented by green curve. The yellow points represent the desired static points of the point cloud.

Note that if most points are not static as is assumed here, this process would do the opposite task and keep only the moving targets. This described problematic can lead to poor results, especially in a relatively empty scene with mostly other moving vehicles with almost the same linear velocity, such as it occurs in slow moving traffic multi-lane scenario.

Alternatively to *RANSAC*, the static points could also be filtered by *MLESAC* [34] or *PROSAC* [35]. The difference to the normal *RANSAC* is that in *MLESAC* the quality of the sampled consensus set gets evaluated by its likelihood. Thus, *MLESAC* is less sensitive to a sub optimal noise threshold parameter. In *PROSAC* the sampling process of the consensus set is guided by a priory information of the input parameter, for example using the ego vehicle velocity to estimate a radial velocity V_r for static parameters, which will be likely to produce an inlier point.

SUBMAPS:

To circumvent the sparsity of radar scans, submaps are generated from multiple scans. To do so the scans belonging to one submap must be converted to common cartesian coordinates with the GPS information. Using the combined GPS location and orientation, called pose, successive scans can be merged together by applying the translation and rotation from Equation (4).

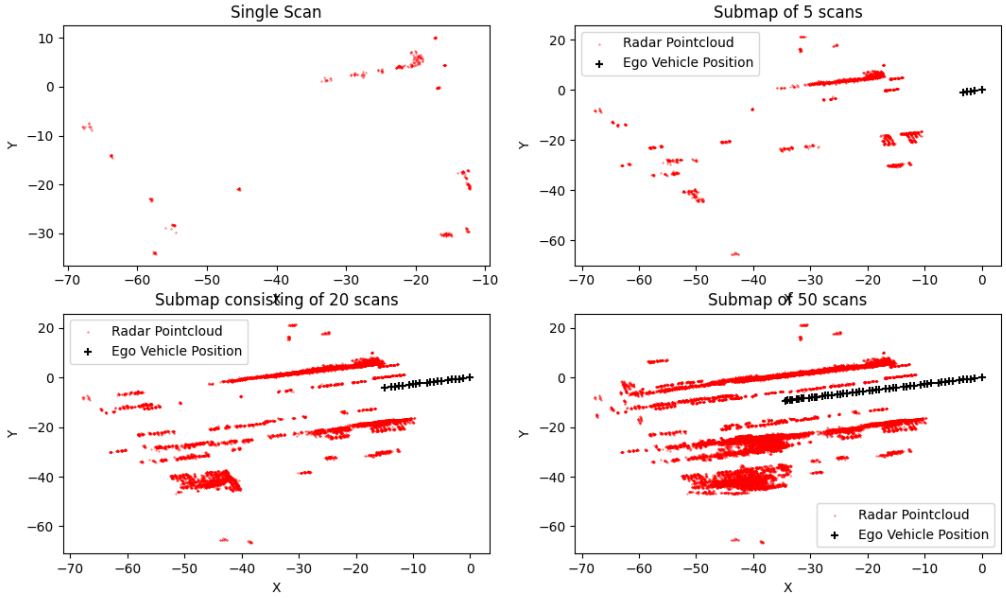
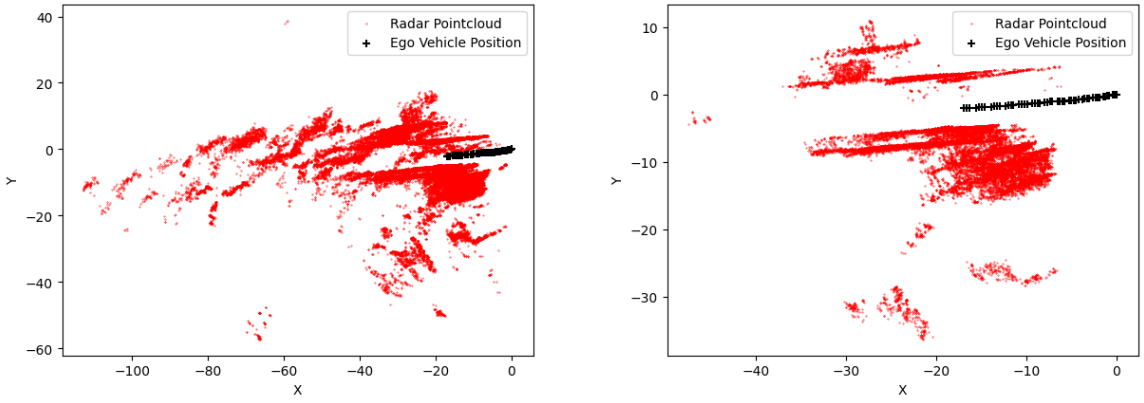


Figure 8: Creation process of the submaps. Radius outlier filtering is done here with a radius of 2 meters and demanding at least 10 neighbors. All axes are in meters.

OTHER FILTERS:

Before generating the submaps, we apply a radius based outlier filter on individual scans which removes points with no neighbor in an $1m$ radius as clutter. This quickly removes points that won't need to be considered in the numerically more complex filter described next.
After generating the submaps, an extra filter is used to remove points with less than a certain amount of neighbour points within a certain radius. This helps remove a large portion of noisy points which hurt the performance of the next steps like loop closing.



(a) Submap without neighbouring filtering (b) Submap with neighbouring filtering

Figure 9: Result of the additionally applied filters. The neighbour filtering is applied here with $2m$ radius and with 30 required neighbours in it. All axes are in meters.

4.2.2 Odometry

In theory, consecutive scans could also be merged together in common cartesian coordinates using odometry data of the vehicle instead of GPS. However, frame drops add unrecoverable drift when using just odometry, ruining the accuracy of future locations even if the frames work accurately. GPS information is therefore much more suitable as frame drops only lose the position of the particular dropped frames and not all future ones which will stay just as accurate.

4.2.3 Scan Matching

To improve the accuracy of the relative transformation between two consecutive Radar frames, a scan-matching method based on 3D point-to-point *ICP* algorithm [6] is used. From the GPS information, we are able to obtain a heading and localization of the two successive scans. As an initial guess for the transformation matrix described in Equation 3, the difference between the two GPS locations and headings is used. The heading difference θ leads to a rotation matrix $\mathbf{R}$ with Equation 2.

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & \mathrm{t}_x \\ \sin\theta & \cos\theta & 0 & \mathrm{t}_y \\ 0 & 0 & 1 & \mathrm{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \text{or short} \quad \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathrm{t} \\ 0 & 1 \end{bmatrix} \tag{3}$$

The main idea behind the *iterative closest point (ICP)* algorithm [6] is to minimize the squared Euclidean distance between the closest point pairs in a scan and its estimate based on the previous scan.
Using the initial transformation matrix, the first scan is translated and rotated according to Equation (4), giving an estimate of the obtained successor scan. Point pairs are selected by choosing all points with its nearest neighbour lying within a threshold distance. The distance between pairs must then be minimized with Equation (5) to obtain an improved transformation matrix.

$$\begin{aligned} &P'_{\text{estimatedSuccessor}} = \{p'_1, ..., p'_n\} \\ &\text{with} \quad p'_i = \mathbf{R}_k P_i + \mathrm{t}_k, \text{ and } \quad \mathbf{R}_k = \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} \end{aligned} \tag{4}$$

The above described process is repeated until it reaches convergence. In [6] it was shown that if all points p_i belong to the $3D$ model M, *ICP* will converge monotonically to the nearest local minimum. In our case the model M represents all static objects of the scene.

$$\mathrm{t}_{k+1}, \theta_{k+1} = \arg\min_{\mathrm{t}_k, \theta} \sum_{(i,j)} \|\mathbf{R}_k(\theta)\mathrm{p}_i + \mathrm{t} - \mathrm{q}_j\|^2 \tag{5}$$

While experimenting with *ICP*, we discovered that the z-component of the geometric point cloud is too noisy for scan matching. Therefore, we decide to not solely rely on geometric information. As an additional feature the reflection magnitude of the radar detection points is used. The principle of not solely relying on geometric information for matching during *ICP* is evaluated more closely in [36]. Here, the authors use the color intensity value of laser scans as a third dimension for the matching process.
Due the sparse point cloud density of single radar scans, the scan matching of consecutive scans does not lead to an improvement of the transformation. Hence, we are applying the same scan matching principles on a submap level to improve the alignment of successive submaps. Therefore, we are assuming here that the drift inside a submap of N scans is small enough to be negligible. Here, N is a design parameter, which has to be chosen based on measurement frequency.
For the implementation of *ICP* we use the *Open3D* library [37].

4.2.4 Loop Closing:

The classical loop closing approaches we considered are both based on evaluating the similarity of two submaps in a way that is not invariant to the observer's point of view. This means, for example, that two submaps which are received while driving at the same location, but in opposite directions, should be able to be recognized as very similar, which isn't obvious when considering just the submap point clouds. Matching the current with all previous submaps in a brute-force method quickly becomes computationally infeasible as the number of submaps increases. We therefore need to compute signatures for the submaps from their point clouds, such that we can find similar ones with an efficient nearest neighbour search for signatures.

The considered signature generation approaches are Geometric Landmark Relations (GLARE) [38] and Geometrical Surface Relation(GSR) [39] :
GLARE: For GLARE, feature points l_i from the submap are extracted with AKAZE [40] and the Euclidean distance $\rho_{i,j}$ are computed, as well as the angle $\theta_{i,j}$ between every two different features l_i and l_j. This can be seen in the two graphs on the left hand side of Figure 10.
These value pairs are then plotted together in a 2D histogram bin. Furthermore, a multivariate gaussian centered around these value pairs is added to allow for uncertainties in relative distance and bearing estimates (uncertainties originating from the sensors in this case). The resulting histogram serves as a scan signature and can be compared to other GLARE signatures via the L1-norm.
To further illustrate the function of the multivariate gaussian: if its variance parameter is set to 0, submaps would need the exact same relative feature positions to be considered similar. If the variance is set way too high, all submaps would obtain similar signatures and it wouldn't be possible to distinguish submaps containing the same scene.

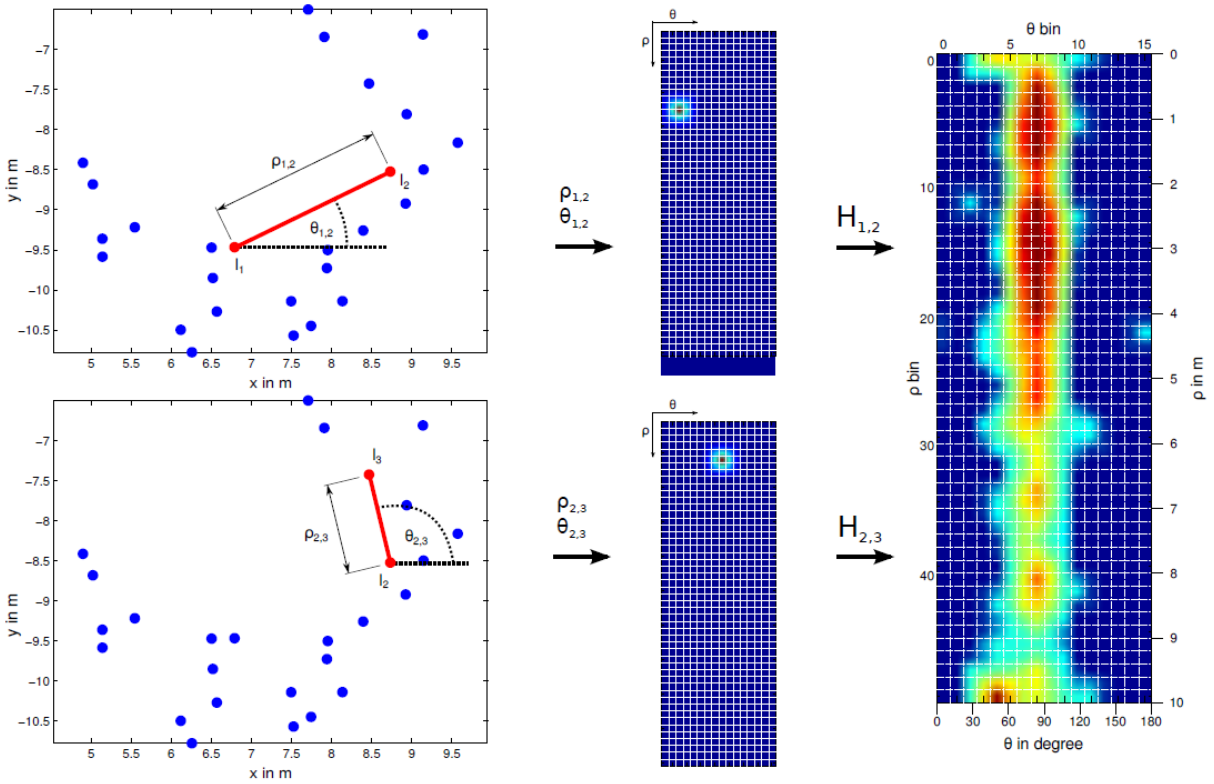


Figure 10: Generation of GLARE: Each landmark relation, given by its orientation $\theta_{i,j}$ and distance $\rho_{i,j}$, is incorporated as a multivariate Gaussian. The scan signature S is obtained by summing over all histograms $H_{i,j}$. The Figure is taken from [39].

GSR: At first GSR projects the point clouds onto a 2D grid and computes mean μ_i and covariance matrix $\sum_i$ using the points of every non-empty cell l_i (see Figure 11).

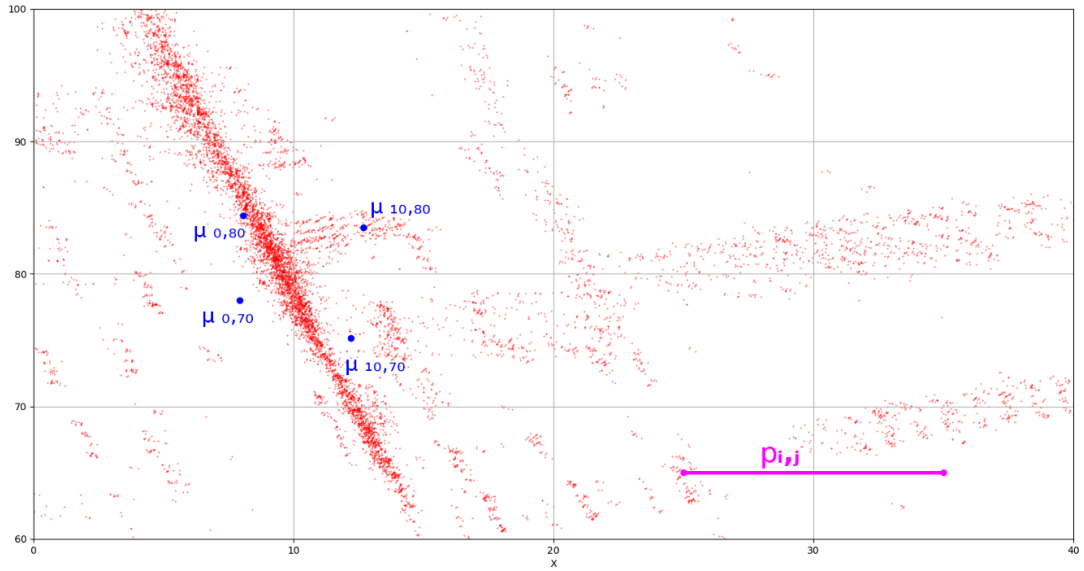


Figure 11: Part of a submap with cellsize 10. Exemplary computed means μ_i of four cells in blue and Euclidean distance of two cells in magenta

Similar to GLARE, the algorithm then calculates the Euclidean distances $p_{i,j}$ and the angles $\theta_{i,j}$ of any two cells. But instead of computing the angles with the point coordinates, like in GLARE, GSR uses the orientation θ_i of a cell, which is estimated with the eigenvector e_{min} with the smallest eigenvalue of the covariance matrix:

$$\theta_i = arctan2(e^y_{min}, e^x_{min}) \tag{6}$$

π will be added to θ_i if the eigenvector e_{min} points away from the sensor's origin. This is determined by:

$$\theta_i = \theta_i + \pi \qquad if\ arctan2(\mu^y_i, \mu^x_i) + \pi - \theta_i \geqslant \tau_{max} \tag{7}$$

τ_{max} in this equation is a threshold that is set to $\pi/3$, according to [39]. After thtat, the bearings are computed by substracting the two orientations:

$$\theta_{i,j} = \theta_i - \theta_j \tag{8}$$

In the end, the value pairs of distance and bearing are also assigned to histogram bins, where the histograms, analogously to GLARE, are used as GFS signatures.

Both our approaches are based on generating one of the previously described signatures for each submap (GLARE or GSR). Using a nearest neighbour search the closest submaps are considered as loops candidates. Successive submaps tend to have similar signatures and are therefore ignored. Loops candidates whose euclidean distance fall under a certain treshold are then seen as a detected loop. Calcuating the transformation matrix between the two submaps of the detected loop makes it possible to correct the drift accumulated over the journey between them.
It is important to note that detecting false loops introduces a heavy distortion to the total map and localization of the vehicle as it connects two points of the trajectory together which don't actually belong together. Therefore it is generally a good idea to stay conservative in the acceptance of loop closing candidates.

4.3 Deep learning based approaches

In addition to evaluating classical radar SLAM approaches, we also evaluate different deep learning based approaches for the front end part of SLAM.
We will first discuss different deep learning models for processing point clouds and then show how we applied them to the front end part of radar SLAM.
In Section 3.2.1 we briefly introduced some papers that work on deep odometry on lidar point clouds [21, 11, 22, 23]. They map these point clouds onto a cylindrical projection and then use convolutional neural networks to predict relative poses. This approach is only possible for dense point clouds - such as those coming from lidar sensors, as otherwise the images resulting from the projections would be very sparse. Thus we instead focus on deep models that work directly on point clouds.

4.3.1 Deep models for point cloud processing

One of the most challenging aspects of point clouds is that they are unordered sets of 3D points. Therefore normal models for sequential processing don't necessarily work very well. Some works propose to voxelize the point cloud - meaning discretizing the 3D space into discrete voxels and then assigning points to their respective voxels. Then 3D convolutions are used to process these voxelized point clouds. Approaches based on voxels typically suffer from either low resolution due to big voxel sizes or high computational cost for the 3D convolutions.
PointNet [17] instead uses the raw point clouds. Given a a set of points $X = \{x_i \in \mathbb{R}^3\}$ it computes a feature for the point cloud by

$$PN(\{x_1, .., x_n\}) = g(\{f(x_1), .., f(x_n)\}) \tag{9}$$

where $f : \mathbb{R}^3 \to \mathbb{R}^D$ maps each point to a high dimensional feature vector and $g : P(\mathbb{R}^D) \to \mathbb{R}^D$ is a symmetric accumulation function. In practice, f is implemented by a simple Multi Layer Perceptron (MLP) and g is simply the maximum over the single dimensions: $g(F)_j = \max_{f \in F} f_j$. This feature can then be used to predict desired outputs, for example by using another MLP.

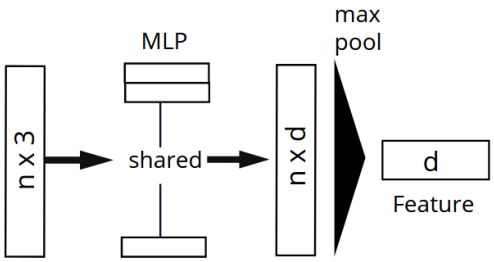


Figure 12: Simplified architecture of PointNet for computing a point cloud feature

Taking inspiration from convolutional neural networks in image processing, one can also try to define convolutions on point clouds. Convolutions typically work on a neighborhood around the interest point. To make this possible, one first has to define the neighborhood for a point cloud. This can be done by a k-nearest neighbor approach or by including all points in a certain radius. This gives edges e_{uv} between points x_u, x_v. For each of those edges a feature vector $f_{uv} \in \mathbb{R}^{\mathbb{D}}$ is computed by an MLP g, typically receiving the position and, if available, feature vectors of the nodes: $f_{uv} = g(x_u, x_v, f_u, f_v)$. Then the feature $f_u \in \mathbb{R}^D$ of each node x_u is aggregated from the respective connecting edges, typically via max-pooling: $(f_u)_j = \max_{(u,v) \in E} (f_{u,v})_j$
If a k-nearest neighbor graph is used, this yields the dynamic edge convolution from Dynamic Graph Convolutional neural networks [41], while using a radius graph yields the set convolution layer from FlowNet3D [24].
A sequence of these convolutions are typically applied after each other. Between convolutions one can choose to sub sample the point cloud (eg. by farthest point sampling or random sampling) to reduce the computational cost. To get a global feature for the whole point cloud, global max pooling over all points can be performed in the end.

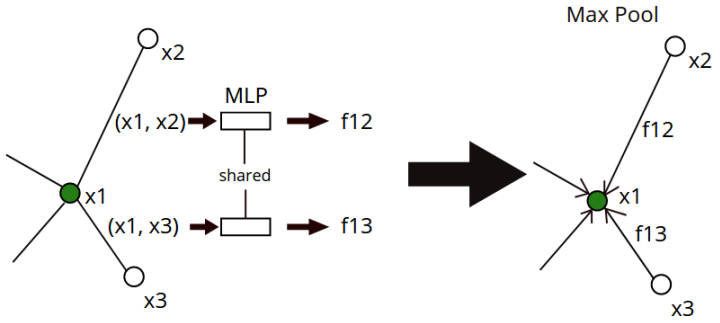


Figure 13: Point cloud convolution based on the neighborhood graph. First features for all edges are computed using an MLP, then those features are accumulated in the nodes with max pooling.

4.3.2 Deep odometry on point clouds

Inspired from the direct approaches (see Section 3.2.1), we estimate the pose between two point clouds X, Y, by using a deep model (PointNet, DGCNN or FlowNet3D) that maps the point clouds X, Y to their respective feature vectors F_X, F_Y. We then concatenate those feature vectors and use a MLP to predict the relative pose.

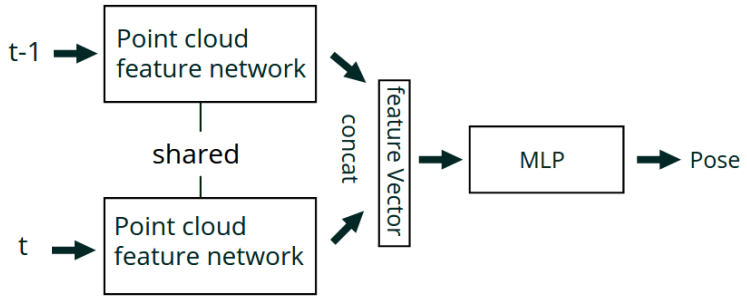


Figure 14: General network architecture to predict poses of consecutive point clouds.

While a pose can be represented by a matrix $P = \left\{ \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} | R \in SO(3), t \in \mathbb{R}^3 \right\}$ acting on points in homogeneous coordinates, where R is the rotation matrix and t a translation vector, this representation is not suitable as direct output for a deep network. The rotation matrix has to satisfy $det(R) = 1$ and $RR^T = I$ in order to be a proper rotation matrix, which is not guaranteed if it is being predicted by a neural network.
If a 3D pose is predicted, one instead has to resort to other representations of the rotation. Suitable choices are either unit quaternions or the rodrigues vector w, which defines a rotation around the axis w with angle $||w||$. These representations are somewhat problematic, since they have singularities that have to be dealt with properly.
If one instead knows that all poses only rotate around the z axis and translate around the x and y axis, instead a 2D pose can be predicted. This can easily be represented by a translation vector $t \in \mathbb{R}^2$ and angle θ without any singularities.
We use the deep network to predict a 2D relative pose and then use a L2 loss for training. We find that normalizing the ground truth 2D pose across each of its dimensions indi-

vidually greatly improves performance. We suspect that in the unnormalized case the L2 loss leads to different weights due to the scales of translation (typically $\in [-0.1m, 0.1m]$) and rotation (typically $\in [-0.06°, 0.06°]$).

For the feature computation we evaluate different models, namely Pointnet, DGCNN and Flownet3D. The last two are based on graph convolutions.
Since we could only recover ground truth pose information sequence B of the astyx training data, we resort to the KITTI odometry dataset [4] to train the models and test their generalization capability. We sample 1000 random points from the lidar point clouds to simulate the sparcity of radar point clouds and extract the relative pose between consecutive frames as ground truth. In Section 5.2.1 we demonstrate, that our selected model can also learn to recover poses from radar point clouds.
To evaluate the models we compute the 25-th, 50-th and 75-th percentile of the translational error and absolute rotational error of the predicted relative poses, which does not evaluate the total drift of the predictions, but only the local difference to the ground truth.
Traditional SLAM algorithms often improve their performance by registering the current point cloud not only to the point cloud of the previous frame, but to a sub map from the N previous frames, which is simply built from the previous pose estimates. We observe that doing this slightly improves performance for deep odometry as well.
In some related approaches, the estimated pose is refined with a few iterations of Iterative Closest Points. We observe that this improves our pose estimates quite a bit as well.

4.3.3 Training details

We use the unmodified architechtures of PointNet [17], DGCNN [41] and FlowNet3D [24] for feature computation. The resulting features f_1, f_2 of two consecutive point clouds 512-dimensional feature vectors and are concatenated to yield a 1024-dimensional feature vector. This is processed by an MLP with hidden layer sizes 1024, 512, 256 and an output size of 3. We use the ReLU nonlinearities [42] followed by Batch Normalization [43] in all layers except the output layer of the feature networks and the final MLP. We use the Adam optimizer [44] with the default learning rate $\alpha = 10^{-3}$ and halve this learning rate after every epoch. All models are trained for a total of 5 epochs. The PointNet based model is trained with a batch size of 64, while DGCNN and FlowNet3D are trained with a batch size of 8 due to their increased memory requirements.

5 Results

5.1 Classical approach

Due to the lack of exact ground truth data in the Astyx datasets, the classical approaches implemented around their setup could not be evaluated quantitatively. However, by plotting the resulting trajectories and point clouds qualitative assessments could be carried out.
The static/dynamic segmentation with RANSAC shows good filtering results in most frames. The evaluation of the performance can only be done qualitatively however there is no ground truth with points labeled moving or static. With the help of the camera data of the Astyx hires dataset it is possible to manually assess if moving objects are successfully filtered out. One example is shown in Figure 15 where both moving cars in front of the ego vehicle generate points originally but are filtered. Note that camera's view is much narrower and only shows a few meters to the sides while the radar shows points more than 40 meters away

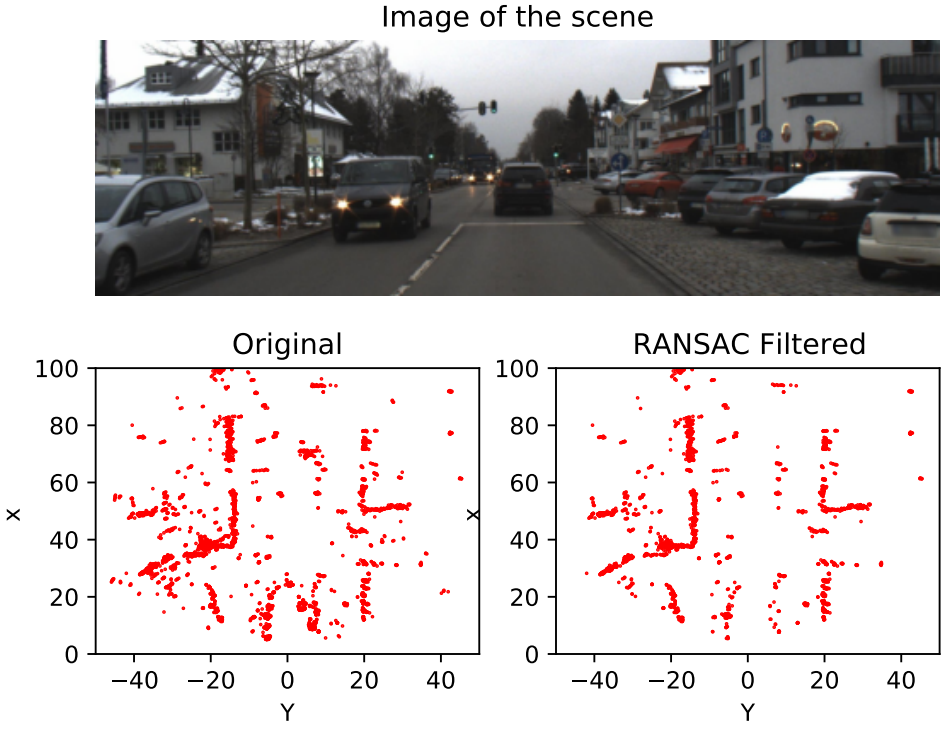
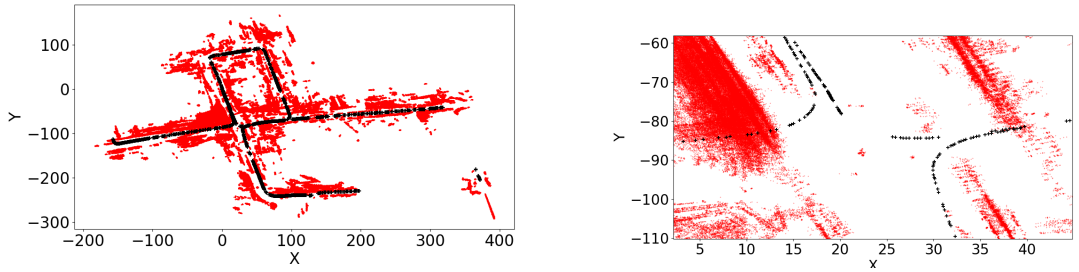


Figure 15: Result of static detections filtering process, described more detailed in 4.2.1. Point cloud is illustrated in $2D$ due to the high clutter of the $z-$component which would decrease the visibility. Both axes are in meters.

While the obtained maps generated with ICP and GPS information generally look decent, street intersections where the vehicle quickly changes direction stay problematic. This is partly due to the static/dynamic segmentation keeping stationary cars in the middle of the road (which is common in street intersections) but also due to the GPS' heading

information being less accurate in a 90 degree turn where the orientation changes rapidly. This phenomena can be observed in subfigure 16b.



(a) Compilation of all scans

(b) Zoomed version on a intersection scene

Figure 16: Obtained map zoomed on a intersection (which was visited twice by the vehicle) highlighting the rotations being off when quickly turning by 90 degrees and points in the middle in the road due to stationary cars. Frame drops are also visible here. All axes are in meters.

Loops in the Astyx dataset can successfully be detected using GLARE and GSR signatures of submaps. However, the low number of loop examples in the data makes it difficult to obtain a meaningful quantitative performance measure. In the example 17, the vehicle drives through the same intersection twice which is recognized as a loop. The problems described above in street crossings can also be seen here when zoomed in.

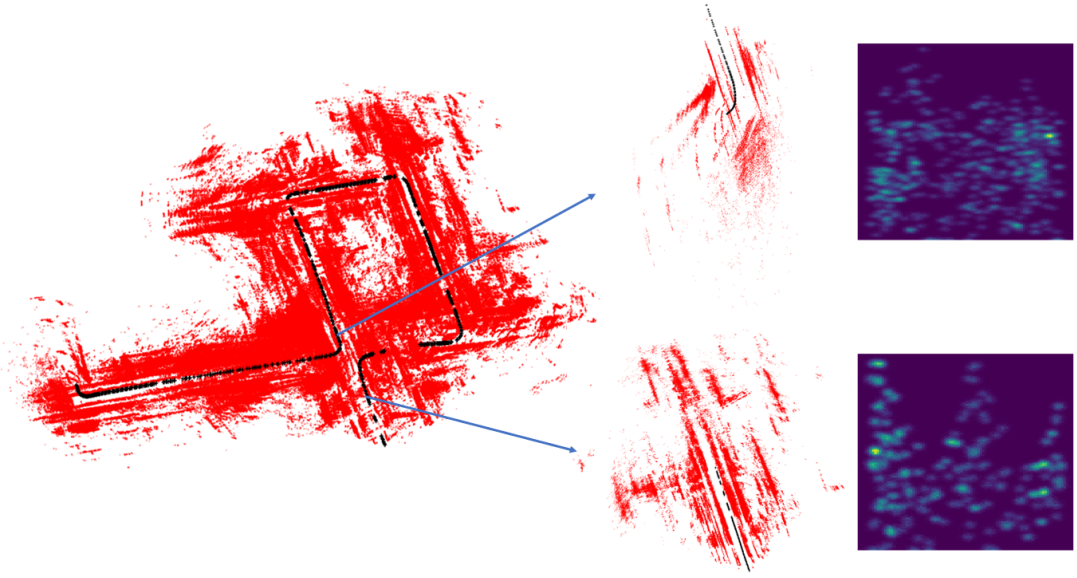


Figure 17: Example of detected loop in the Astyx dataset using the GLARE signature approach. The total drive trajectory and their point clouds can be seen on the left while the submaps containing the loop are shown in closer detail. Both submaps also have their GLARE signature shown on the right.

5.2 Deep odometry

We first evaluate PointNet, DGCNN and FlowNet3D as feature models for the network architecture from Figure 14. We train on sequences $00-07$ of the Kitti Odometry dataset and run evaluation on sequences $08-10$. The results for sequence 8 can be seen in Table 1. The results for sequences 9 and 10 are comparable and are not shown for space reasons. From this two things can be seen:

1. The models successfully generalize to the evaluation data.
2. All models have similar accuracy. Since PointNet is faster and requires less memory for inference and training, we continue all other experiments only with PointNet.

Table 1: Rotational and translational errors on sequence 8 of the KITTI odometry dataset

Model	Rotation error (in °)			Translational error (in m)		
	25%	Median	75%	25%	Median	75%
PointNet	0.1947	0.4008	**0.7159**	0.0791	0.1606	0.2733
DGCNN	**0.1580**	**0.3718**	0.7449	0.0775	0.1557	0.2684
FlowNet3D	0.1840	0.4033	0.7781	**0.0758**	**0.1526**	**0.2652**

Additionally, we evaluate the performance when using a submap from the previous 5 frames and when refining the poses with ICP. Since both DGCNN and FlowNet3D require a lot of memory to run, we can not train and evaluate them on this task. The result can be seen in Table 2. Both methods bring a slight increase in accuracy.

Table 2: Rotational and translational errors on sequence 8 of the KITTI odometry dataset when trained with submaps

Model	Rotation error (in °)			Translational error (in m)		
	25%	Median	75%	25%	Median	75%
PointNet with submaps	0.1529	0.3394	0.6513	0.0694	0.1445	**0.2563**
PointNet with submaps & ICP	**0.1284**	**0.3130**	**0.6365**	**0.0652**	**0.1263**	0.2577

Using the relative poses from all consecutive frames, we can recover the trajectory that is predicted by our deep odometry and compare it to the ground truth trajectory. It should be noted that this trajectory is subject to drift - the accumulation of errors from the relative poses. This visualization can be seen in Figure 18.

5.2.1 Evaluation on Astyx data

Finally we train PointNet on astyx sequence B. We compare the simple point cloud registration and the registration of the submap of the previous 5 frames. As radar point clouds also have two additional dimensions of information for each point - its relative radial velocity and magnitude - we also compare the performance with this information included (giving 5 features per point) versus not included (giving 3 features per point).

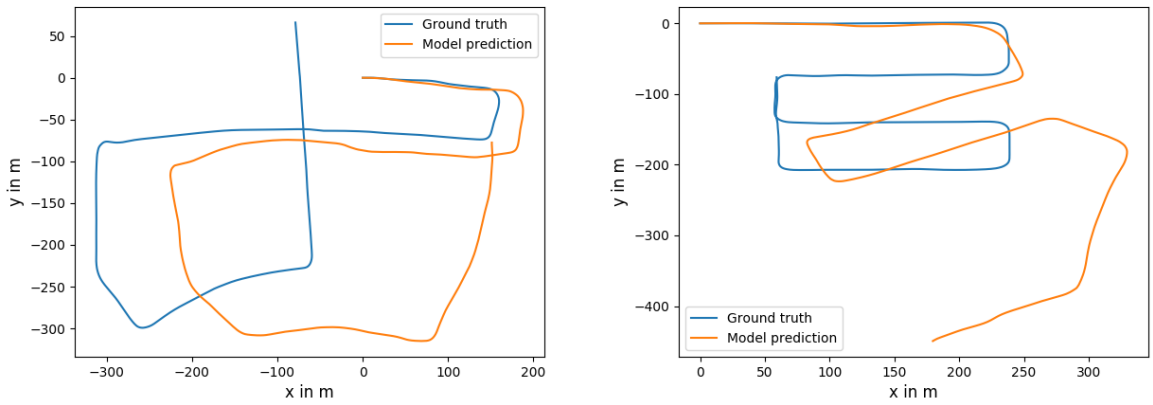


Figure 18: Trajectory predicted by the deep odometry compared to the ground truth trajectory for sequences of the KITTI dataset. While the overall poses are similar, the accumulation of errors leads to increasing absolute error over time.

Table 3: Rotational and translational errors on Astyx sequence B. p is the number of previous frames in the submap, f the number of features per point.

Model	p	f	Rotation error (in °)			Translational error (in m)		
			25%	Median	75%	25%	Median	75%
PointNet	0	3	0.1295	0.2612	0.4539	0.0501	0.1049	0.1815
PointNet	0	5	**0.1122**	**0.2296**	**0.3964**	**0.0379**	**0.0759**	**0.1332**
PointNet	5	3	0.1428	0.3083	0.5480	0.0617	0.1255	0.2109
PointNet	5	5	0.1377	0.3008	0.5411	0.0504	0.1009	0.1697

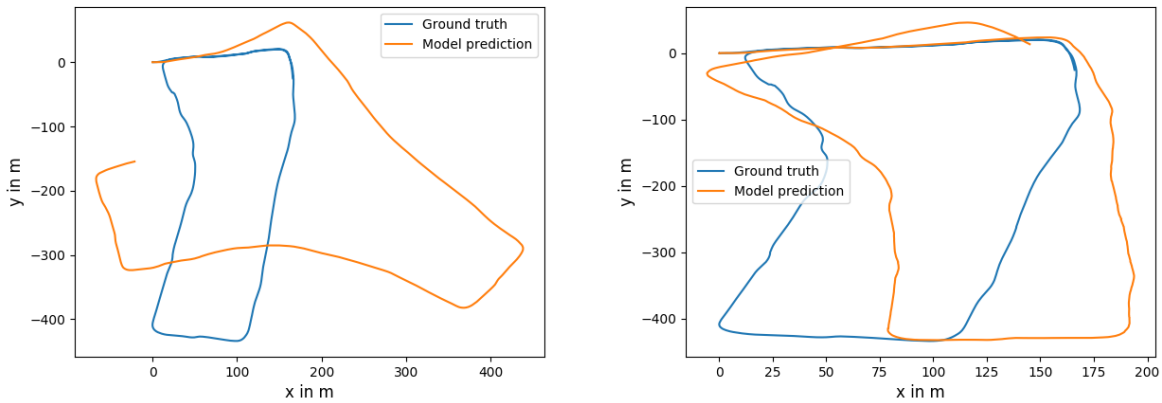
The results can be seen in Table 3, and visualizations in Figure 19. The matching with a submap of the previous 5 frames interestingly brings no advantage. We suspect, that this is because the features of the points - especially the radial velocity - can not be transformed to a sensible common frame in the submap.
We also refine the pose estimates with a few iterations of ICP. This increases accuracy, as can be seen in Table 4 and Figure 19.

Table 4: Rotational and translational errors on the astyx sequence B when refining the pose estimate with ICP

Model	p	f	Rotation error (in °)			Translational error (in m)		
			25%	Median	75%	25%	Median	75%
PointNet	0	5	0.0443	0.0919	0.1716	0.0208	0.0332	0.0493

The relative pose error is a lot lower than for the KITTI dataset. We suspect this has two reasons:

- The relative radial velocities in the radar data encode the velocity of the vehicle. This can be used by the network to improve the accuracy.



(a) Result without ICP refinement

(b) Result with ICP refinement

Figure 19: Trajectory predicted by the deep odometry compared to the ground truth trajectory for Astyx sequence B.

- The points in the radar point cloud are not randomly sampled like for the KITTI dataset, but are selected as interesting features by the radar sensor. These features are likely to persist between frame and can be easier matched than randomly selected points.

We also noticed that using two separate feature networks, instead of sharing weights for both point clouds (compare with Figure 14) improves the registration result as well. Since we dont have separate testing data and could not confirm this on the KITTI data, we suspect this simply allows the network to overfit more to the training data.

6 Conclusions

With our work we showed that optimizing the accuracy of $6DoF$ relative poses can be achieved by applying classical approaches on $3D$ radar point clouds. Furthermore, segmentation of static/dynamic objects was explored and plays a key factor in the success of other classical algorithms. For classification of static targets we used $RANSAC$ [32] with the relative radial velocity V_r and the direction of arrival as input features.

We have also shown that deep learning based approaches can perform point cloud registration even when the point clouds are very sparse and no further information from a GPS or IMU is available. The additional information in radar point clouds (radial velocity and magnitude) can be used to improve the registration accuracy. For deep learning based approaches we only focused on the front end part of SLAM. Other works have already shown that deep learning can be used to achieve better pose estimates by viewing SLAM as a sequence to sequence learning problem. This could be incorporated into our work but would likely require more data for training and evaluation.

This project did also not consider combining different sensors to achieve better SLAM results. In visual SLAM, preintegrated factors from an Inertial Measurement Unit have been shown to improve the overall system accuracy [45] by reducing drift. Using Cameras could possibly allow to segment even sparse point clouds, resulting in in higher stability in environments with many moving objects - like traffic in cities.
Unfortunately, we could not apply a comparison of our result to a multi-sensor setup due to lack of camera information in datasets.

References

[1] Holger Caesar et al. "nuScenes: A multimodal dataset for autonomous driving". In: *arXiv preprint arXiv:1903.11027* (2019).

[2] M. Meyer and G. Kuschk. "Automotive Radar Dataset for Deep Learning Based 3D Object Detection". In: *2019 16th European Radar Conference (EuRAD)*. 2019, pp. 129–132.

[3] C. Cadena et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[4] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[5] Martin Friedrich Holder, Sven Hellwig, and Hermann Winner. "Real-Time Pose Graph SLAM based on Radar". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. See https://tuprints.ulb.tu-darmstadt.de/8756/ for video attachment. 2019, pp. 1145–1151. URL: http://tuprints.ulb.tu-darmstadt.de/9285/.

[6] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256.

[7] Ziyang Hong, Yvan Petillot, and Sen Wang. *RadarSLAM: Radar based Large-Scale SLAM in All Weathers*. 2020. arXiv: 2005.02198 [cs.RO].

[8] R. Rouveure, M. O. Monod, and P. Faure. "High resolution mapping of the environment with a ground-based radar imager". In: *2009 International Radar Conference "Surveillance for a Safer World" (RADAR 2009)*. 2009, pp. 1–6.

[9] Tuo Feng and Dongbing Gu. "SGANVO: Unsupervised Deep Visual Odometry and Depth Estimation With Stacked Generative Adversarial Networks". In: *IEEE Robotics and Automation Letters* PP (June 2019), pp. 1–1. DOI: 10.1109/LRA.2019.2925555.

[10] S. Wang et al. "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2043–2050.

[11] Ronald Clark et al. "VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem". In: *AAAI*. 2017.

[12] Christopher Choy, Jaesik Park, and Vladlen Koltun. "Fully Convolutional Geometric Features". In: *ICCV*. 2019.

[13] Andy Zeng et al. "3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions". In: *CVPR*. 2017.

[14] Xuyang Bai et al. "D3Feat: Joint Learning of Dense Detection and Description of 3D Local Features". In: *ArXiv* abs/2003.03164 (2020).

[15] Yasuhiro Aoki et al. “PointNetLK: Robust and Efficient Point Cloud Registration Using PointNet”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 7156–7165.

[16] Simon Baker and Iain Matthews. “Lucas-Kanade 20 Years On: A Unifying Framework”. In: *International Journal of Computer Vision* 56.3 (2004), pp. 221 –255.

[17] R. Q. Charles et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 77–85.

[18] Vinit Sarode et al. *One Framework to Register Them All: PointNet Encoding for Point Cloud Alignment*. 2019. arXiv: 1912.05766 [cs.CV].

[19] Yue Wang and M. Justin Solomon. “Deep Closest Point: Learning Representations for Point Cloud Registration”. In: *International Conference on Computer Vision* (2019), pp. 3523–3532.

[20] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPSâFIXME^{TM}17. Long Beach, California, USA: Curran Associates Inc., 2017, 6000âFIXME“6010. ISBN: 9781510860964.

[21] Chris Xiaoxuan Lu et al. *milliEgo: mmWave Aided Egomotion Estimation with Deep Sensor Fusion*. 2020. arXiv: 2006.02266 [cs.RO].

[22] Younggun Cho, Giseop Kim, and Ayoung Kim. “DeepLO: Geometry-Aware Deep LiDAR Odometry”. In: *CoRR* abs/1902.10562 (2019). arXiv: 1902.10562. URL: http://arxiv.org/abs/1902.10562.

[23] Zhichao Li and Naiyan Wang. “DMLO: Deep Matching LiDAR Odometry”. In: *ArXiv* abs/2004.03796 (2020).

[24] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. “FlowNet3D: Learning Scene Flow in 3D Point Clouds”. In: *CVPR* (2019).

[25] Aseem Behl et al. “PointFlowNet: Learning Representations for Rigid Motion Estimation from Point Clouds”. In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[26] G. Kim, B. Park, and A. Kim. “1-Day Learning, 1-Year Localization: Long-Term LiDAR Localization Using Scan Context Image”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1948–1955.

[27] Giseop Kim and Ayoung Kim. “Scan Context: Egocentric Spatial Descriptor for Place Recognition within 3D Point Cloud Map”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Madrid, 2018.

[28] Emilio Parisotto et al. “Global Pose Estimation with an Attention-Based Recurrent Network”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2018), pp. 350–35009.

[29] Jens Behley and Cyrill Stachniss. “Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments”. In: *Proc. of Robotics: Science and Systems (RSS)*. 2018.

[30] Ji Zhang and Sanjiv Singh. “LOAM: Lidar Odometry and Mapping in Real-time”. In: July 2014. DOI: 10.15607/RSS.2014.X.007.

[31] Dominik Kellner et al. “Instantaneous ego-motion estimation using Doppler radar”. In: Oct. 2013, pp. 869–874. ISBN: 978-1-4799-2914-6. DOI: 10.1109/ITSC.2013.6728341.

[32] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), 381âFIXME“395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[33] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[34] P. Torr and A. Zisserman. “Robust computation and parametrization of multiple view relations”. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 727–732.

[35] O. Chum and J. Matas. “Matching with PROSAC - progressive sample consensus”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 220–226 vol. 1.

[36] S. Druon, M. J. Aldon, and A. Crosnier. “Color Constrained ICP for Registration of Large Unstructured 3D Color Data Sets”. In: *2006 IEEE International Conference on Information Acquisition*. 2006, pp. 249–255. DOI: 10.1109/ICIA.2006.306004.

[37] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: *arXiv:1801.09847* (2018).

[38] Marian Himstedt et al. “Large scale place recognition in 2D LIDAR scans using Geometrical Landmark Relations”. In: Sept. 2014, pp. 5030–5035. DOI: 10.1109/IROS.2014.6943277.

[39] M. Himstedt and E. Maehle. “Geometry matters: Place recognition in 2D range scans using Geometrical Surface Relations”. In: *2015 European Conference on Mobile Robots (ECMR)*. 2015, pp. 1–6. DOI: 10.1109/ECMR.2015.7324185.

[40] Pablo Fernandez Alcantarilla. “Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces”. In: Sept. 2013. DOI: 10.5244/C.27.13.

[41] Yue Wang et al. “Dynamic Graph CNN for Learning on Point Clouds”. In: *ACM Trans. Graph.* 38 (2019), 146:1–146:12.

[42] Abien Fred Agarap. “Deep Learning using Rectified Linear Units (ReLU)”. In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375.

[43] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICMLâFIXME^{TM}15. Lille, France: JMLR.org, 2015, 448âFIXME“456.

[44] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[45] Christian Forster et al. “On-Manifold Preintegration Theory for Fast and Accurate Visual-Inertial Navigation”. In: *CoRR* abs/1512.02363 (2015). arXiv: 1512.02363. URL: http://arxiv.org/abs/1512.02363.