Learning to See the Wood for the Trees: Deep Laser Localization in Urban and Natural Environments on a CPU

Georgi Tinchev, Adrian Penate-Sanchez, and Maurice Fallon

***Abstract*—Localization in challenging, natural environments such as forests or woodlands is an important capability for many applications from guiding a robot navigating along a forest trail to monitoring vegetation growth with handheld sensors. In this work we explore laser-based localization in both urban and natural environments, which is suitable for online applications. We propose a deep learning approach capable of learning meaningful descriptors directly from 3D point clouds by comparing triplets (anchor, positive and negative examples). The approach learns a feature space representation for a set of segmented point clouds that are matched between a current and previous observations. Our learning method is tailored towards loop closure detection resulting in a small model which can be deployed using only a CPU. The proposed learning method would allow the full pipeline to run on robots with limited computational payload such as drones, quadrupeds or UGVs.**

***Index Terms*—Localization; Deep Learning in Robotics and Automation; Visual Learning; SLAM; Field Robots**

I. Introduction

LOCALIZATION is a fundamental task in robotic perception, a robot needs to know where it is to navigate in the environment and to make decisions. It has been heavily explored with computer vision, demonstrating impressive results at large scales [1], [2], [3]. These types of approaches typically assume a certain inherent structure in the scene, image features are dependant on repeatable camera viewpoint [3], and methods are often tested in urban environments which guide the robot along the route in question [4]. While the aforementioned visual approaches have many promising characteristics, here we explore LIDAR due to its robustness to varying lighting conditions, changes in viewpoint, and trackline offsets. It is a precise and long range sensing modality.

The SegMatch system [5] proposed a modular segment-based approach for LIDAR teach-and-repeat, which could localize within a prior map while also retaining a degree of semantic meaning, as the segments matched corresponded to large physical objects such as cars and parts of a building.

Manuscript received: September, 10, 2018; Revised December, 5, 2018; Accepted January, 8, 2019.

This paper was recommended for publication by Editor Cyrill Stachniss upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by EPSRC RAIN and ORCA Robotics Hubs (EP/R026084/1 and EP/R026173/1 respectively). M. Fallon is supported by a Royal Society University Research Fellowship.

The authors are with the Dynamic Systems Group, Oxford Robotics Institute, University of Oxford, United Kingdom. {gtinchev,adrian,mfallon}@robots.ox.ac.uk

Digital Object Identifier (DOI): see top of this page.

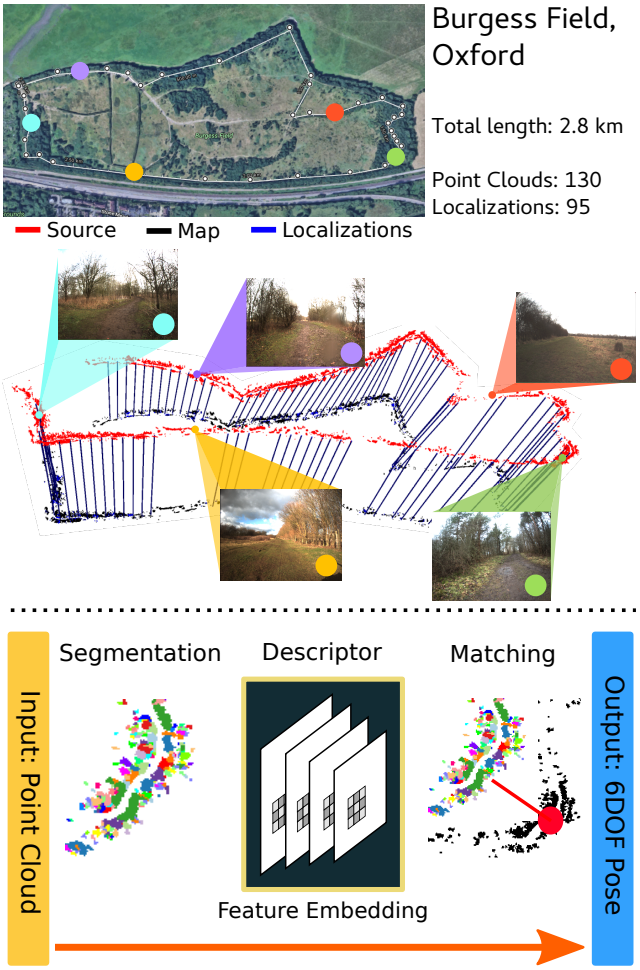

Fig. 1: **Top:** Depiction of the obtained results on large, unstructured environments. The proposed approach regularly proposes correct loop closures in these challenging natural scenes, using a deep learning architecture running only on a CPU. **Bottom:** Overview of the proposed methodology. Segments are extracted from the input point cloud (coloured section) and the map cloud (in black on top). Our neural network features are extracted for all segments. The features from the map are stored in a database. The features from the query cloud are matched against the database. The final pose is then estimated with PROSAC using both the probability of a segment matches(from the network) and the position of the segments (from the map).

The approach was further improved with learned feature descriptors [6] which achieved greater accuracy. The approach

uses a GPU to achieve real-time performance.

We are motivated to localize in natural scenarios like forests. In these environments many assumptions about the appearance or the geometry of landmarks are no longer valid - vegetation grows between seasons and there is no planar structure. Our previous work showed promising results in both structured (urban) and unstructured (natural) environments [7]. While reliable localization was achieved, we relied on a hand-crafted set of features which limited the performance.

The main contribution of this work is a novel description method for segment-based LIDAR localization. We aim to improve upon previous works by learning segment representations in a way that inherently handles the variability of the given environment. The proposed approach learns a descriptor space which efficiently represents the similarities between partial observations of the same segment which makes it robust to incomplete data. We use a neural network to learn this high dimensional feature space. The proposed approach utilizes the convolution operation proposed in [8] to learn an embedding space for both urban and natural scenarios directly from the raw point cloud data. The neural network has the following characteristics:

- Unordered point clouds as input: does not require a specific ordering of the point clouds in a segment. This makes our approach flexible by avoiding the computational cost of creating specific structures for the point cloud and aligning the inputs to a grid or voxelization.
- Feature space is capable of being generalized: experimental validation has proven that our proposed deep learning solution creates a feature space that can generalize, without the need to be retrained to a new sensor or a different environment.
- The network can estimate the quality of a match: a probability is computed and can be used when carrying out probabilistic geometric validation such as PROSAC [9], making our approach more efficient.

In the results section we demonstrate that our proposed method significantly outperforms other hand-engineered approaches, while also improving the computational speed in comparison to other deep learning approaches. In particular we achieve localization performance similar to SegMap [6], but do not require a GPU at runtime. Instead the method can be deployed online on the CPU of a mobile robot. This performance improvement comes in large part because our network is more specifically tailored to the task of localization. As presented in [6], the SegMap learning approach can be used to compress and reconstruct point clouds as well as extract and use semantic meaning from the segments to aid localization. We demonstrate performance in urban environments as well as natural environments so as to demonstrate our approach's generality and robustness.

II. Related Work

Robot localization has been heavily explored using different sensors such as LIDAR [10], vision [11], GPS [12] or radar [13] with reliable approaches often combining different sensing modalities [10], [12], [13]. Our proposed work focuses on global LIDAR localization particularly applied to unstructured environments rather than incremental localization or odometry. Here we will briefly overview some methods performing LIDAR localization.

For self-driving vehicles many approaches have taken advantage of LIDAR reflectivity to achieve precise localization. These methods are commonly used in commercial approaches, for example, the approach of [14] uses a prior map of reflectivity and exploits road marks to reliably localize a vehicle in an urban environment. The authors formulate the world as a mixture of Gaussians (GMM) over 2D grid cells. The GMM represents the heights of points in each cell and the reflectance in a vigorous way that allows the approach to be robust to weather alteration and road degradation.

When localization is carried out concurrently with mapping it is often treated as a research field of its own (Simultaneous Localization and Mapping, SLAM) where the key problem of loop-closure detection is often equivalent to global localization. In this context the robot's odometry provides a rough estimate of the position of the robot within the world which can be leveraged to avoid searching for loop closures over the entire map, thus keeping the computational cost low [15]. When the assumption of a local neighbourhood is lost most of these approaches require ICP alignments or similar costly alternatives [16].

Recent approaches aim to localize on a higher level than on explicit point-to-point basis. The concept of segment localization was presented in [17] but more specifically our work is directly motivated by the works of Dubé *et al* [5], [6]. We will describe the framework of segment-based localization in Section III. Features created from LIDAR data are less informative or unique than visual features, thus approaches often prune the candidate matches to reduce the percentage of outliers before performing a robust geometric validation [18]. One of the key advantages of using segments instead of keypoints is that more semantically meaningful entities are extracted, increasing the repeatability of the descriptors. This was demonstrated in [5] and was applied to urban environments. In [6] segment localization evolved into a segment-based SLAM that introduced a new segment feature that also encoded semantics and volumetric shape. This interesting line of work learns features suitable for localization of voxelized segments and the semantics of the map simultaneously. The approach was evaluated in realistic urban setting and in derelict buildings. Elbaz *et al* [19] used segments produced from a Random Sphere Cover Set - overlapping point cloud spheres where each point of the original point cloud could be part of more than one sphere. These spheres were then projected to 2D depth images and processed by a deep auto-encoder. In our previous work [7] we proposed a larger hybrid feature descriptor which improved detection performance and achieved localizations in unstructured environments such as forests, while also being capable of handling the urban scenarios.

The main drawback of classical approaches that use LIDAR data is that they rely on handcrafted features which do not perform as well as learned features [20]. When trying to apply learning techniques to 3D data there are two prominent approaches - representing the data as a voxelized grid, or

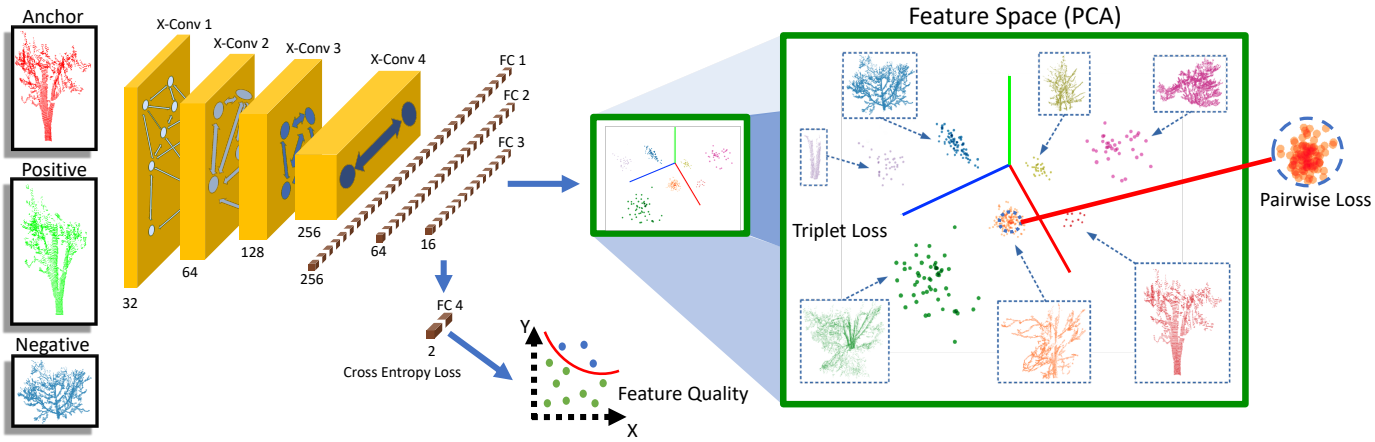

Fig. 2: Network architecture used for learning the feature embeddings (left) and a visualization of the embedding space after training (right). The network takes as input raw point data, it uses four x-convolutional layers ($\mathcal{X}$-conv) and three fully connected (FC) layers to estimate the feature space. A feature quality branch is also created by appending a fully connected layer to the feature embedding. The networks learns to cluster similar objects (denoted in the same colour) close together while separating dissimilar objects. The final feature descriptor is trained using a combination of triplet and pairwise losses.

by using the more common representation of point clouds. Initially these approaches tended to use voxelized inputs as the tensor operators generalize directly to this representation and achieve promising results as shown in [21]. The main drawback of this approach is the amount of computation required to model the entirety of the environment. In [21] the authors used a 32x32x32 voxelized grid, resulting in a dimensionality of 32768, with 3D convolutions applied in strides. Because the input resolution is low (due to the computational complexity), these approaches are not capable of handling detailed shapes such as the legs of a chair, for example, as shown in [22]. The issue of handling point clouds was addressed in [23], [24] by learning a symmetry function approximated by a multi-layer perceptron (MLP). The authors created a neural network layer that was invariant to point order. In contrast to the aforementioned volumetric approach, a raw processed point cloud containing 1024 points resulted in a dimensionality of only 3072. In addition, by using a MLP as the basis of their layer and by aggregation of spatial data, Li *et al* [8] managed to create a more descriptive layer to handle unordered point clouds.

In [25] a query point cloud is downsampled and matched against a collection of previously collected submaps globally. All clouds are described on a GPU using a combination of networks trained with a lazy quadruplet loss to produce a 256-dimensional feature vector. In contrast, we firstly segment the query and map point clouds to produce a small set of segments, local keypoints. The segments are then described in real time on a CPU. Thus, our approach uses a combination of local features for each segment and matches those in the map. In this way we are robust to disturbances from occlusions and changing environments.

III. Methodology

In this section we present Efficient Segment Matching (**ESM**). This work retains the approach to pose estimation which was initially proposed by [5]. Our contributions in this paper are 1) learning a novel segment descriptor, 2) together with a learned per descriptor measure of performance which enables the pruning of the features before matching and 3) the use of a probabilistic robust pose estimation [9] to improve matching performance.

The problem is formulated as follows: given an input point cloud $\mathcal{P} = \{p_i \in \mathbf{R}^3\}$ and a prior map $\mathcal{M} = \{p_j \in \mathbf{R}^3\}$, we wish to estimate the pose of $\mathcal{P}$ in the map $\mathcal{M}$. We split the task into four modules - Segmentation, Description, Matching and Pose Estimation. Our methodology is illustrated in Fig. 1, bottom. An input point cloud is first segmented using Euclidean segmentation to produce a number of 3D point cloud objects based on the distance of points between them.

Next the segments are passed to a neural network to extract a descriptor of each segment and a measure of quality of that descriptor. For each segment in the live point cloud the N-closest neighboring segments in the map are found based on the Euclidean distance of the descriptors. These matches are ordered based on the extracted feature quality and passed to PROSAC [9] to produce a set of possible localizations. In this work the ordering is done by computing the joint probability of both features (the feature from the live point cloud and the one from the map). The order of the matches is important because the geometric consistency and PROSAC consider first the matches with higher likelihood. Each module of this pipeline is described in depth in previous works [7], [5]. In the following section we detail the architecture and the learning approach.

A. Architecture

Our network architecture is constructed based on the novel convolutional layers presented in [8]. Fig. 2 illustrates our model. The network learns directly from point cloud data, it takes as input a batch of raw point cloud segments. Each segment is uniformly downsampled to 256 points, zero-centered with normalized variance.

The network used for our descriptor learning approach consists of four $\mathcal{X}$-conv operators [8] and three fully connected layers; dropout [26] of 0.5 is applied at the second fully connected layer. The outputs ot the last fully connected layer are used as the descriptor. The $\mathcal{X}$-conv operator convolves local regions, similar to convolutions in images by a CNN. For each point, its closest N neighbors are projected to a local coordinate frame and lifted to a higher dimensional space with a multi-layer perceptron (MLP). The $\mathcal{X}$-conv operator learns a convolution based on the MLPs of the neighbouring points. In order for the top $\mathcal{X}$-conv operators to see a larger portion of the point cloud a dilation rate (D) is applied [27]. In this way the receptive field of the top layers is increased without an increase of the neighbouring point size (N) or the kernel size.

We have selected a specific configuration of the dilation and neighbourhood in each of our layers that better represents the problem we are trying to solve. The first layer uses a neighbourhood of 8 and a dilation of 1 per $\mathcal{X}$-conv operator, the second a neighbourhood of 12 and dilation 3, the third a neighbourhood of 16 and a dilation of 3 and the fourth a neighbourhood of 16 and a dilation of 4. The reason for this is that the first layer will look at a few points (neighbourhood 8) that are immediately close (dilation 1). The second will look into more points (neighbourhood 12) further out (dilation 3). This way the network creates a representation that aggregates the information slowly creating a hierarchical representation of the whole segment. This is done for each point so as to create a feature of the entire segment from the relative viewpoint of each point. The final descriptor is computed as an average of the features for all the points in a single point cloud. In this way, the feature fuses data from different points creating an expressive yet simple representation. In our experiments we managed to achieve good performance by using an embedding size of 16 dimensions. The dimensionality of a descriptor has a strong impact on the performance of matching. This keeps the computational cost low even if the segmentation algorithm produces many more segments.

The proposed network architecture also includes a classification branch that estimates a measure of the quality of each descriptor. To be able to train this branch we need to train the feature network first. When training the classification branch the descriptor layers are not modified, the model optimizes a single fully connected layer of size two. This represents a logistic regression that classifies whether a feature is good for matching or not. The quality of the feature is determined during training if a successful match is found within the first K neighbors in the dictionary. The classification branch is trained until its accuracy converges at $\approx 70\%$ with $K = 1$ neighbor. At test time the confidence score from the classification branch is used to compute a joint probability of all matched segments. Since two matched segments are gathered during independent observations, we simply multiply the probabilities. During the last stage of the localization pipeline the candidates are sorted using the joint probability distribution and the pose is estimated using PROSAC [9] with applied geometric consistency constraints [7].

B. Learning the Segment Feature Space

To learn the feature space in the proposed architecture a variation of the triplet loss [28] is used. The triplet loss clusters together samples in the feature space that have been labelled as similar and tries to separate samples that have been labelled as different. By applying the same label to different examples of a segment we introduce invariance to many factors such as noise in the measurements or incomplete segments. The triplet loss defines a triplet as a combination of a sample (anchor) with other two samples, one with the same label and the other with a different label (Fig. 2, left). A pairwise term is defined as a combination of a sample (anchor) with a different sample that has the same label. To train our descriptor we use a variation of the triplet loss similar to the one defined in [29] - in our case we use a squared L_2 norm as shown in Eq. (1) while they use a regular L_2 norm. To train the model we use a batch size of 256 point clouds from which we extract a large set of triplet and pairwise terms.

$$\mathcal{L} = \mathcal{L}_{triplets} + \alpha \mathcal{L}_{pairs} + \lambda \|w\|_2^2. \tag{1}$$

During training we identify $\approx$ 300 times more triplet than pairwise terms. We balance this effect using the α parameter. In Eq. (1) w denotes the parameters of our model, weighted by $\lambda = 10^{-6}$. The proposed architecture contains only 300 K trainable parameters. We use an initial learning rate of $\eta = 10^{-3}$, decaying to minimum $\eta = 10^{-6}$ with ADAM [30] as optimizer. We describe the two losses in detail below.

The classification network is trained using the same data used for the embeddings layer. The features are precomputed and used as a dictionary. To train the classification branch, for each sample we find the K closest matches in the dictionary. A binary label is assigned for each sample: if a match is found in the dictionary it is labelled as 1, otherwise 0. The network optimizes a softmax cross-entropy loss for the classification branch, given the aforementioned labels.

C. Triplet Loss

As commented before, we use a variation of the definition of triplet loss similar to the one in [29]. This loss modified the original triplet loss to solve the vanishing gradient problem. We modify this loss by using squared L_2 norm, instead of the standard L_2 norm, as it created a better clustering of the feature space. The final triplet loss is defined as the sum of the following cost function over all the triplets:

$$\mathcal{L}_{triplets} = \sum_{(s_i, s_j, s_k) \in \mathcal{T}} c(s_i, s_j, s_k) \tag{2}$$

$$c(s_i, s_j, s_k) = \max\left(0, 1 - \frac{\|f_w(x_i) - f_w(x_k)\|_2^2}{\|f_w(x_i) - f_w(x_j)\|_2^2 + m}\right) \tag{3}$$

$\mathcal{T}$ denotes the set of all possible triplets. s_i and s_j are segments with the same label, while s_i and s_k are dissimilar. $f_w(x_i)$ is the output of the last descriptor layer for an input point cloud x_i and m is a margin regularizer. The latter denotes the minimum ratio for the Euclidean distances between dissimilar pairs of point clouds and similar ones. We use $m = 0.01$ in our experiments. In this manner the triplet loss will cluster

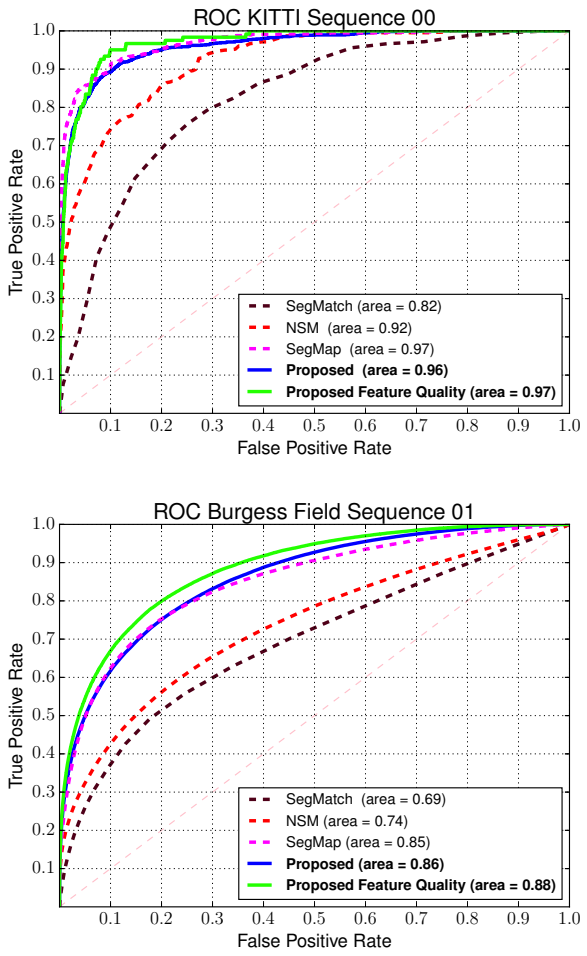

Fig. 3: Receiver Operating Characteristic (ROC) curves for the proposed and baseline approaches [5], [6], [7] on urban (top) and natural (bottom) datasets. The proposed approach performs comparably to the current state of the art on both environments. Learning approaches outperform hand-engineered ones in both urban and natural scenarios.

similar objects together, while separating dissimilar ones to be farther apart in the feature space. The proposed architecture will provide a descriptor of the whole segment per point, centered at each of those points. Our feature is computed as the average of those descriptors. Our losses are computed using the averaged features. Fig. 2 (right) presents a PCA visualization of the feature space after the network has been trained with the corresponding segments. Different instances of the same tree are clustered together while other trees form their own clusters.

D. Pairwise Loss

The pairwise loss minimizes the distance between samples of the same class (Fig. 2, orange). As proposed by [29], we optimise the following:

$$\mathcal{L}_{pairs} = \sum_{(s_i,s_j)\in\mathcal{P}} \|f_w(x_i) - f_w(x_j)\|_2^2 \qquad (4)$$

where $\mathcal{P}$ denotes the set of all pairs. The loss aids the training process by generating very tight clusters, which in turn improve the KNN retrieval during the matching phase.

IV. Experimental Evaluation

The main focus of this work is to utilize learned features in point clouds to localize with respect to a prior map. The experiments are designed to support our key claims:

- A novel learned feature descriptor that generalizes across a variety of natural and urban datasets.
- Real time operation on a CPU as a result of a compact network architecture.
- The approach produces a measure of quality for each feature which can be used during the pose estimation step to decrease computation.

We provide comparisons against a popular method for segment-based localization, SegMatch[1], as proposed in [5], a data-driven incremental approach SegMap[1] [6], and our previous work NSM [7]. As a metric to evaluate the approaches we have chosen to compare the True Positive Rate (TPR) against the False Positive Rate (FPR) for each classifier, the number of localizations on each dataset, and the computation time for each pipeline. Supplementary material about our model's hyper-parameters and a video accompany the paper at http://ori.ox.ac.uk/esm-localization.

A. Datasets

We perform evaluations using three datasets of our own which were collected in natural environments as well as on a publicly available dataset. First, we compared our novel descriptor against the baseline approaches using the KITTI [4] dataset (Karlsruhe, Germany), taken by a 3D Velodyne-HDL64 sensor in an urban scenario. We use Sequence 06 and 05 to train all descriptors, and Sequence 00 to test them. We also compare the localization performance on Sequence 00. Second, we compared localization performance in a park and forest datasets - George's Square (Edinburgh, Scotland) and Cornbury Park (Oxfordshire, UK). The former was captured by a Clearpath Husky UGV equipped with a SICK LMS511 LIDAR, while the latter was captured by a vehicle equipped with a Velodyne HDL32E. The datasets are described in detail in [7].

Finally, we used a dataset collected in a natural environment, located at Burgess Field (Oxfordshire, UK). The dataset was captured by a Clearpath Husky UGV equipped with a Bumblebee3 forward facing camera, a Velodyne VLP-16 LIDAR and a Push-broom LIDAR LMS151. The vehicle traversed the same loop of ≈ 2.8 km twice a month during a span of six months. We used the Velodyne data of Sequence 02 and Sequence 01 for training and testing the classifiers. These two sequences were collected two weeks apart each other in February when the foliage was shed. We built a prior map from the Push-broom LIDAR of Sequence 02 using [2]. The source swathes were built using the VO method [2] for every 22 meters the vehicle traversed in Sequence 01.

[1]We use the open-source implementations of SegMatch/SegMap.

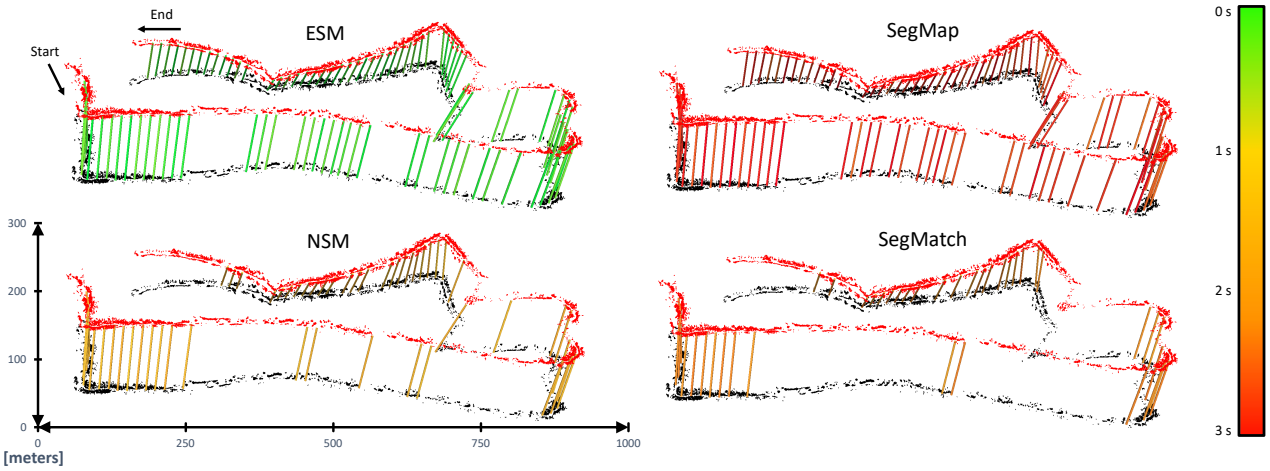

Fig. 4: Illustration of the localization performance of our approach (top left) compared to the baselines on the Burgess Field dataset. The current point clouds (red) are registered to a previous map (black). Each vertical line corresponds to a recognised place, while its color corresponds to the time taken to perform the localization.

Dataset Characteristics					Number of Localizations			
Name	**Type**	**Length (meters)**	**Sensor**	**Num. Clouds**	**SegMatch**	**NSM**	**SegMap**	**ESM**
KITTI	Urban	1300	Accum. Velodyne-HDL64E	134	41	53	62	**63**
George Square	City Park	500	Push-broom	30	8	9	8	**11**
Cornbury Park	Forest	700	Velodyne-HDL32E	1125	N/A	29	231	**248**
Burgess Field	Forest	2800	Push-broom	130	41	54	**95**	94

TABLE I: Number of localizations across one urban and three natural datasets. The proposed approach performs comparably to the state of the art. These experiments also show the ability of the learning approaches to handle data from different types of LIDAR sensors.

To label the data we combined four consecutive Velodyne scans using the motion estimator from [2] and extracted segments using distance-based segmentation algorithm. All segments to be within 0.5 m in consecutive point clouds are considered the same object and labelled as such. All segments further than 4.0 m apart are considered a non-match. In this manner we extracted a total of 82871 segments across 38714 classes for training and 45409 segments in 20583 classes for testing. Training our model on Burgess Field takes 3 hours and 25 minutes from scratch. We have used this model to evaluate our approach on all natural datasets. For experiments carried out on urban datasets, we trained a different model on KITTI in 11 hours and 45 minutes. For all the experiments and methods we use Euclidean segmentation. We segmented the data into individual point clouds depending on the sampling density: 3D Velodyne data: 200–15000 points due to higher frequency, Push-broom type LIDAR: 200–50000 points due to higher density.

B. Classifier Performance

The first experiment evaluates the performance of our model, with and without the feature quality, on KITTI and Burgess Field datasets. Fig. 3 illustrates the Receiver Operator Characteristic (ROC) curves for each of the algorithms. The ROC curve for the feature quality network is created by removing the top *bad* matches prior to evaluation. A performance decrease is seen between the two datasets for all algorithms. We attribute this to the more challenging natural structure in the second environment. In brief, our learning approach generalizes better than engineered features (NSM, SegMatch) across datasets and performs comparably to the learned approach (SegMap). Pruning the matches based on the feature quality shows an improvement with respect to the basic features. For all models we have set the classifier thresholds at FPR=0.1 for urban and FPR=0.2 for forest environments. We did not retrain the models for each forested scenario or different sensor modality. This shows the proposed features can generalize between datasets and sensors.

C. Localization Performance

In the next set of experiments we aim to support our claim that the proposed algorithm performs comparably to the state of the art on both urban and natural environments when localizing, while requiring less computation. Tab. I shows the total number of loop closures detected in each of the datasets. For each algorithm we have optimized the parameters to retrieve the highest number of true localizations, while having zero false localizations. The learning approaches have not been retrained for each individual natural dataset but still manage to detect about two times more loop closures than the engineered methods. In brief, our algorithm has similar accuracy to [6] on all datasets. Fig. 4 presents qualitative results for our approach compared to the baselines on the Burgess Field dataset, while also highlighting the computational times for each approach. In this environment the vehicle regularly detected loop closures during more than 2 km of operation. Fig. 5 demonstrates the performance of our algorithm across all the datasets. The method did not produce any false localizations, while operating in real time on a CPU.

CPU Multi-core execution times								
Algorithm	**Desc. size**	**Segmentation**	**Preprocessing**	**Descriptor**	**Matching (K)**	**Prunning (RF)**	**Pose Est.**	**Total**
SegMatch	7 (647)	17	0	605	50 (200)	755	64	1491 ms
NSM	66	17	0	24	207 (200)	913	65	1226 ms
SegMap	64	17	15	5902	19 (25)	0	21	5974 ms
ESM	16	17	3	578	8 (25)	0	9	615 ms
CPU Single-core execution times								
Algorithm	**Desc. size**	**Segmentation**	**Preprocessing**	**Descriptor**	**Matching (K)**	**Prunning (RF)**	**Pose Est.**	**Total**
SegMatch	7 (647)	21	0	596	70 (200)	773	64	1524 ms
NSM	66	21	0	37	213 (200)	936	65	1272 ms
SegMap	64	21	26	25945	26 (25)	0	21	26039 ms
ESM	16	21	3	2126	12 (25)	0	11	2173 ms
GPU Computation comparison								
Algorithm	**Desc. size**	**Segmentation**	**Preprocessing**	**Descriptor**	**Matching**	**Pruning (RF)**	**Pose Est.**	**Total**
SegMap	64	17	15	15	19	0	15	81 ms
ESM	16	17	3	2	8	0	6	36 ms

TABLE II: Average computational times in milliseconds recorded per point cloud on the Burgess Field dataset.

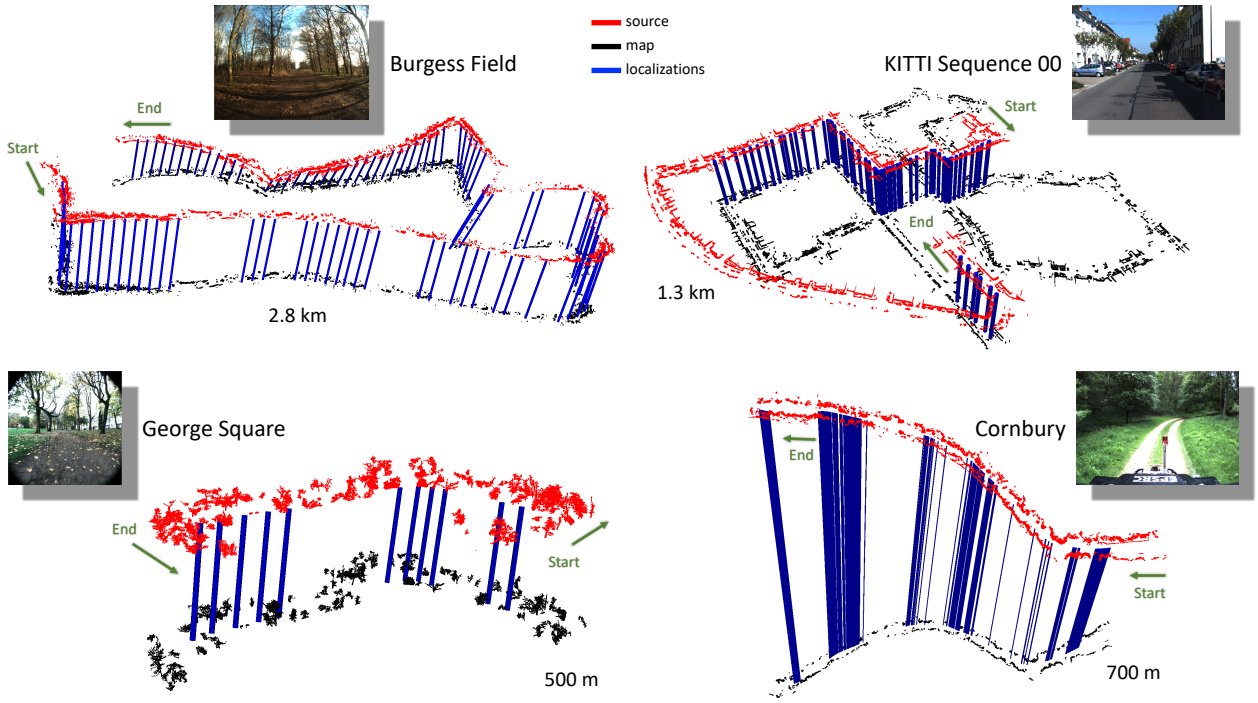

Fig. 5: Qualitative visualization of the performance of the proposed approach (ESM) on all datasets. The used datasets provide a variety of contexts to show the behaviour of ESM when coping with structured and unstructured locations.

D. Computation Efficiency

Finally, we evaluate the computational performance of the various approaches. We tested our approach on a mobile Intel Xeon E3-1535M CPU with the following configurations: (1) single-core of the processor, (2) multi-core on the same processor, (3) CPU + NVIDIA Titan Xp GPU. We have processed the push-broom point cloud data from Burgess Field and recorded the mean computational speeds. The target map consisted of 2783 segments — individual trees, bushes, and interleaved shrubs. Individual source clouds were created for every 22 m travelled with each source cloud containing an average of 46 segments. In order to provide a fair comparison, we have utilized the same Euclidean segmentation method across algorithms, to create the same segments. This is important for the description stage of each pipeline. The segmentation was carefully parametrized to increase localization performance in this challenging natural environment. To estimate the pose we have used the same geometric consistency algorithm for all baselines substituting RANSAC [31] with PROSAC [9] for the proposed method to incorporate the quality of the features. In this setting, using an incremental geometric consistency as in [6] is not possible, as the push-broom LIDAR provides only a single observation of a segment.

We have empirically evaluated the number of neighbors in feature space (K) on Burgess Field to retrieve the most positive localizations, while keeping zero false positives. For this experiment K influences the KNN retrieval and pruning speeds. $K = 25$ worked well for learning approaches as the features were very descriptive, while we kept $K = 200$ for NSM and SegMatch. Tab. II summarizes the average runtime performance, per point cloud, in milliseconds. The size of the descriptor dimension corresponds directly to the computation

time taken to describe a segment. The preprocessing and descriptor times scale linearly with the number of segments in a live point cloud (≈ 46 in Burgess Field). The time required for matching also depends on the embedding dimension. Point clouds tend to be of similar size, thus the computation time for the segmentation does not vary. The pose estimation depends on the number of segments and their closest neighbors (K) and the pose estimator approach. The total size of the SegMatch descriptor is 647 dimensions, only 7 of which are used during the matching stage. These 647 are compressed efficiently to the 45 processed by the Random Forest. This results in fast KNN retrieval and slower RF pruning. The total size of the NSM descriptor is 66, which are extended to 330 for the pruning stage, making it less efficient for KNN retrieval and pruning. The SegMap descriptor requires PCA alignment and voxelization of the segments as preprocessing steps, after which the forward pass of the model is executed in C++. Even though the implementation is efficient, the $9.3\,\mathrm{M}$ parameters of the network describe a single segment in $0.5\,\mathrm{s}$ on a CPU. Tab. II also summarizes the recorded GPU times for the learned approaches with $K = 25$ during matching.

We focused our analysis on CPU-only solutions due to the particular efficiency of our approach. Our network does not require expensive pre-processing of the segments, the model consists of 4 X-conv layers with an input of 256 points. This results in only $300\,\mathrm{K}$ parameters, which represents the number of operations needed to be performed. Compared to other learning methods our network has 30–65x less parameters, which allows the model to work in real time on a CPU. In addition, the embedding dimension is kept to just 16 dimensions which speeds up matching.

V. Conclusion

In this paper, we presented a novel descriptor for place recognition based on LIDAR segment matching in both urban and natural environments. Our method exploits an efficient deep learning architecture that operates directly on point cloud data without the need of extensive preprocessing. This allows us to successfully detect loop closures at $\approx 1\,\mathrm{Hz}$ while using only a CPU. We implemented and evaluated our approach on four different datasets containing natural forest and parkland as well as urban scenes and provided comparisons to other existing techniques. Our approach operates in real time on a CPU and achieves performance comparable to the state of the art, SegMap [6], which requires a GPU to run in real time. The experiments suggest that our approach can be applied to mobile robots with limited computational power. In future work we are interested in deploying the approach on an ANYMAL quadruped and UAVs both of which lack a GPU.

VI. Acknowledgements

We would like to thank Oliver Bartlett and our colleagues at the Oxford Robotics Institute for the datasets.

References

[1] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *TRO*, vol. 31, no. 5, pp. 1147–1163, 2015.

[2] W. Churchill and P. Newman, "Experience-based Navigation for Long-term Localisation," *IJRR*, 2013.

[3] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building Rome in a Day," *CACM*, vol. 54, no. 10, pp. 105–112, 2011.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *IJRR*, vol. 32, no. 11, pp. 1231–1237, 2013.

[5] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. C. Lerma, "SegMatch: Segment based place recognition in 3D point clouds," in *ICRA*, 2017, pp. 5266–5272.

[6] R. Dubé, A. Cramariuc, D. Dugas, J. Nieto, R. Siegwart, and C. Cadena, "SegMap: 3D Segment Mapping using Data-Driven Descriptors," in *RSS*, 2018.

[7] G. Tinchev, S. Nobili, and M. Fallon, "Seeing the Wood for the Trees: Reliable Localization in Urban and Natural Environments," in *IROS*, 2018.

[8] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution On X-Transformed Points," in *NIPS*, 2018, pp. 828–838.

[9] O. Chum and J. Matas, "Matching with PROSAC - Progressive Sample Consensus," in *CVPR*, 2005.

[10] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Monte Carlo Localization for Mobile Robots," in *ICRA*, 1999.

[11] P. T. Furgale and T. D. Barfoot, "Visual Teach and Repeat for Long-Range Rover Autonomy," *JFR*, vol. 27, no. 5, pp. 534–560, 2010.

[12] J. Levinson, M. Montemerlo, and S. Thrun, "Map-Based Precision Vehicle Localization in Urban Environments." in *RSS*, 2007.

[13] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *TRO*, vol. 17, pp. 229–241, 2001.

[14] R. W. Wolcott and R. M. Eustice, "Robust LIDAR localization using multiresolution Gaussian mixture maps for autonomous driving," *IJRR*, vol. 36, pp. 292–319, 2017.

[15] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," in *ICRA*, 2016.

[16] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets," *AURO*, vol. 34, no. 3, pp. 133–148, Apr. 2013.

[17] B. Douillard, A. Quadros, P. Morton, J. P. Underwood, M. D. Deuge, S. Hugosson, M. Hallstrm, and T. Bailey, "Scan segments matching for pairwise 3D alignment," in *ICRA*, 2012.

[18] C. Zach, A. Penate-Sanchez, and M.-T. Pham, "A dynamic programming approach for fast and robust object pose recognition from range images," in *CVPR*, 2015.

[19] G. Elbaz, T. Avraham, and A. Fischer, "3D Point Cloud Registration for Localization using a Deep Neural Network Auto-Encoder," in *CVPR*, 2017, pp. 2472–2481.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[21] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction," in *ECCV*, 2016.

[22] F. Haoqiang, H. Su, and L. J. Guibas, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image," in *CVPR*, 2017.

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *CVPR*, 2017.

[24] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *NIPS*, 2017.

[25] M. A. Uy and G. H. Lee, "PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition," in *CVPR*, 2018.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *JMLR*, vol. 15, pp. 1929–1958, 2014.

[27] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *ICLR*, 2016.

[28] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance Metric Learning for Large Margin Nearest Neighbor Classification," in *NIPS*, 2006.

[29] P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3D pose estimation," in *CVPR*, 2015, pp. 3109–3118.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

[31] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *CACM*, vol. 24, no. 6, pp. 381–395, 1981.