Tampere University

JUKKA PELTOMÄKI

LiDAR Place Recognition with Image Retrieval

Tampere University Dissertations 755

Tampere University Dissertations 755

JUKKA PELTOMÄKI

LiDAR Place Recognition with Image Retrieval

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the auditorium TB109
of the Tietotalo, Korkeakoulunkatu 1, Tampere,
on 24 March 2023, at 12 o'clock.

ACADEMIC DISSERTATION
Tampere University, Faculty of Information Technology and Communication Sciences
Finland

Responsible supervisor and Custos	Professor Joni Kämäräinen Tampere University Finland	
Supervisors	Dr. Heikki Huttunen Visy Oy Finland	Associate Professor Esa Rahtu Tampere University Finland
Pre-examiners	Professor Heikki Kälviäinen LUT University Finland	Dr. Bertil Grelsson Saab Dynamics Sweden
Opponents	Professor Tapio Seppänen Oulu University Finland	Professor Heikki Kälviäinen LUT University Finland

ISBN 978-952-03-2787-3 (print)
ISBN 978-952-03-2788-0 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-2788-0

HIILINEUTRAALI PAINOTUOTE
ClimateCalc CC-000025/FI
PunaMusta Printing

Carbon dioxide emissions from printing Tampere University dissertations have been compensated.

PunaMusta Oy – Yliopistopaino
Joensuu 2023

ACKNOWLEDGEMENTS

The research work for this dissertation was done in 2018 – 2022 in the Machine Learning Group and the Computer Vision Group at the Hervanta campus of Tampere University.

I would like to profoundly thank Professors Joni Kämäräinen, Heikki Huttunen, and Esa Rahtu for their invaluable guidance and work during this roller coaster of a journey.

I am very grateful for Professor Tapio Seppänen for acting as my opponent. I am also very grateful for Professor Heikki Kälviäinen for acting as my public defence opponent as well as a pre-examiner. I am also very grateful for Dr. Bertil Grelsson for pre-examining the thesis.

Big thanks to Jussi Puura and Sandvik Tampere for all the critically important support over the years.

Thanks also to the doctoral school of industry innovation (DSII) for the peer support. Thanks to all the co-authors and colleagues at the university, Signal Processing lab, Machine Learning group, and Computer Vision group. Especially Farid Alijani, Bishwo Adhikari, Sakira Hassan, Yi Zhou, Xingyang Ni, Mengyang Chen, Francesco Lomio, Yue Bai, and Pedram Ghazi. Thanks to Kari Suomela, Virve Larmila and many others for their help along the way.

Special thanks to all the loved ones for the great memories, friendship, and support! Eija, Aru, Johanna, Kati, and Jesse.

Keep on rocking!

ABSTRACT

This thesis is about LiDAR place recognition. Place recognition is the problem of being able to recognize already seen or visited places – an important sub-problem of robot navigation. LiDAR sensors offer accurate and cost-effective range and reflectivity data that can replace or complement RGB cameras.

Place recognition has been studied with different sensors and methods for many years. Traditional methods use handcrafted features to match images in order to recognize places. In recent years, the surge of deep learning has made learned features the main approach.

In this work LiDAR place recognition is studied with exported 2D pixel images and deep learning models. Place recognition is posed as an image retrieval problem, where a model is trained to learn a feature space in such a way that the similarity of images can be conveniently compared. With a trained model, one can use an image to search for other similar images, and thus recognize places.

The key finding of the thesis publications is that place recognition with image retrieval using exported pixel images from LiDAR scans is a well performing method, as evidenced by achieving about 80% recall@1 with 5 meter test distance in urban outdoors and 1 meter indoors. The other key findings are: Loop points in the route are detectable with image retrieval type methods. LiDAR is a competitive modality versus RGB. LiDAR depth maps are more robust to change than intensity maps. Generalized mean is a good pooling method for place recognition. Simulated data is beneficial when mixed in with real-world data at a suitable ratio. Dataset quality is very important in regards to ground truth position and LiDAR resolution.

CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

BoW	Bag-of-words is a simplifying representation used in information retrieval.
CNN	Convolutional neural network.
FOV	Field of view.
GeM	Generalized mean. In this thesis GeM most often refers to a generalized mean pooling layer used to produce the final feature vector.
GPS/INS	Global positioning system with inertial navigation system. It calculates position, orientation, and velocity using satellites and inertial sensors.
HOG	Histogram of oriented gradients. It is a feature descriptor.
INS	Inertial navigation system. In practice uses accelerometers and gyroscopes to calculate the position, orientation, and velocity.
LiDAR	Light detection and ranging. (Alternatively also laser imaging, detection, and ranging.) Refers to an active sensor that projects laser beams and measures how they are reflected back from the environment.
PCA	Principal component analysis.
RANSAC	Random sample consensus.
RGB	Red, green, and blue. In this thesis it refers to a typical digital photography image, due to being often defined by the three color values per pixel.
SIFT	Scale-invariant feature transform. It is an algorithm to detect, describe, and match local features.

SLAM	Simultaneous localization and mapping.
SURF	Speeded up robust features. It is a patented algorithm to detect and describe local features.
VGG	Visual geometry group at University of Oxford. In this thesis refers to the network architecture developed in the group named after the group. The network architecture name is often preceding the number of layers in the network, for example, VGG-16.
VLAD	Vector of locally aggregated descriptions.

ORIGINAL PUBLICATIONS

Publication I	Jukka Peltomäki, Mengyang Chen, and Heikki Huttunen. “Semantic segmentation with inexpensive simulated data”. In: *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC).* 2019, pp. 1–6.
Publication II	Jukka Peltomäki, Xingyang Ni, Jussi Puura, Joni-Kristian Kämäräinen, and Heikki Huttunen. “Loop-closure detection by LiDAR scan re-identification”. In: *2020 25th International Conference on Pattern Recognition (ICPR).* 2021, pp. 9107–9114.
Publication III	Jukka Peltomäki, Farid Alijani, Jussi Puura, Heikki Huttunen, Esa Rahtu, and Joni-Kristian Kämäräinen. “Evaluation of Long-term LiDAR Place Recognition”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 2021, pp. 4487–4492.
Publication IV	Jukka Peltomäki, Farid Alijani, Jussi Puura, Heikki Huttunen, Esa Rahtu, and Joni-Kristian Kämäräinen. “LiDAR Place Recognition Evaluation with the Oxford Radar RobotCar Dataset Revised”. In: *Image Analysis: 22nd Scandinavian Conference, SCIA 2023, Levi Ski Resort in Lapland, Finland, April 18–21, 2023, Proceedings*. 2023. Springer. In press.

Author’s contribution

Publication I	As the first author, Jukka Peltomäki built the simulation game environment with virtual LiDAR and RGB sensors with the ground truth data class tagging and automatic dataset exporting function-

	alities, ran all the deep learning segmentation experiments, and drafted the paper.
Publication II	As the first author, Jukka Peltomäki modified a custom deep re-identification network to be used for loop-detection with a newly coded custom data loader and test metrics, ran all the experiments, created the visualizations, and drafted the paper.
Publication III	As the first author, Jukka Peltomäki wrote all the code to modify an image retrieval pipeline to be used for place recognition with a custom data loader, data batch loading for thresholded distances and evaluation metrics for recall, ran all the experiments, and drafted the paper.
Publication IV	As the first author, Jukka Peltomäki wrote all the tools for dataset alignment, ran all the experiments, and drafted the paper.

1 INTRODUCTION

1.1 Background and motivation

Place recognition is an important function for a robot to have. Being able to make sense of where one is, even when missing extrinsic information like the GPS. Understanding that a place is the same when the snow has piled up, the traffic has changed, or the building has been completely renovated or replaced, is essential if the robot is to be able to navigate reliably around our dynamic world and carry on with its business.

We humans have wonderful automatic brain functionality for building internal models out of our environments. We can tell where we are intuitively without much effort. It is only occasionally in new or repetitive places where we struggle and need to pay attention or check the map. Robots do not have the luxury of such wonderful model building capability. The level of operation is much lower.

In robotics, place recognition is used with navigation algorithms such as simultaneous localization and mapping (SLAM)[20, 21, 9], in which the robot reconstructs a metric map of the environment as it goes along, while locating its place on it. Place recognition system is useful when visiting places for the second time, which allows the robot to error-correct its internal map of the environment.

In practice, place recognition can be implemented by being able to match images or other sensory data [2]. Finding the very best matches for a new image from a huge library of other images is called image retrieval [41], and it is one way to implement place recognition. In experiments, correct matches for images are found from another list of images, and the ratio of found correct matches is calculated in order to evaluate the performance of the matching method.

Different sensory data for place recognition is available. One sensor modality is LiDAR (Section 2.6.1), which has improved in recent years. A typical LiDAR used in robotics and autonomous vehicles can provide range and surface reflection intensity data in decent quality and speed. It is an active sensor that shines laser beams into its environment and senses how they are reflected back. For that reason the usual range is quite limited compared to photography and other methods that passively capture natural phenomena such as photons. Typical RGB photography images are not used in this thesis, but instead LiDAR scans where the range and reflectivity data is projected as 2D pixel images (Section 2.6.2).

An illustrative example of LiDAR data and difficulty of matching is presented in Figure 1.1, which shows visualized LiDAR images as well as normal photograph images for better understandability. The situation depicted is such that we have an image we are trying to find matches for (the query image) on top. In the middle there is an image of what the place recognition system thought was the best match, but in this case the system got it wrong, and the best found match is actually over a kilometer away. The bottom images are for a match we know to be correct (ground truth). In contrast, a well working example would have the best match from the same place as the query image.

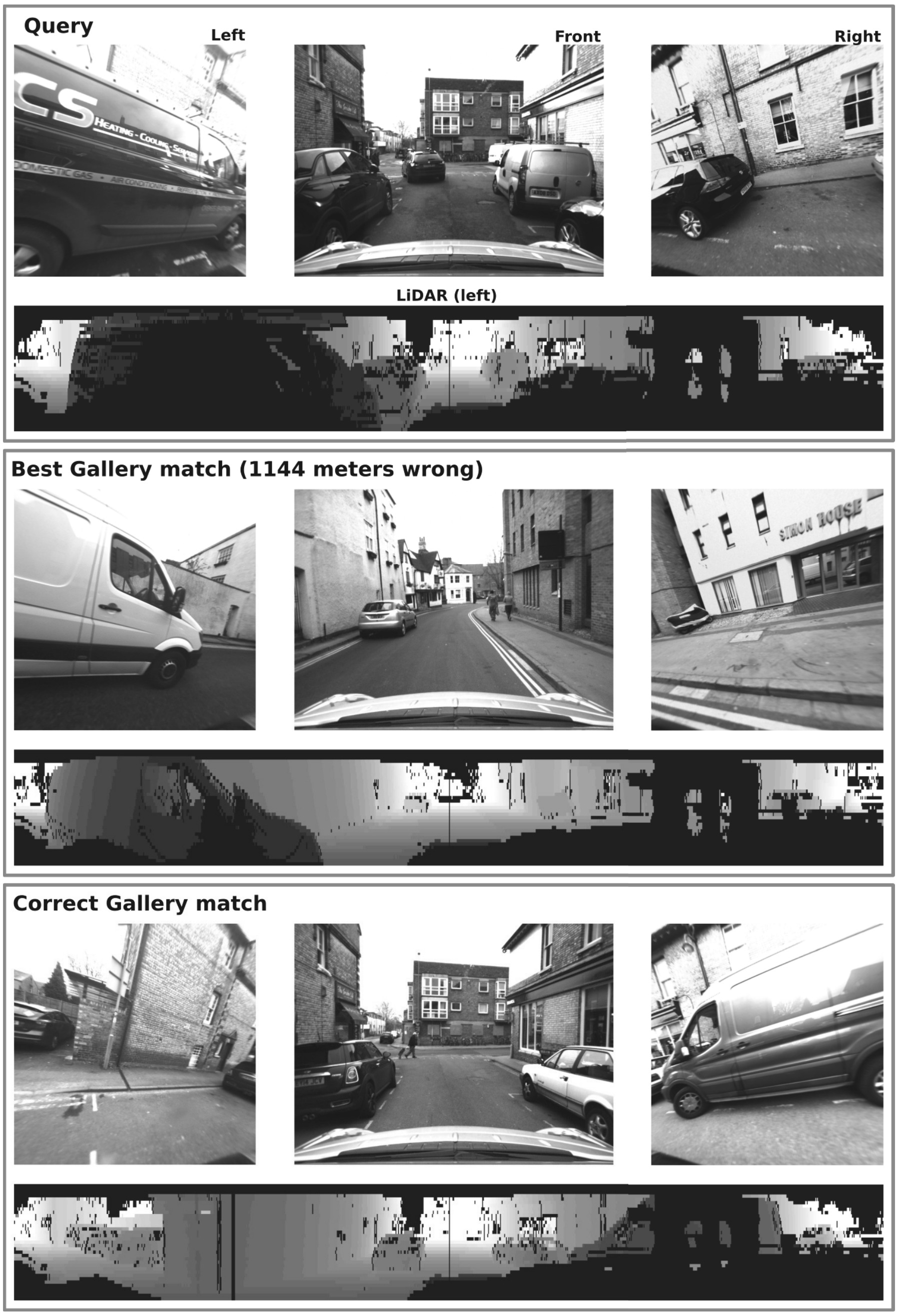


Figure 1.1 Example situation for LiDAR image matching. The top LiDAR image is the image we are finding matches for – the query image. The middle LiDAR image is the wrong match that the place recognition system thought was correct. The bottom LiDAR image is one of the existing correct matches for the query image. The left, front, and right images are not used in the matching, they are just for better human readability. Publication IV, Figure 1. (The data is from the Oxford Radar RobotCar dataset [5].)

1.2 Research focus

The research in this thesis is focused on LiDAR place recognition with image retrieval methods. More specifically, the focus is on extracting the best possible embeddings for the exported 2D pixel images, which can be used to find the best matches for a given input. The main point is to explore the performance of place recognition in different scenarios to demonstrate that LiDAR place recognition with image retrieval methods does reach reasonable recall performance, and that it is thus realistically applicable to robot navigation challenges.

The focus includes the ability to detect loops in the route, the performance effects of architectural choices within the image retrieval pipeline, the suitable mix ratio of simulated to real world data, and how much the dataset ground truth and LiDAR quality affect the performance. These topics were explored by developing and executing the suitable test scenarios and answered by evaluating and reporting the resulting performance.

The research focus is captured in the following research questions:

- How to apply existing image retrieval methods for place recognition?
- How to handle LiDAR in image based methods?
- How to improve the recall performance of the deep image retrieval based place recognition method?

The work and publications are applied research, in which the theoretical ideas from convolutional neural networks, image retrieval, and place recognition are understood and experimented with. The topic is closely related to image retrieval [41, 2], visual place recognition [46, 61, 11], localization [57, 67, 68], and simultaneous localization and mapping (SLAM)[20, 21, 9].

1.3 Summary of original publications

This thesis consists of four main authored publications, each of them looking at the main problem from different perspectives.

Publication I explores how artificially generated data can be mixed with real world datasets to improve results. The context is image segmentation, with a convolutional processing pipeline. The key result is that adding simulated data to augment real-world data improved results, as long as the amount of simulated data was below 50 to 70 % of the combined dataset. Visually complex objects, such as trees, benefited from simulated segmentation ground truth data, because the simulated segmentation is more detailed than a human annotator would perform.

Publication II deals with the problem of loop-closure, where the embeddings are trained and extracted similarly to place recognition. The test metric is simpler and binary: is the query in a loop zone or not. The dataset used in the publication is of high quality, but private and small. Backbone network, loss function, and augmentation choices are also explored. The key result is that it is feasible in the depicted scenario to detect loop points with image retrieval type methods.

Publication III evaluates two existing pooling methods for LiDAR place recognition: Generalized mean pooling (GeM) and NetVLAD. Two very different datasets are used: an urban outdoor dataset with a high quality LiDAR sensor and an indoor office space dataset with a low fidelity sensor. Also different LiDAR modalities of depth and reflectivity are compared to RGB. The key results are that in place recognition, LiDAR is a competitive modality versus RGB, depth maps are more robust to change than intensity maps, GeM is a good pooling method, and 80% recall@1 is achievable with 5 meter test distance in urban outdoors and with half a meter test distance in office indoors.

Publication IV proposes improved position ground truth and evaluation protocol for place recognition with the Oxford Radar RobotCar dataset. The radar odometry ground truth supplied by the dataset gets appended with manually tuned starting poses for each sequence. This allows for more accurate benchmarking on this popular dataset in tasks relying on aligned sequences, such as place recognition. The key result is providing fixed radar odometry ground truth starting positions and demonstrating the better testing accuracy.

2 PLACE RECOGNITION BACKGROUND

Place recognition is the problem of being able to recognize already seen or visited places. It is an important sub-problem of navigation. Recognition requires some kind of a library of stored history of visited places, a sensor to get information about the places, and a computational method to do the recognition. [46, 51]

The library does not have to exist before embarking on a voyage of discovery, as it can be collected as the agent moves about in the exploratory space. But it can improve the recognition performance a great deal if such a library is already collected and processed into powerful recognition models beforehand. For example, if a robotics delivery company would like to employ new robots to its fleet, it would make sense to use the best available models already gathered from the fleet, rather than having the robot to learn the streets from scratch.

The utilized sensors can be any kinds of available sensors: cameras, LiDARs, radars, sonars, event-cameras, and so forth. Different sensors thrive in different environments, and have different capabilities and prices. Many robotic applications employ multiple sensor modalities for best performance. For instance, typical RGB-cameras are relatively cheap and offer information in a format easily understandable by a human – the color, resolution, and field-of-view are close enough to what our eyes can see and our brains intuitively interpret. A LiDAR provides a sparser array of accurate range information, which is very useful in cases such as docking a space shuttle to a space station.

The computational method to process the sensory input for place recognition can also be implemented in different ways. In recent years the deep neural network implementations have dominated all kinds of machine learning tasks, and so has been the case in place recognition, as well. The range of best methods can also be plotted somewhere on the accuracy–computability curve. A well recognizing method can be so heavy that it can only be run slowly on a server, while some light method can run in real-time on modern mobile robotic hardware.

A machine learning approach is to first gather lots of data, then use that data to train a good place recognition model on a server, and then deploy the model on mobile hardware. The learning phase is more computationally intensive than just using the same learned model. Then, while using the model and navigating the environment, the agents can collect more sensory data, especially from challenging situations and edge cases where the existing model performs poorly. This gathered data can then be used to train better and better models to be deployed, benefiting the whole fleet of agents.

Localization is a problem related to place recognition. In localization the whole 3D pose is considered, so it is a harder problem than just recognizing the place. [57, 67] Place recognition is sometimes used as the first step in more resource intense methods, such as localization or accurate point cloud matching, in order to significantly cull the amount of input data for the heavier process.

2.1 Visual place recognition

Images can be matched by their visual similarity, which allows for place recognition. Traditionally handcrafted local feature methods such as SIFT [45], SURF [7], and HOG [18] have been used for visual place recognition. These local feature vectors have been combined to global feature vectors most notably with the visual bag-of-features model, improved upon by, for example, DBoW [23] and FAB-MAP [17]. Traditional place recognition methods have been surveyed by Lowry et al. in 2016 [46] and by Masone and Caputo in 2021 [51].

Deep learning (more in 3.1) based learning methods have surpassed the handcrafted features in performance. Local features are still used, for example DELF [53]. Global features offer faster matching after extraction, but local features might allow for more accurate matching. Methods such as NetVLAD [3] and GeM [61] can be layered on top of any convolutional pipeline for a decent system. Combining local and global features have been proposed, for instance DELG [11]. Many learning feature extractors can take advantage of any typical CNN architecture, such as ResNet [27], VGG [70], MobileNet [31, 66], or EfficientNet [76]. The training can leverage ImageNet [64] pre-training to save resources. Localization systems can be used for place recognition as well, even though they also do provide extra information. Hierarchical localization with SuperPoint [67] and SuperGlue [68] is one of the best performing methods for visual localization [57]. Modern place recognition methods have been surveyed by Masone and Caputo in 2021 [51].

2.2 LiDAR place recognition

LiDAR place recognition utilizes LiDAR sensors (2.6.1) for place recognition. Most visual place recognition methods can be used with LiDAR data by first exporting the LiDAR point clouds to pixel images, as has been done by many [72, 28, 79, 36, 86, 63, 84]. A season-invariant and viewpoint-tolerant pixel image exportation method for point clouds has been proposed by Cao et al. [12]. LiDAR sensors are explored more in depth in Section 2.6.1, and the point cloud to pixel image exporting process in Section 2.6.2.

There are also place recognition methods which process the LiDAR point cloud data directly, without exporting to pixel images. PointNetVLAD [79] can take point clouds as an input, and is based on PointNet [59] with NetVLAD [3] pooling. MinkLoc3D-SI [87] is based on MinkLoc3D [38], and introduces place recognition with sparse convolutions, spherical coordinates, and intensity values. Point clouds can be segmented into discrete voxels as input to 3D convolutional pipelines. This has been experimented for tasks other than place recognition [52] [83], but these pipelines might also be used to pool global features for place recognition.

Scan Context is an egocentric spatial descriptor for place recognition [36]. ISHOT [26] is a local descriptor that combines LiDAR range and reflection data. Semantic segmentation data has been combined with the point cloud for a descriptor [44]. BVMatch is a bird's eye view LiDAR descriptor [47].

Hybrid methods have also been introduced that combine the usage of LiDAR with other sensory input. Collier et al. [14] combined LiDAR and RGB inputs with FAB-MAP [17, 16]. They used VD-LSD [75] fetures for LiDAR and SIFT [45] features for RGB. Joint learning with LiDAR and radar has also been proposed for place recognition [85].

2.3 LiDAR datasets

Along with the rising interest in autonomous driving, various LiDAR datasets have become more popular. There are several datasets available with good quality ground truth positioning and LiDAR data. However, indoor datasets were hard to find at the time of choosing. For good place recognition results, the sensor data needs to be of high enough quality, and therefore recorded with a LiDAR that can produce big enough point clouds per scan. A 32 beam 360 degree LiDAR is a decent baseline.

MulRan [37] is an urban outdoors dataset with high quality LiDAR data and their own egocentric spatial descriptor [36]. They also assess the suitability of several similar datasets for LiDAR place recognition.

Many datasets with high quality ground truth position and LiDAR data are usable for place recognition, even though they are not specifically intended for it. High quality LiDAR datasets include: The Newer College Dataset [62], University of Michigan North Campus Long-Term (NCLR) [13], nuScenes [10], Kitti [24], Complex Urban [33], Ford Campus [55], and Ford Multi-AV Seasonal [1], Waymo Open dataset [73], The Oxford RobotCar [49], and The Oxford Radar RobotCar [5].

While all the LiDAR datasets are also usable for RGB, there are datasets available only for RGB place recognition: Mapillary Street-Level Sequences [80] is a large dataset for lifelong place recognition. Eynsham [15] is a dataset with a 35 km long sequence that is traversed twice. The Nordland dataset [74] consists of 3000 km in northern Norway over four seasons. St. Lucia dataset [25] from the urban streets of Brisbane covers several times of day over a few days three weeks apart.

2.4 Loop-closure

Place recognition is related to the problem of loop-closure, in which the recognition of already visited places is used to reduce error in the route mapping. A loop is formed in the route when the agent comes back to a place it has already visited. The reason a loop point is important is because a loop point gives very reliable information about two points in the route being at the very same location. If the internal navigation odometry has drifted along the traveled way, as is most often the case, a loop point provides a stable check-point to rectify the drift. [40, 30]

Illustrative visualization of using the loop-closure detection method is seen in Figure 2.1, in which a loop point and a non-loop point can be told apart by looking at the feature space distances of the best matches. This indicates that the model has learned to distinguish nearby images by having them be closer in the feature space just as they are in the real-world.

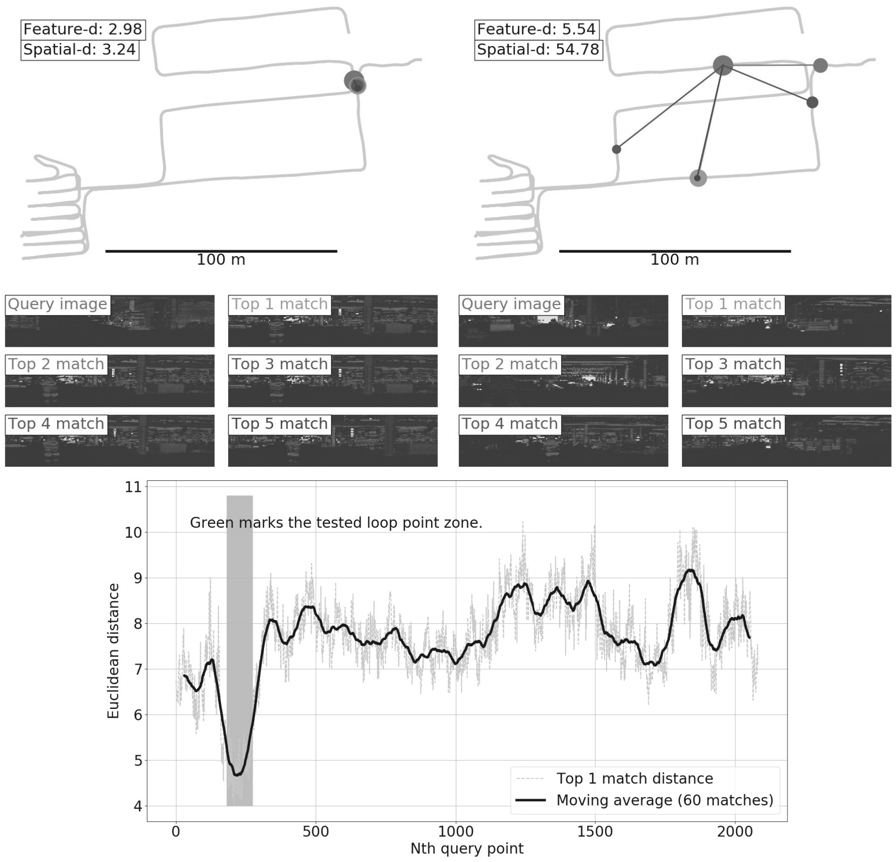


Figure 2.1 © 2021 IEEE. Publication II, Figures 2 and 4. Examples of loop closure (top-left) and non-loop closure locations (top-right) where the method effectively detects the loop closure. In the loop-closure location the best five matches (red points) are all near the query spatial location (green point) and within a small feature distance while in the non-loop-closure location the feature distances are large and the matches are found in random spatial locations. The model has been trained to match nearby images. The detection for loop points could be thresholded with the feature space Euclidean distance, as seen in the bottom figure. Data from a private factory dataset. (Note: the graph is from a different model, so the distances do not match the map images above.)

2.5 Image retrieval

Image retrieval is a practical way to implement place recognition [2]. Image retrieval is the problem of using images to find other similar images [41]. There is a pool of images called the gallery, which is where we are trying to find images from. An image that is used as the search term is called a query image.

A query image is given to the image retrieval system, and the image retrieval system gives the best matching images from the gallery set as a result to the query image. The result list can include n best matches, or it can be thresholded by a similarity score to the query image. The image retrieval system therefore needs to be able to compare images to each other and assign similarity scores to image pairs. In the thesis, similarity is dealt with at two levels: Metric learning (Section 3.3.1) with a deep neural network (Section 3.1) is used to learn relevant similarities between images, and Euclidean distance (Section 2.5.1) is used to compare the learned similarities.

To test a query set against a gallery set, first we get the feature vectors from the model for all the images in both sets. Then we can calculate the Euclidean distance of each query set image to all the gallery set images. The Euclidean distances can then be sorted from smallest to largest. The shortest distance is the best match that the image retrieval system provided, and it may be correct or incorrect depending on the quality of the model and how similar the images in the gallery are. A high level view of an image retrieval system can be seen in Figure 2.2.

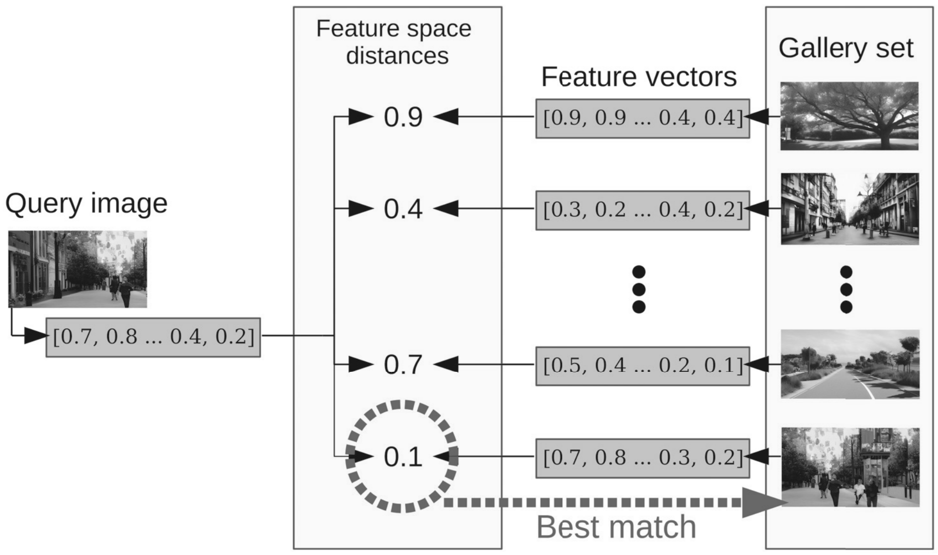


Figure 2.2 Image retrieval system high level diagram. A query image can be given to the system, and the system finds similar images from the gallery set. A function for calculating feature vectors from images is assumed, and all the images have the feature vectors calculated. The query image feature vector is compared to each feature vector from the gallery set. The comparison can be done as the Euclidean distance. The closer two images are in their feature space Euclidean distance, more similar they are. In the figure it can be seen that the last image in the gallery set is the best match for the query image.

2.5.1 Euclidean distance

In image retrieval, the learned similarity between images is often represented as two feature vectors – one extracted from each image. To get a single number value for easy comparison, the similarity between two vectors can be calculated by various distance metrics. In the context of this thesis, Euclidean distance is used as the distance metric, and the final similarity score is the inversed feature space Euclidean distance between the images, so that the closer the distance, the more similar the images are expected to be.

The Euclidean distance for two feature vectors, p and q, in the n-dimensional feature space can be calculated by taking a square root of the sum of all the dimension-wise squared subtractions:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \tag{2.1}$$

2.5.2 Feature vectors

A feature vector is a vector that describes a piece of data, such as an image or a feature point in an image. A good feature vector is like a compressed essence of the data, focusing in on the areas that are important for the task at hand. Feature vectors can be of any length. [65]

An l-length feature vector habits an l-dimensional feature space. Similarity between different data inputs can be distinguished by how close they are to each other in the feature space. For example, a feature vector of a model that is trained to be invariant to lighting conditions can have largely different input images with lots of the pixel values completely different, and still be very close to each other in the learned feature space. A model that is fed an image, produces a feature vector describing the features the training of the model has deemed important.

In the image retrieval context, feature vectors are usually either global or local features. A local feature describes an area around a point in the image. Often the methods find many local features from the image. A global feature describes the whole image in one vector. Global features are generally faster to extract and to use, but local features often allow for more accurate matching of the images, and there is also geometric information of the local points that can be used.

2.5.3 Comparing the feature vectors faster

The extracted local or global feature vectors are compared to each other to find the best matches. Matching is usually comparing feature space distances, for example, as Euclidean distance (equation 2.5.1). This can be done one by one, but research work has been conducted to speed up the process. Speed of search is particularly beneficial in practical real-time scenarios or massive datasets.

Many methods [3, 61, 53] use principal component analysis (PCA) to decrease the length of feature vectors for faster comparability and lower memory consumption. Optimized software and methods, such as FAISS [34] and locally optimized product quantization [35], can be used to speed up the similarity search process for feature vectors. Hierarchical grouping has been found to decrease the linear complexity to logarithmic, at least for exported LiDAR imagery [48].

For increased accuracy for matching images with local feature vectors, geometric verification can be used. The local feature descriptor coordinates in the images are calculated for inlier points with the Random Sample Consensus (RANSAC) algorithm [22], and the inlier count is used as matching similarity instead, as in the DELF [53] method.

2.6 Exporting LiDAR point clouds to 2D pixel images

2.6.1 LiDAR sensors

LiDAR is an active sensor that sends a laser beam into the environment and senses how the beam is reflected back. The accurate measurement of the time it took for the beam to reflect back is used to measure the distance to the reflected object. In addition to range, the sensor often also provides some kind of reflectivity data of how well the encountered material reflected the laser beam. LiDAR is an interesting robotics sensor due to accurate range information, which complements an RGB camera. The price has also become reasonable for practical applications.

LiDARs are often called either 2D or 3D. Two dimensional LiDARs operate on a single plane in space, producing two dimensional point clouds. They often use only a single laser beam, which is fixed vertically but might be rotating horizontally. Three dimensional LiDARs also rotate the laser vertically or, more often, have multiple lasers with different vertical angles, thus capable of producing three dimensional point clouds.

In the context of robotic navigation, the focus in this thesis is on the relatively cheap commodity hardware LiDAR sensors. These sensors typically are rotated around an axis, giving a 360 degree horizontal field of view. Several different vertical angles can be scanned, giving the rotating unit a 3D view of the environment. A typical commodity 3D LiDAR, such as the Velodyne HDL-32E, has 32 channels, a 360 degree horizontal field-of-view and a 40 degree vertical field-of-view. This is good enough quality to be used for place recognition in urban outdoors experiments, as seen in publication III, Evaluation of long-term LiDAR place recognition.

3D LiDAR scan data is usually processed as a three dimensional point cloud. One rotation of the sensor is though of as one scan. By combining multiple scans one can build a 3D point cloud of a larger environment, such as a room, a cave, or a city.

2.6.2 Projecting LiDAR points to a plane

Point cloud data can be exported to a panoramic 2D image, in which the pixel values represent the range or reflection in that point of the scan. These exports are also called depth maps and intensity maps, respectively. Points clouds, like any other three dimensional data, can be projected onto a two dimensional plane. LiDAR point clouds are sparse, so the resolution has a practical upper limit and it is markedly worse compared to RGB photography.

A good quality commercial 3D LiDAR sensor, Ouster OS1-128, can produce enough points for a 2048 by 128 pixel image per scan. A medium quality LiDAR, Velodyne HDL-32E, can produce a 1024 by 32 pixel image. Cheaper 2D LiDARs only produce measurements on a plane, which in practice can be exported into one dimensional pixel "images". For example, the 2D LiDAR used in COLD dataset [58] can export only to 512 by 1 pixel images.

For the projection, we need to know the vertical and horizontal field-of-view and resolution for the sensor. Then the points in the space can be mapped onto a cylindrical surface that can be displayed as a plane. Either the range or reflection data can be used as the greyscale color for the pixels. The 32 bit float values indicating the distance or reflectivity can be scaled to a range suitable for images. The maximum real-world range of the LiDAR sensor can be set as the color white, and the minimum range as the color black, and the other values interpolated to somewhere inbetween. This results in a discrete gradient with useful data, although the gradient is not consistent between differently ranged sensors. The 32 bit float distances can be quantized down to 8 bits, so that the biggest value 255 is scaled to the maximum real-world range of the LiDAR sensor. The projected and quantized point cloud produces a low resolution image, as seen in Figure 2.3. The low resolution is especially pronounced because of the wide full circle field-of-view, which makes the pixels per degree very low. The sensors also often have a big difference between the vertical and the horizontal field-of-view, which makes the aspect ratio of the projected image quite high. For all these reasons the projected LiDAR pixel images are often visualized with suitable color mapping and rescaling, as seen in Figure 2.3. The neural networks, of course, use the unaltered greyscale images as data.

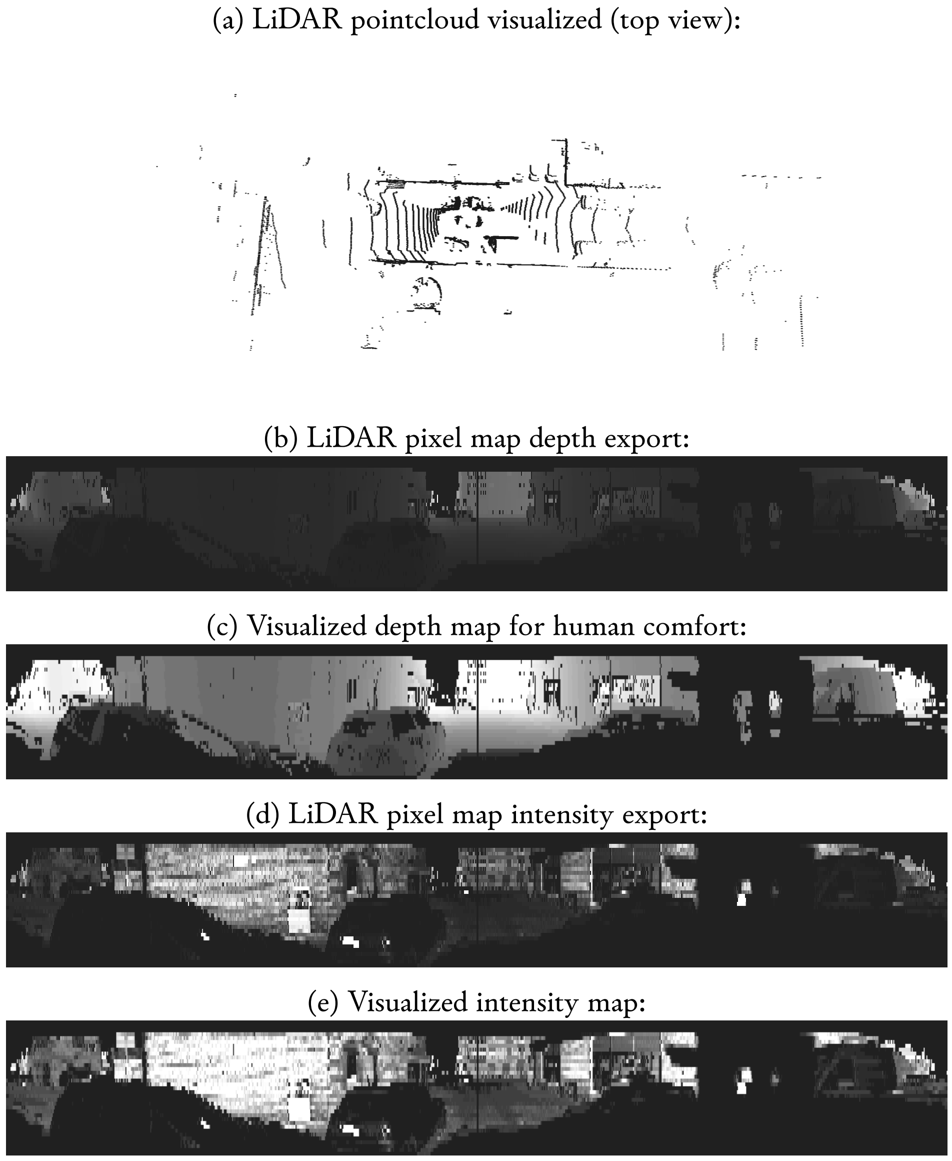

(a) LiDAR pointcloud visualized (top view):
(b) LiDAR pixel map depth export:
(c) Visualized depth map for human comfort:
(d) LiDAR pixel map intensity export:
(e) Visualized intensity map:

Figure 2.3 LiDAR data is usually stored as (a) point clouds, which can be visualized as sparse points in a three dimensional space. After projecting the point cloud depth to a (b) greyscale pixel image, the image is ready to be fed to the image retrieval system for place recognition and other tasks. (c) The exported image can also be visualized – brightened, colorized, and scaled – for better human understandability. (d) The raw intensity map and (e) the visualized intensity map are visibly different from the depth maps. In the pixel images the x-axis is the horizontal 360 degree field of view, and the y-axis is the vertical FoV. The depth map pixel brightness corresponds to the distance from the sensor. In the intensity map the pixel brightness corresponds to the intensity of the reflection from the material. The point cloud data is from the Oxford Radar RobotCar dataset [5] left side LiDAR.

3 DEEP PLACE RECOGNITION

3.1 Deep models

Many image retrieval and place recognition methods employ deep learning. Convolutional deep learning networks are powerful at learning features from sensory data such as images, and therefore perform well for place recognition.

Deep learning means neural networks that have more than a few layers. There is no clear limit, and it depends on the application. In image processing there are typically between a dozen and a few hundred layers. Deep learning is one type of machine learning. In machine learning the algorithm is fed data in the form of inputs and outputs, and the system learns a connection between the two. Machine learning is a sub-field of artificial intelligence.

There are widely used open source tools freely available for building and experimenting with neural network architectures, loss functions, evaluation metrics, workflows, and so forth, with ready made implementations available. Notably, Tensorflow [50] and PyTorch [56] are used commonly in the field.

Neural networks and deep learning are widely researched topics which go well beyond images and place recognition. Natural language processing (NLP) for translating and generating text, text-to-speech and speech-to-text for switching modalities, and training agents via reinforcement learning are just a slice of the application domains.

3.1.1 Convolutional neural networks

Convolutional layers allow spatial data to be processed with fewer neural connections than fully connected layers. The data, for example an image, is processed by different learned filters which spatially go over the image. Convolutional layers are often interlaced with pooling layers to form a convolutional pipeline. In the network, the layer size starts with the spatial size of the input data, which gets smaller and smaller towards the end of the pipeline, whereas the filter size increases. It could be said that information about the image gets more generalized towards the end of the convolutional pipeline, as there is more information about a larger spatial area.

The fact that convolutional layers have less weights than fully connected layers, means that they are faster to compute. The speed increase has brought many problems previously seen as too slow to calculate into the field of practical experimentation. Convolution is not suitable for everything though, as sometimes dropping connections and exploiting spatial information is not preferred, and in many situations a fully connected or some other type of layer is more suitable. Often for sensory input at least, convolutional pipelines are generally good.

Deep learning and convolutional networks have been developed by many over the years, such as LeCun, Bengio, and Hinton [42] and their collaborators [43] [8] [39] [71]. The current community has naturally grown much larger.

3.1.2 Backbone networks

A backbone network refers to the neural network that is used as the basis for the feature vector extraction. For place recognition, the backbone network is usually stripped of the possible extra layers at the end, such as categorization. Typically, for place recognition, a layer of pooling is added on top of the backbone network to produce better performing feature vectors. In this thesis two pooling methods are explored: generalized mean pooling (GeM) [61] and NetVLAD [3].

In practice, the backbone networks are often the same ones which are generally used in many image processing tasks. Networks such as ResNet [27], VGG [70], EfficientNet [76], and MobileNet [31] architectures. These backbone networks usually have pre-trained weights provided by the developers. The pre-training is often done with the ImageNet dataset [19] and the image classification task. Using the pre-trained weights as a starting point saves resources, as training big networks from scratch would require lots of training cycles.

It is important to note that training done for one visual task is most often applicable to other visual tasks as well. The convolutional network learns to process images in useful and general ways. A network trained for image classification is surprisingly good for place recognition, as the network has already learned to extract essential information from the images. Still, the pre-trained network is often further trained with task specific data in order to improve the results. The further task specific training is called fine-tuning.

3.2 Pooling methods

When the extraction of good suitable feature vectors (embeddings) for the dataset is required, a typical way is to have a general convolutional neural network pipeline for extracting the features and add a pooling layer on top of it to produce the final embedding. While all the layers in the pipeline can be used as feature vectors (with or without pooling), it is usually the last layer that is used in order to get the shortest vector with the highest abstractions of the features. The three dimensional last layer output tensor (with width, height, and depth or filter count) is concatenated as an l-dimensional vector (where l = w*h*d) and fed to the pooling layer. For place recognition, the network is trained in a way that the Euclidean distance of pooling layer 2D vectors of two images is indicative of their real-world distance. A high level diagram of a deep convolutional neural network with a pooling layer on top is seen in Figure 3.1.

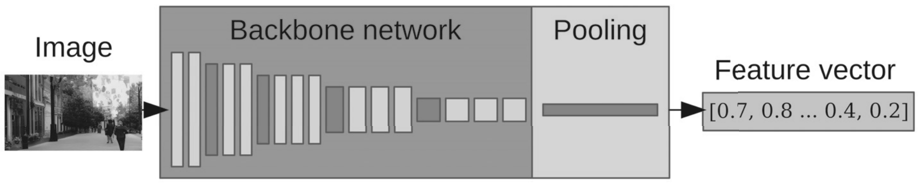


Figure 3.1 Deep backbone network with suitable pooling on top. The network is fed an image, and the output is a feature vector.

3.2.1 Generalized mean pooling (GeM)

Generalized mean pooling (GeM) is a learnable generalization between average pooling and max pooling. It was introduced in CNN Retrieval, a state-of-the-art image retrieval system by Radenović et al. [60, 61]. They use GeM to generate the final feature vector after the convolutional pipeline. The GeM calculates the generalized mean of each channel in a tensor. The GeM layer is added on top of the last convolutional layer in the pipeline. The 3D tensor output χ, sized $W * H * K$, is fed into the pooling layer, where the GeM vector **f** is calculated:

$$\mathbf{f} = [f_1 ... f_k ... f_K]^\top, f_k = \left(\frac{1}{|\chi_k|} \sum_{x \in \chi_k} x^{p_k} \right)^{\frac{1}{p_k}}, \tag{3.1}$$

The set of activations χ_k is calculated for the feature map $k \in \{1...K\}$. GeM is a generalization of average pooling and max pooling, and the learned parameter p_k controls how close the per channel pooling is to average ($p_k = 1$) and max ($p_k = \text{inf}$). The pooled vector $\mathbf{f}$ is l_2-normalized before using.

3.2.2 NetVLAD

NetVLAD [3] is a deep learning place recognition method. The main components of NetVLAD are the NetVLAD (pooling) layer, triplet ranking loss, and PCA dimensionality reduction. The NetVLAD pooling layer is a differentiable version of the VLAD [32] encoding of SIFT [45] features. VLAD encoding outputs a $K \times D$-dimensional matrix, in which K is the number of visual words and D is the number of feature dimensions.

The VLAD encoding,

$$V(j,k) = \sum_{i=1}^{N} a_k(\mathbf{x}_i)(x_i(j) - c_k(j)), \tag{3.2}$$

is calculated from the N SIFT vectors. $x_i(j)$ and $c_k(j)$ are the j-th dimensions of the i-th SIFT vector and k-th cluster center. Feature x_i belonging to the k-th visual word is noted $a_k(\mathbf{x}_i)$.

The resulting VLAD matrix encodes feature distances from the visual words. VLAD encoding is more powerful than Bag-of-Words (BoW) encoding, but it is also slower to process [32]. The NetVLAD layer implements a learnable VLAD encoding:

$$V(j,k) = \sum_{i=1}^{N} \frac{e^{\mathbf{w}_k^T \mathbf{x}_i + b_k}}{\sum_{k'} e^{\mathbf{w}_{k'}^T \mathbf{x}_i + b_{k'}}} (x_i(j) - c_k(j)), \tag{3.3}$$

The $\{\mathbf{w}\}_k$, $\{b\}_k$, and $\{\mathbf{c}\}_k$ are sets of learned parameters.

3.3 Loss functions

Loss functions are at the very core of how the learning happens in a neural network. For place recognition, the loss functions are similar in that they implement metric learning (Section 3.3.1). The learned representation feature vector provides that similarity can be measured by standard distance functions such as Euclidean distance. All the loss functions introduced below push matching images closer towards each other and non-matching images further from each other in the feature space.

3.3.1 Metric learning

Metric learning means learning to measure similarities between objects [54, 81]. In image retrieval and place recognition, it is beneficial to know similarities between objects. Meaningful real-world similarities between complex patterns, such as images, cannot be calculated with simple metrics like the Euclidean distance. We use a deep convolutional neural network to learn meaningful similarities between images.

For the training, the network is given pairs or triplets of images with the knowledge of whether they are a match. In the loss function, the total model error is calculated for the input data based on the ground truth. The total error is backpropagated through the network to find weight-wise error contribution, after which the weights are updated according to the chosen learning parameters. In essence, the loss function for the training pulls matches closer together in the feature space, and pushes non-matches further apart. When repeated enough times with enough data, the network has learned a metric to measure similarity in images in the specific way that the matches and non-matches were chosen. A high level diagram of a metric training process can be seen in Figure 3.2.

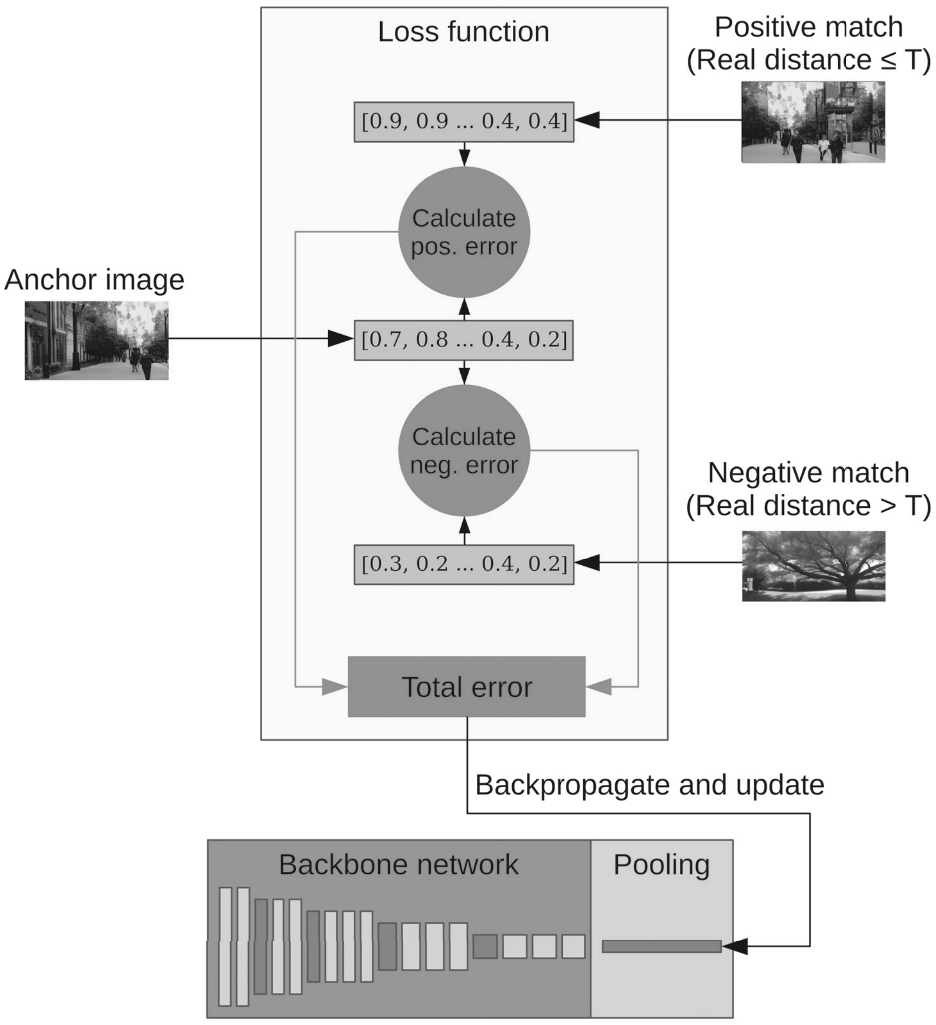


Figure 3.2 A simplified training process of the place recognition system using triplet loss function. For each anchor image (Section 3.3.3), a positive match and a negative match are chosen based on ground truth real-world position. A positive match needs to be within the chosen training distance threshold T, and a negative match needs to be over the threshold. The feature vectors for all three images are calculated. The model errors for both pairs of anchor–positive and anchor–negative are calculated. The combined total error is backpropagated and the training step is carried out.

3.3.2 Contrastive loss

The contrastive loss is the default used by CNN Retrieval system with GeM [61]. Training is done with a contrastive training loss function used in a Siamese network configuration, in which two inputs are fed into two identical networks with shared weights, and the loss is calculated from the output of both networks. Unlike triplet style loss functions, contrastive loss only deals with images as pairs. The input images are fed as pairs (i,j) with labels $Y(i,j) \in \{0,1\}$, where 0 means that the images are non-matches and 1 means that the images are matches, as seen in the contrastive loss equation:

$$L(i,j) = \begin{cases} \frac{1}{2}||\mathbf{f}(i) - \mathbf{f}(j)||^2, & Y(i,j) = 1 \\ \frac{1}{2}(max\{0, \tau - ||\mathbf{f}(i) - \mathbf{f}(j)||\}^2, & Y(i,j) = 0 \end{cases} \tag{3.4}$$

The parameter τ is the enforced margin between the non-matching images

3.3.3 Triplet loss

The triplet loss function [81] [69] takes as input three images: an anchor, a, a match for the anchor, p, and a non-match for the anchor, n. The anchor image is chosen randomly, and used as a point of reference to simultaneous matches and non-matches.

$$L = \sum_{i}^{N} \left[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha\right]_+ \ , \tag{3.5}$$

The α is the distance margin between non-matching images.

Hard negative and positive mining is a process that searches for negative and positive samples (in relation to the anchor image) which the model still has not learned to distinguish properly. It is possible to use only either hard negative or hard positive mining, as using both is in no way necessary. With triplet loss, the learning can be significantly improved with hard mining when forming the training batches. The reason is that as the size of the dataset grows, the amount of available triplets (anchor, positive, and negative) grows cubically, making the full exploration of big datasets impractical. A powerful model (such as a convolutional neural network) learns the trivial triplets quickly, making most of the data irrelevant for further learning. Improved learning performance is gained by having a hard mining process to feed the learning process with hard triplets that are relevant to improving the model. [29]

3.3.4 Hermans triplet loss

Hermans triplet loss [29] is an improvement to the triplet loss (Section 3.3.3). The upside of Hermans triplet is speed, as it does not require hard negative mining to achieve good results. The loss function is more complicated than triplet loss, because Hermans triplet loss considers all anchor-positive pairs within the batch.

$$\begin{aligned} L_{LG}(\theta;X) = \sum_{i=1}^{P}\sum_{a=1}^{K}\Bigg[&\log \sum_{\substack{p=1\\p\neq a}}^{K} e^{D\left(f_\theta(x_a^i),f_\theta(x_p^i)\right)} \\ &+ \log \sum_{\substack{j=1\\j\neq i}}^{P}\sum_{n=1}^{K} e^{\alpha - D\left(f_\theta(x_a^i),f_\theta(x_n^j)\right)}\Bigg]_+ \end{aligned} . \tag{3.6}$$

The parameter D is the distance function (in this thesis, Euclidean distance is used). The f_θ is the feature space mapping function (in this thesis, a convolutional neural net with suitable pooling is used). The P is the number of classes available (within a batch), and the K is the number of samples (within a batch) from each class. The X is a batch of training data. The subscript '+' denotes the positive numbers only and therefore the minimum value for the loss is zero.

3.3.5 Lifted structured loss

Lifted structured embedding loss [54] implements a similar metric learning concept as the triplet loss. In lifted structured embedding loss every image in the training batch is compared against every other image in the training batch. The batches are mined in a specific way, as a few randomly chosen positive pairs are used to mine for neighboring hard negatives. The overall performance of lifted structured loss is demonstrated to be above triplet and contrastive losses [54].

The lifted structured loss function,

$$\begin{aligned} \tilde{J}_{i,j} &= \log\left[\sum_{(i,k)\in N} e^{(\alpha - D_{i,k})} + \sum_{(j,l)\in N} e^{(\alpha - D_{j,l})}\right] + D_{i,j} \\ \tilde{J} &= \frac{1}{2|P|}\sum_{(i,j)\in P} \max(0,\tilde{J}_{i,j})^2 \end{aligned} . \tag{3.7}$$

optimizes a smooth upper bound $\tilde{J}(D(f(x)))$ to avoid bad local optima. The N is all the negative samples in the batch, and the P is all the positive samples in the batch.

4 DATASETS, EXPERIMENTS, AND PERFORMANCE EVALUATION

4.1 Overall deep architecture for place recognition

To setup image retrieval for place recognition, minor modifications are needed to typical image retrieval pipelines. Basically we want the model to learn a feature space that corresponds to a physical space in a way that two images taken close-by in real life have a short distance in the feature space, as well. For this, we choose a training threshold distance that we use to sort images into matches and non-matches according to their ground truth position data.

In the thesis publications, various architectural choices for place recognition are explored. The common elements through all are: Feature extraction is done with a convolutional backbone network with an added global pooling layer at the end. The loss functions used in image retrieval in this thesis and the publications always implement metric learning, and are either based on triplet or paired samples. In a triplet the samples are an anchor, a positive match, and a negative non-match. In a pair, as used in contrastive loss, an anchor is combined with either a match or a non-match.

To achieve the wanted feature space, the training batches (for triplet data) are built as follows:

1. Choose a random image from the training set. This is the anchor sample.
2. As a positive match, choose a random image that is within the chosen training threshold distance to the anchor.
3. As a negative match, hard negative mine an image that is outside the chosen training threshold distance to the anchor.

This way the categories are not pre-chosen, but are dynamically built by mandating that matches are anything near enough and non-matches are anything far enough.

Otherwise the training is similar to image retrieval with static categories. The dataset loader is also extended with a loop that pre-builds a list of all the nearby match images for each image, so the batch forming is faster. The metric learning loss functions for these batches push further apart the feature space distance of images that are far away in the ground truth real world positioning, and pull closer together the feature space distance of images that are close by in the ground truth positioning. A high level diagram of the training process can be seen in Figure 3.2.

Multiple experiments were run with varying the training match distance threshold. There was not much difference in performance, as long as the train and test distance thresholds were within "reasonable" values. The reasonability of the values comes from accuracy of ground truth position information, other researchers, as well as how different the images actually are. For example, in a big open field two images taken 10 meters apart might look the same, but indoors 10 meters apart might look very different. The chosen training distance was 5 meters in urban outdoors, and one meter indoors.

For place recognition training and testing, a dataset needs to have:

1. Sensory input data.
2. Corresponding place information for input data.
3. At least two sequences that have overlapping driven routes in the environment. One sequence for the query set and another for the gallery set, so we can test the recall performance.
4. More data for training, if fine-tuning. The driven route for the training set is preferably completely independent of the driven routes for the query and gallery sets. It does not even have to be a sequence as such, but there needs to be enough data below and above the training distance threshold for multiple images. This means there needs to be at least some kind of clusters of data in terms of position. For practical real-life implementations where the goal is to have a functional system and not to run benchmarks, all the available data can be used as the training set.

In the context of this thesis, place recognition requires positional data as ground truth so accuracies can be measured in real-world metric units. To contrast, in categorical approximate place of image recognition the place information can be just an ID describing the general place of the data. For example, the Google Landmarks datasets [53, 82] have images taken of landmarks identified by an ID. While the method of matching and retrieving images is quite similar in both, the dataset requirements are very different for place recognition and categorical image recognition.

The most important dataset used in this work is the Oxford Radar RobotCar dataset [5], which features long overlapping sequences captured with high quality LiDAR. In addition, the Cosy Localization Database (COLD) [58], an indoor dataset with a 2D LiDAR is experimented with. Despite the 2D LiDAR, the COLD dataset is interesting as it has long-term capability via several dynamic sequences from same places.

4.2 Simulated data

Simulated data is interesting, as the lure of easily collectable data from a fully controlled environment with perfect ground truth is very tempting. The downsides are that even well generated simulation rarely matches real-world data, and depending on the area of application, it might be very time consuming and hard to create realistic simulated data.

In Publication I, simulated data from a game environment was experimented with. The environment built from freely available Unity asset store items, was added with the capability to tag environment objects with the wanted segmentation labels. Thus, gathering of image segmentation datasets was possible with ease. The studied semantic segmentation problem is different from place recognition. However, for both, convolutional neural networks are used as the feature extraction pipeline, and input is images. With similar simulation setup, LiDAR place recognition data could be produced with very high quality LiDAR scans and ground truth. That is, however, left as a future research avenue for now. In the publication, it was demonstrated that even cheaply built simulation environments can produce datasets that can improve training results when mixed in with real-world data at a suitable ratio.

4.3 LiDAR 360 degree rotation augmentation with private factory data

Image augmentation shares a similarity to simulated data. In augmentation the original data is altered to produce more data in order to make the algorithm learn to generalize better. Due to the horizontal 360 degree view of the LiDAR sensors typically used, experiments with virtually rotating the sensor to augment data can be conducted. The virtual rotation of the sensor is quickly doable by rolling over the exported pixel image. The pixel columns are moved forward, and the overflow is copied back to the beginning to fill the first columns. All data is preserved, only the image "direction" is moved, simulating a rotation of the sensor itself.

In Publication II, a 360 degree random panoramic rotation and a 180 degree panoramic direction flipping were introduced as augmentation methods specific to exported LiDAR images. In a 360 degree random panoramic rotation the image is randomly rotated between $[0, 360]$ degrees to simulate more viewpoints and situations. The 180 degree panoramic direction flipping only rotated either 0 or 180 degrees, to simulate driving the other way around. The random rotation and direction flipping were experimented with on a 9707 frame high quality private dataset gathered from an industrial factory indoors environment. A sample image from the dataset can be seen in Figure 4.1. The best result in loop detection was achieved with utilizing 360 degree random rotation augmentation.

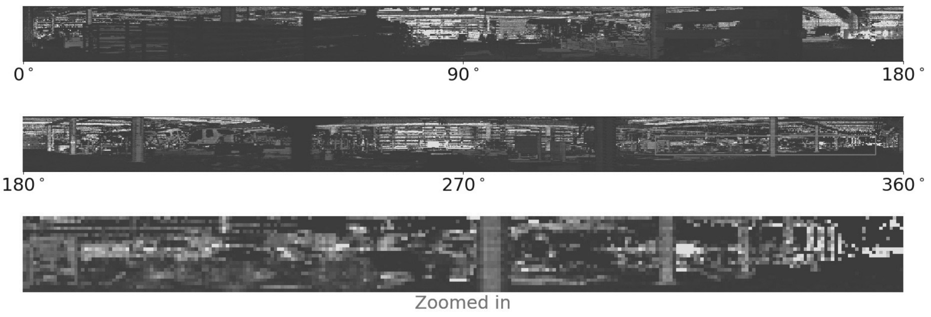


Figure 4.1 © 2021 IEEE. Publication II, Figure 3. A visualized sample from the Ouster LiDAR intensity scan used for route loop detection.

4.4 Oxford Radar RobotCar dataset

The Oxford Radar RobotCar dataset [5] is a newer version of the Oxford RobotCar dataset [49]. The Oxford Radar RobotCar dataset features 32 sequences driven in January 2019 in downtown Oxford. Each sequence follows the same 9 kilometer long route with several loop points. The Radar RobotCar features radar measurements, and more importantly in this context, two LiDAR sensors of decent quality. The dataset is large enough and there is lots of dynamic variability in traffic, sunshine, clouds, and rain. There is however no seasonal variability or extreme weather conditions. An example of Oxford Radar RobotCar data can be seen in Figure 1.1.

The complete list of sensors in the Oxford Radar RobotCar dataset is as follows: On the roof there is a Navtech CTS350-X radar and two Velodyne HDL-32E LiDARs, one on each side. The other sensors are familiar from the older RobotCar dataset: NovAtel SPAN-CPT ALIGN for GPS+INS data, Point Grey Bumblebee XB3 stereo camera pointed forwards, Point Grey Grasshopper2 cameras pointed left, right, and rear. There are also two low quality SICK LMS-151 2D LiDARs. In studying LiDAR place recognition, the interest was mostly on the Velodyne LiDARs and the positional data. The absolute positional data is important as it allows us to compare data from different sequences and test the system performance as accurately as possible. The dataset provided GPS/INS positional data drifts too much for reliable performance testing with distance thresholds less than about 15 meters. The provided radar odometry ground truth is more accurate, but completely relational to the starting position and direction, which are not provided. The GPS/INS data has approximate timestamped pose information that can theoretically be used as the odometry start points, but in practice are inadequate.

The Oxford Radar RobotCar dataset was used as the urban outdoors long-term LiDAR place recognition dataset in Publication III. The environment in Oxford consists of diverse old buildings along narrow streets with parks and trees visible every now and then. It does seem that the dataset offers plenty of diversity and distinguishable visual features, as well as lots of dynamic elements in the form of traffic and weather.

4.4.1 Significance of the ground truth accuracy

The significance of ground truth accuracy and the effect of gallery selection for place recognition evaluation was studied in Publication IV. While the ground truth radar odometry provided with the Oxford Radar RobotCar dataset is higher quality than the provided GPS/INS positioning, the odometry could not immediately be used for place recognition evaluation, as seen in Figure 4.2. The reason for that is that odometry is always starting from the point of origin, $(0, 0)$, while GPS/INS starts from a correct (but possibly inaccurate) global positional point. For place recognition it is critical that the routes are aligned precisely with each other so that one route can be chosen as the query set and another as the gallery set and use the positioning as ground truth. It is impossible to do accurate place recognition benchmarking without precise alignment.

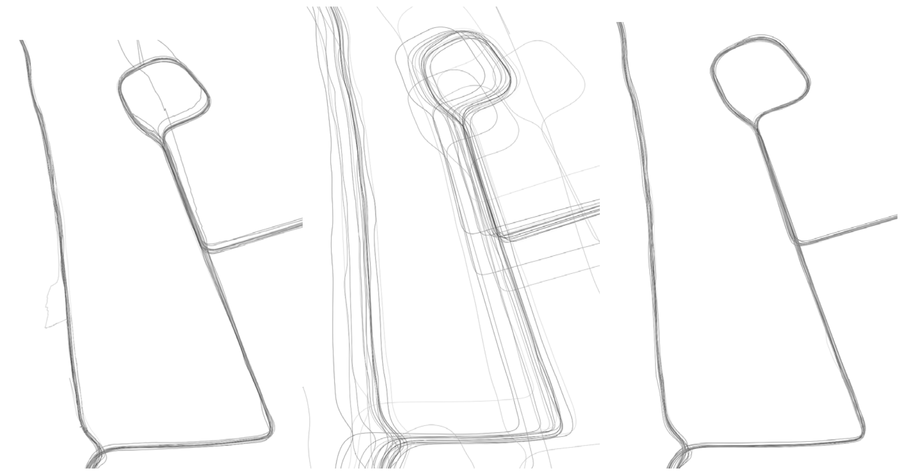


Figure 4.2 Various Oxford Radar RobotCar ground truth sources visualized for all 22 sequences used in the experiments (2 gallery, 2 train and 18 query sequences). "GPS" is omitted as it is practically GPS/INS with less samples and provides almost identical results. In Radar Odometry (fixed) each sequence start pose is manually tuned (translation and rotation). Figure 2 from Publication IV.

The GPS/INS position drifts too much for reliable place recognition benchmarking for thresholds under about 15 meters. Many recent papers use a 5.0 meter threshold for determining pairs, and the GPS/INS drift was beyond that in many cases, giving confusing and erroneous results. To make place recognition benchmarking with the Oxford Radar RobotCar dataset more accurate, the radar odometries were manually aligned for a demonstratedly better testing accuracy. A new evaluation protocol was also proposed, as seen in Table 4.1, for the Oxford Radar RobotCar dataset to encourage more comparable study results.

Table 4.1 Selected place recognition gallery, training and test sequences for benchmarking the Oxford Radar RobotCar place recognition. All sequences are during daylight as the starting times are between 11:46 and 15:20, single traversal takes about 30 minutes and in January the sun sets approximately at 16:50 (4pm50) in Oxford. Publication IV, Table 1.

Name	Data	Time (GMT 24h)	Weather[†]
Gallery 1	Day00 (Jan 10)	Noon (11:46)	Low clouds
Gallery 2	Day00	Afternoon (15:19)	Low clouds
Train 1	Day00	Afternoon (14:02)	Low clouds
Train 2	Day00	Afternoon (14:50)	Low clouds
Query 01	Day01 (Jan 11)	Noon (12:26)	Broken clouds
Query 02	Day01	Afternoon (14:37)	Broken clouds
Query 03	Day01	Afternoon (13:24)	Broken clouds
Query 04	Day04 (Jan 14)	Noon (12:05)	Broken clouds
Query 05	Day04	Afternoon (14:48)	Broken clouds
Query 06	Day04	Afternoon (13:38)	Broken clouds
Query 07	Day05 (Jan 15)	Noon (12:01)	Partly sunny
Query 08	Day05	Afternoon (14:24)	Partly sunny
Query 09	Day05	Afternoon (13:06)	Partly sunny
Query 10	Day06 (Jan 16)	Noon (11:53)	Light train, partly sunny
Query 11	Day06	Afternoon (14:15)	Light train, partly sunny
Query 12	Day06	Afternoon (13:09)	Light train, partly sunny
Query 13	Day07 (Jan 17)	Noon (11:46)	Passing clouds
Query 14	Day07	Afternoon (14:03)	Passing clouds
Query 15	Day07	Noon (12:48)	Passing clouds
Query 16	Day08 (Jan 18)	Noon (12:42)	Partly sunny
Query 17	Day08	Afternoon (15:20)	Partly sunny
Query 18	Day08	Afternoon (14:14)	Partly sunny

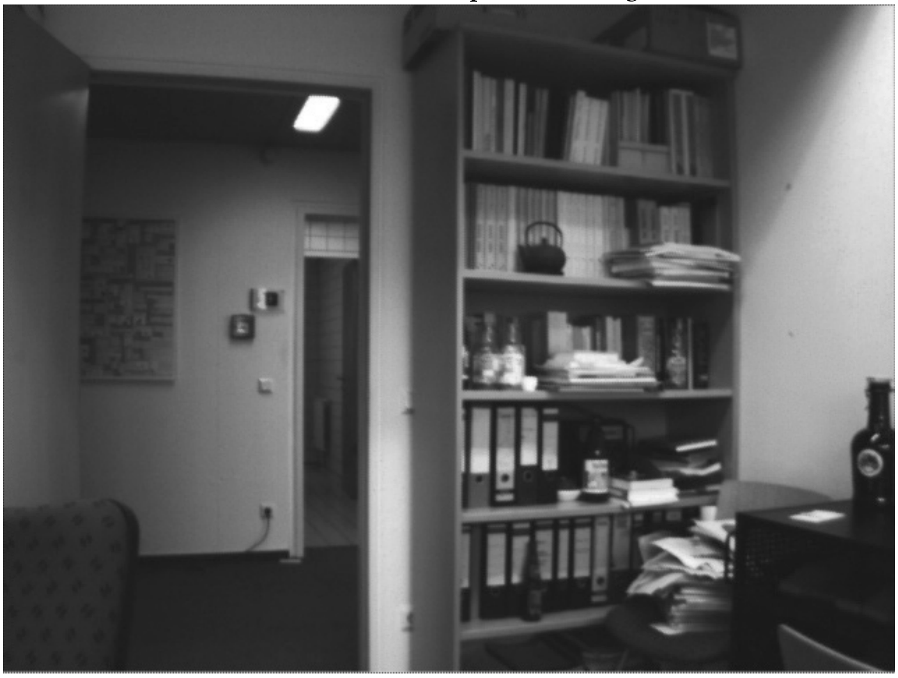
[†] weather conditions obtained from https://www.timeanddate.com

4.5 COLD: COsy Localization Database

The COsy Localization Database (COLD) [58] is a navigation dataset gathered from three different indoor offices: the Autonomous Intelligent Systems Laboratory at the University of Freiburg in Germany, the Visual Cognitive Systems Laboratory at the University of Ljubljana in Slovenia, and the Language Technology Laboratory at the German Research Center for Artificial Intelligence in Saarbrücken in Germany. There are a total of 76 continuous sequences available. The dataset is collected over multiple days, and includes moving objects and people in the busy offices. The lighting and weather conditions are also varied, including sunny day-time, cloudy day-time, and night-time sequences. The dynamic conditions make the COLD dataset a good option for testing place recognition robustness in busy environments. COLD was used in Publication III as the office indoors dataset.

Manually driven robots are used for the capture. The robots at each three location are of different height, but feature the same set of sensors. Odometry, perspective and omnidirectional camera images, as well as laser scanner data are included in the dataset. Two Videre Design MDCS2 cameras are used; one in perspective mode and another in omnidirectional mode. The SICK 2D LiDAR provides a single vertically static 360-degree scan with 361 samples per rotation. The data includes only range information, not reflection intensity. The LiDAR is very low resolution by modern standards, which should be taken into account when considering COLD for LiDAR place recognition experiments. A sample RGB and LiDAR can be seen in Figure 4.3.

COLD dataset sample RGB image:



The corresponding 360-degree 2D laser scan export:

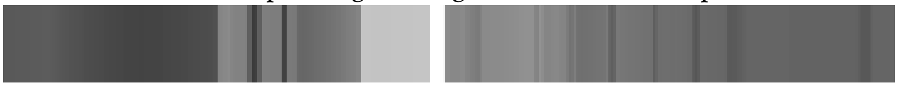


Figure 4.3 © 2021 IEEE. Publication III, Figure 3. Example from the COsy Localization Database (COLD) [58]. The dataset is fully indoors. The greyscale image below is the expanded SICK laser scan spanning 360-degree around the robot (white encodes distances at and above 8 m and black is 0 m). The completely white strip in the middle of the scan is the narrow view into the room on the other side of the corridor that measures 8 meters or more from the sensor. The wider light grey strip on the left side of the white strip is the corridor wall. The rest of the readings are from inside the room and thus closer by.

4.6 Performance evaluation metric

In this thesis and the publications, recall@1 is used consistently with various test thresholds from half a meter to 25 meters, depending on the dataset. Recall@1 is a fairly strict metric as only the best found match is considered at all. A typical place recognition evaluation metric for place recognition is recall@N, with a certain test distance threshold. For example, recall@N with 25 meter test thresholding is used in [4, 3, 78, 77], as well as recall@N and recall@1 with a 5 meter threshold in [6, 37].

To begin an evaluation of the recall@N, first a test is run, giving us a similarity-sorted list of all gallery images per each query image. For each of the Q query images $I_q(Q)$, we then have G gallery images $I_g(G)$ sorted such that the first gallery image $I_g(1)$ is the match that the system is predicting to be the best match for the query, and the last gallery image $I_g(G)$ is the predicted to be the worst possible match for the query.

The real world ground truth position of all the images is known in the dataset, and required for the evaluation. The test distance threshold, τ, is required for determining whether an image pair found by the system is actually considered a correct match or an non-match. In urban outdoors the typical test thresholds are around 5 to 25 meters. In indoors the thresholds are between half a meter to 10 meters.

If the system found k-th gallery image $I_g(k)$ has a ground truth position $p(I_g(k))$ that is within the test threshold distance τ to the j-th query image position $d(I_q(j))$, the pair is considered a successful recognition. If the distance is more than the test threshold, the pair is not a match.

The match function returns 1 if the pair is a match and 0 if the pair is not a match:

$$match(I_q, I_g) = \begin{cases} 1, & \text{if } dist(p(I_q), p(I_g)) \leq \tau \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

For evaluation purposes, only the top N best gallery images are considered per query image. The top-N metric determines whether or not the query was a success. For a query to be successful, the query image needs to have at least one correct match within the best N gallery images. For example, top-1 metric means we only check matches between each query image and the very best gallery image in the ordered list, i.e. the gallery image that the system predicted to be the very best match for the query image. Top-5 means we check if there are matches in any of the 5 best gallery images, and so forth. Top-1 is the most demanding metric.

We can think of top-N as a function, where a success would be represented by 1 and failure would be represented by 0. The sum of the first N match scores would have to be one or more correct matches for the top-N function to return a success:

$$topN(I_q, N) = \begin{cases} 1, & \text{if } \sum_{k=1}^{N} match(I_q, I_g(k)) \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

As we iterate the top-N metric for all the query images in the query set, we get the recall@N metric. Recall@N is the fraction of top-N scores over the whole query set, or in other words, the fraction of successful queries.

In the recall@N function, the top-N scores for each query image are first summed up, and then divided by the total number of query images, giving us a fraction of successful queries:

$$recall@N = \frac{\sum_{j=1}^{Q} topN(I_q(j), N)}{Q} \tag{4.3}$$

The evaluation can be presented by looping over many different N values, or many different test distance threshold (τ) values depending on what seems appropriate for the experiment. In the publications, multiple distance thresholds are looped over with the strictest possible N value, in order to emphasize what can be approximately expected in terms of real-world positioning performance.

5 RESULTS

The results of the research work for this thesis have been published in several conference proceedings. The most important results are summarised here.

5.1 Long-term LiDAR place recognition

In publication III, the effects of weather and sensor modality were studied with an outdoor and an indoor dataset. LiDAR being a competitive sensor modality versus an RGB camera was demonstrated. LiDAR depth map images were also compared to LiDAR intensity map images, and depth maps were found to be more robust to change.

An image retrieval system was modified for place recognition and found to achieve about 80% recall@1 with 5 meter test distance outdoors and 1 meter indoors, as seen in Table 5.1. The fine-tuning of the pre-trained model resulted in 10 – 20% recall improvement. In the experiments, GeM seemed to be the better pooling method. The 2D LiDAR in the indoor dataset clearly demonstrated the worse performance of a cheaper LiDAR.

Table 5.1 © 2021 IEEE. Publication III, Table 1, with method name GeM used instead of CNNRetr. Place recognition recall@1 performance. Comparison of NetVLAD [3] and GeM [61] with only pre-training (no-train), and GeM with fine-tuning (trained). On the Oxford Radar RobotCar [5] and COsy Localization Database [58] datasets.

	Outdoor dataset - RobotCar [49]				Indoor dataset - COLD [58]			
Method	Rec@1-25m	Rec@1-10m	Rec@1-5m	Rec@1-2m	Rec@1-100cm	Rec@1-50cm	Rec@1-25cm	Rec@1-10cm
Query 1 (same day, 2h later)					*Query 1 (sunny)*			
NetVLAD [3] (no train)	0.899	0.867	0.728	0.130	0.838	**0.779**	**0.464**	**0.008**
GeM [60] (no train)	0.926	0.901	0.774	0.134	0.783	0.723	0.439	0.004
GeM [60] (trained)	**0.986**	**0.977**	**0.869**	**0.155**	**0.847**	0.774	0.462	0.004
Query 2 (next day, same time)					*Query 2 (cloudy)*			
NetVLAD [3] (no train)	0.895	0.863	0.762	0.454	**0.230**	<0.000	<0.000	<0.000
GeM [60] (no train)	0.887	0.856	0.758	0.468	0.100	<0.000	<0.000	<0.000
GeM [60] (trained)	**0.993**	**0.984**	**0.902**	**0.544**	0.091	<0.000	<0.000	<0.000
Query 3 (after 6 days, 2h later)					*Query 3 (night)*			
NetVLAD [3] (no train)	0.527	0.449	0.325	0.049	0.264	0.005	<0.000	<0.000
GeM [60] (no train)	0.678	0.608	0.465	0.060	**0.267**	0.005	<0.000	<0.000
GeM [60] (trained)	**0.918**	**0.856**	**0.642**	**0.089**	0.263	0.005	**0.001**	<0.000

For future research avenues, it was proposed that indoor datasets with high quality LiDAR data are needed, and that the interplay between LiDAR depth and intensity, along with RGB should be further studied.

5.2 LiDAR place recognition evaluation with the Oxford Radar RobotCar dataset

In publication IV, the effect of gallery selection for place recognition evaluation was studied. A new evaluation protocol for the Oxford Radar RobotCar dataset was proposed, as seen in Table 4.1. Fixed radar odometry ground truth starting positions were provided, with which a better testing accuracy was demonstrated, as seen in Table 5.2.

Table 5.2 Average top-1 recall rates for various ground truth sources with Oxford Radar RobotCar dataset. Selected results for gallery 2 with training, from publication IV, Table III.

Method	Rec@1-25m	Rec@1-10m	Rec@1-5m	Rec@1-2m
GPS/INS gt	0.921	0.898	0.810	0.334
Radar Odometry gt	0.675	0.498	0.234	0.045
Radar Odometry (fixed) gt	**0.943**	**0.922**	**0.861**	**0.535**

5.3 Loop-closure detection with LiDAR

In publication II, multiple technical implementation variations were studied, and loop-detection on a private dataset was demonstrated. The dataset had 9707 high quality LiDAR intensity scans and the architecture had an average pooling layer. Three different loss functions were tested with nine different backbone networks. A 360 degree random panoramic rotation and 180 degree panoramic direction flipping were introduced as augmentation methods specific to exported LiDAR intensity images. Several typical 2D image augmentation methods were experimented with.

The key take-away is that it is feasible in this case to detect loop points with image retrieval type methods, as seen in Figure 2.1.

5.4 Simulated data

In publication I, it was demonstrated that even inexpensively built simulation environments can produce datasets that improve training results when mixed in with real-world data at a suitable ratio.

The key results were that very complex objects such as trees benefited from simulated data, because the ground truth annotation was more detailed than a human would realistically do. Also, adding simulated data to augment real-world data improved results as long as the amount of simulated data stayed suitable to the specific situation. In two cases, one benefited as long as the amount of simulated data was below 50% of the total amount, and the other up to 71%, as seen in Figure 5.1.

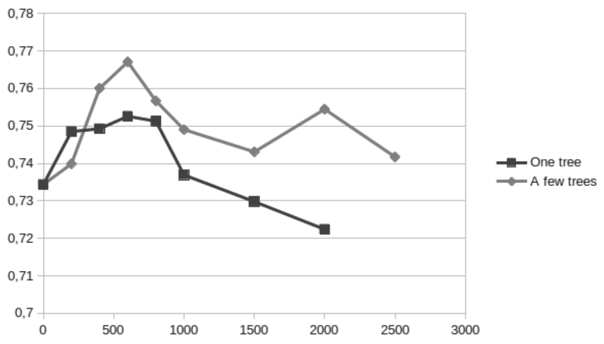


Figure 5.1 © 2019 IEEE. Publication I, Figure 10. Simulated segmentation data results with either one or ten new type of trees in the simulation. The IoU results from merging different amounts of simulated data to 1000 real-world ADE20K images. Clearly adding simulated data is beneficial, but only up to about a 50:50 mix.

The studied semantic segmentation problem is different from place recognition. However, for both tasks, convolutional neural networks are used as the feature extraction pipeline, and input is images. With similar simulation setup, LiDAR place recognition data could be produced with very high quality LiDAR scans and ground truth. That is, however, left as a future research avenue for now.

6 CONCLUSIONS

In this thesis about LiDAR place recognition by image retrieval, the key topic areas were covered in general detail and based on published research. In an overall conclusion, LiDAR place recognition by image retrieval works well. Exported pixel images work well with the convolutional pipeline, and image retrieval yields accurate enough results for place recognition. The place recognition pipeline was also turned into an Android app by a student colleague, demonstrating practical applicability. LiDAR is a sensor well matched to the task in urban outdoor and office indoor environments. The thesis and the related publications did, however, not take into account the great variability of environments, such as for example, forests, caves, shopping malls, and the countryside.

In urban outdoors, a five meter accuracy is possible with about 85 percent recall@1, as measured with the Oxford Radar RobotCar dataset. Indoors, an accuracy of one meter is possible with a similar recall. The system relies on standard, generally well understood components, which is good for future improvements alongside with the overall research and development in the field. For example, an improved backbone network is easy to swap in and take advantage of, and the ImageNet pre-training saves GPU hours and is a surprisingly good starting point for exported LiDAR imagery. Any new pooling method can be put on top of the backbone network.

The first research question of how to apply existing image retrieval methods for place recognition is directly discussed in Chapter 3. More specifically, the overall deep architecture of the place recognition modifications for the image retrieval system (4.1) is discussed, as well as the pooling methods (3.2) and loss functions (3.3). The performance of the method was explored in various ways in publications II, III, and IV. In publication II, we studied several backbone networks and loss functions with the largely related problem of loop-closure detection. In publication III, we evaluated long-term place recognition on two datasets and two pooling methods. In publication IV, we demonstrated the importance of ground truth accuracy in datasets by manually improving the radar odometry for the Oxford Radar RobotCar dataset.

The second research question of how to handle LiDAR in image based methods is directly discussed in Section 2.6.2. In publication III, we concluded LiDAR to be a competitive sensor modality compared to RGB. This demonstrates that exporting LiDAR scans to pixel images for processing in image-based pipelines is a decent way to handle LiDAR data. We also demonstrated the quality difference between a 2D and a 3D LiDAR, and found depth maps to be more robust to long-term changes than intensity maps.

The third research question of how to improve the recall performance is discussed throughout the thesis, but most directly in sections about pooling methods (3.2), loss functions (3.3), vector comparison speedups (2.5.3), simulated data (4.2), 360 degree augmentation (4.3), the significance of ground truth accuracy (4.4.1), and in the section about the metric we use to evaluate performance (4.6). In publication I, we explored the usage of simulated data to improve performance in the semantic segmentation task. In publication II, we explored 360 degree LiDAR augmentation with a private dataset. In publication III, we found the GeM pooling to be a good choice for place recognition.

For real life practical robot navigation systems, place recognition is but a small piece of the software toolbox, and extracting well distinguished feature embeddings for LiDAR sensor data is a part of place recognition. In ongoing research in the same lab the developed methods are used in autonomous robot navigation, which has proved their practical power for real world applications.

REFERENCES

[1] Siddharth Agarwal, Ankit Vora, Gaurav Pandey, Wayne Williams, Helen Kourous, and James McBride. “Ford multi-AV seasonal dataset”. In: *The International Journal of Robotics Research* 39.12 (2020), pp. 1367–1376.

[2] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. “NetVLAD: CNN Architecture for Weakly Supervised Place Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[3] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. “NetVLAD: CNN Architecture for Weakly Supervised Place Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2018).

[4] Relja Arandjelović and Andrew Zisserman. “DisLocation: Scalable descriptor distinctiveness for location recognition”. In: *Asian Conference on Computer Vision*. Springer. 2014, pp. 188–204.

[5] Dan Barnes, Matthew Gadd, Paul Murcutt, Paul Newman, and Ingmar Posner. “The Oxford Radar RobotCar Dataset: A Radar Extension to the Oxford RobotCar Dataset”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Paris, 2020.

[6] Dan Barnes and Ingmar Posner. “Under the radar: Learning to predict robust keypoints for odometry estimation and metric localisation in radar”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 9484–9490.

[7] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. “Speeded-up robust features (SURF)”. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[8] Yoshua Bengio. “Learning deep architectures for AI”. In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[9] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. "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.

[10] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11621–11631.

[11] Bingyi Cao, Andre Araujo, and Jack Sim. "Unifying deep local and global features for image search". In: *European Conference on Computer Vision (ECCV)*. Springer. 2020, pp. 726–743.

[12] Fengkui Cao, Fei Yan, Sen Wang, Yan Zhuang, and Wei Wang. "Season-Invariant and Viewpoint-Tolerant LiDAR Place Recognition in GPS-Denied Environments". In: *IEEE Transactions on Industrial Electronics* 68.1 (2020), pp. 563–574.

[13] Nicholas Carlevaris-Bianco, Arash K Ushani, and Ryan M Eustice. "University of Michigan North Campus long-term vision and lidar dataset". In: *The International Journal of Robotics Research* 35.9 (2016), pp. 1023–1035.

[14] Jack Collier, Stephen Se, and Vinay Kotamraju. "Multi-sensor appearance-based place recognition". In: *2013 International Conference on Computer and Robot Vision*. IEEE. 2013, pp. 128–135.

[15] Mark Cummins. "Highly scalable appearance-only SLAM-FAB-MAP 2.0". In: *Proc. Robotics: Sciences and Systems (RSS), 2009* (2009).

[16] Mark Cummins and Paul Newman. "Appearance-only SLAM at large scale with FAB-MAP 2.0". In: *The International Journal of Robotics Research* 30.9 (2011), pp. 1100–1123.

[17] Mark Cummins and Paul Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance". In: *The International Journal of Robotics Research* 27.6 (2008), pp. 647–665.

[18] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE. 2005, pp. 886–893.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee. 2009, pp. 248–255.

[20] H. Durrant-Whyte and T. Bailey. “Simultaneous Localization and Mapping: Part I”. In: *IEEE Robotics & Automation Magazine* 13.2 (2006).

[21] H. Durrant-Whyte and T. Bailey. “Simultaneous Localization and Mapping: Part II”. In: *IEEE Robotics & Automation Magazine* 13.3 (2006).

[22] Martin A Fischler and Robert C Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[23] Dorian Gálvez-López and Juan D Tardos. “Bags of binary words for fast place recognition in image sequences”. In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1188–1197.

[24] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. “Vision meets robotics: The kitti dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[25] Arren J Glover, William P Maddern, Michael J Milford, and Gordon F Wyeth. “FAB-MAP+ RatSLAM: Appearance-based SLAM for multiple times of day”. In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 3507–3512.

[26] Jiadong Guo, Paulo VK Borges, Chanoh Park, and Abel Gawel. “Local descriptor for robust place recognition using lidar intensity”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1470–1477.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[28] Li He, Xiaolong Wang, and Hong Zhang. "M2DP: A novel 3D point cloud descriptor and its application in loop closure detection". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 231–237.

[29] A. Hermans, L. Beyer, and B. Leibe. "In Defense of the Triplet Loss for Person Re-Identification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017.

[30] Kin Leong Ho and Paul Newman. "Detecting loop closure with scene sequences". In: *International Journal of Computer Vision* 74.3 (2007), pp. 261–286.

[31] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[32] H. Jegou, M. Douze, C. Schmid, and P. Perez. "Aggregating local descriptors into a compact image representation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010.

[33] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. "Complex urban dataset with multi-level sensors from highly diverse urban environments". In: *The International Journal of Robotics Research* 38.6 (2019), pp. 642–657.

[34] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs". In: *arXiv preprint arXiv:1702.08734* (2017).

[35] Yannis Kalantidis and Yannis Avrithis. "Locally optimized product quantization for approximate nearest neighbor search". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 2321–2328.

[36] Giseop Kim and Ayoung Kim. "Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4802–4809.

[37] Giseop Kim, Yeong Sang Park, Younghun Cho, Jinyong Jeong, and Ayoung Kim. “Mulran: Multimodal range dataset for urban place recognition”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6246–6253.

[38] Jacek Komorowski. “Minkloc3d: Point cloud based large-scale place recognition”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 1790–1799.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[40] Mathieu Labbe and François Michaud. “Online global loop closure detection for large-scale multi-session graph-based SLAM”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2014, pp. 2661–2666.

[41] Afshan Latif, Aqsa Rasheed, Umer Sajid, Jameel Ahmed, Nouman Ali, Naeem Iqbal Ratyal, Bushra Zafar, Saadat Hanif Dar, Muhammad Sajid, and Tehmina Khalil. “Content-based image retrieval and feature extraction: a comprehensive review”. In: *Mathematical Problems in Engineering* 2019 (2019).

[42] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), p. 436.

[43] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[44] Lin Li, Xin Kong, Xiangrui Zhao, Tianxin Huang, Wanlong Li, Feng Wen, Hongbo Zhang, and Yong Liu. “Ssc: Semantic scan context for large-scale place recognition”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 2092–2099.

[45] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[46] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford. “Visual Place Recognition: A Survey”. In: *IEEE Transactions on Robotics* 32.1 (2016), pp. 1–19. doi: 10.1109/TRO.2015.2496823.

[47] Lun Luo, Siyuan Cao, Bin Han, Hui-liang Shen, and Junwei Li. "BVMatch: Lidar-Based Place Recognition Using Bird's-Eye View Images". In: *IEEE Robotics and Automation Letters* (2021).

[48] Kirk MacTavish and Timothy D Barfoot. "Towards hierarchical place recognition for long-term autonomy". In: *ICRA Workshop on Visual Place Recognition in Changing Environments*. 2014, pp. 1–6.

[49] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. "1 Year, 1000km: The Oxford RobotCar Dataset". In: *The International Journal of Robotics Research (IJRR)* 36.1 (2017), pp. 3–15.

[50] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[51] Carlo Masone and Barbara Caputo. "A survey on deep visual place recognition". In: *IEEE Access* 9 (2021), pp. 19516–19547.

[52] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 922–928.

[53] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. "Large-scale image retrieval with attentive deep local features". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3456–3465.

[54] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. "Deep metric learning via lifted structured feature embedding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4004–4012.

[55] Gaurav Pandey, James R McBride, and Ryan M Eustice. "Ford campus vision and lidar data set". In: *The International Journal of Robotics Research* 30.13 (2011), pp. 1543–1552.

[56] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in PyTorch". In: *Advances in Neural Information Processing Systems Workshop*. 2017.

[57] N. Pion, M. Humenberger, G. Csurka, Y. Cabon, and T. Sattler. "Benchmarking Image Retrieval for Visual Localization". In: *International Conference on 3D Vision (3DV)*. 2020.

[58] Andrzej Pronobis and Barbara Caputo. "COLD: COsy Localization Database". In: *The International Journal of Robotics Research (IJRR)* 28.5 (2009).

[59] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 652–660.

[60] F. Radenović, G. Tolias, and O. Chum. "CNN Image Retrieval Learns from BoW: Unsupervised Fine-Tuning with Hard Examples". In: *European Conference on Computer Vision (ECCV)*. 2016.

[61] Filip Radenović, Giorgos Tolias, and Ondřej Chum. "Fine-tuning CNN Image Retrieval with No Human Annotation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2018).

[62] M. Ramezani, Y. Wang, M. Camurri, D. Wisth, M. Mattamala, and M. Fallon. "The Newer College Dataset: Handheld LiDAR, Inertial and Vision with Ground Truth". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[63] Dario Lodi Rizzini, Francesco Galasso, and Stefano Caselli. "Geometric relation distribution for place recognition". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 523–529.

[64] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[65] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. "Image classification with the fisher vector: Theory and practice". In: *International Journal of Computer Vision* 105.3 (2013), pp. 222–245.

[66] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2018, pp. 4510–4520.

[67] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. "From Coarse to Fine: Robust Hierarchical Localization at Large Scale". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2019.

[68] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "SuperGlue: Learning Feature Matching with Graph Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2020.

[69] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 815–823.

[70] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[71] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[72] B. Steder, G. Grisetti, and W. Burgard. "Robust Place Recognition for 3D Range Data based on Point Features". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).* 2010.

[73] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2020, pp. 2446–2454.

[74] Niko Sünderhauf, Peer Neubert, and Peter Protzel. "Are we there yet? Challenging SeqSLAM on a 3000 km journey across all four seasons". In: *Proc. of workshop on long-term autonomy, IEEE international conference on robotics and automation (Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)).* 2013, p. 2013.

[75] Babak Taati, Michel Bondy, Piotr Jasiobedzki, and Michael Greenspan. "Variable dimensional local shape descriptors for object recognition in range data". In: *2007 IEEE 11th International Conference on Computer Vision*. IEEE. 2007, pp. 1–8.

[76] Mingxing Tan and Quoc V Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *arXiv preprint arXiv:1905.11946* (2019).

[77] Akihiko Torii, Relja Arandjelovic, Josef Sivic, Masatoshi Okutomi, and Tomas Pajdla. "24/7 place recognition by view synthesis". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 1808–1817.

[78] Akihiko Torii, Josef Sivic, Tomas Pajdla, and Masatoshi Okutomi. "Visual place recognition with repetitive structures". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2013, pp. 883–890.

[79] Mikaela Angelina Uy and Gim Hee Lee. "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2018, pp. 4470–4479.

[80] F. Warburg, S. Hauberg, M. Lopex-Antequera, P. Gargallo, Y. Kuang, and J. Civera. "Mapillary Street-Level Sequences: A Dataset for Lifelong Place Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2020.

[81] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification." In: *Journal of Machine Learning Research* 10.2 (2009).

[82] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. “Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 2575–2584.

[83] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. “3d shapenets: A deep representation for volumetric shapes”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920.

[84] Shaorong Xie, Chao Pan, Yaxin Peng, Ke Liu, and Shihui Ying. “Large-Scale Place Recognition Based on Camera-LiDAR Fused Descriptor”. In: *Sensors* 20.10 (2020), p. 2870.

[85] Huan Yin, Xuecheng Xu, Yue Wang, and Rong Xiong. “Radar-to-Lidar: Heterogeneous Place Recognition via Joint Learning”. In: *Frontiers in Robotics and AI* 8 (2021), p. 101.

[86] Peng Yin, Lingyun Xu, Zhe Liu, Lu Li, Hadi Salman, Yuqing He, Weiliang Xu, Hesheng Wang, and Howie Choset. “Stabilize an unsupervised feature learning for LiDAR-based place recognition”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1162–1167.

[87] Kamil Zywanowski, Adam Andrzej Banaszczyk, Michal Ryszard Nowicki, and Jacek Komorowski. “MinkLoc3D-SI: 3D LiDAR place recognition with sparse convolutions, spherical coordinates, and intensity”. In: *IEEE Robotics and Automation Letters* (2021).

PUBLICATIONS

PUBLICATION
I

Semantic segmentation with inexpensive simulated data

Jukka Peltomäki, Mengyang Chen, and Heikki Huttunen

In: *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. 2019, pp. 1–6

Semantic segmentation with inexpensive simulated data

*Jukka Peltomäki, *Mengyang Chen, and *Heikki Huttunen
*Tampere University, Tampere, Finland

***Abstract*—This paper studies the benefits of adding inexpensively gathered simulated data to improve the training of semantic segmentation models. We introduce our implementation to gather simulated datasets with minimal effort. In our implementation, we utilize a commonly available game engine (Unity) and auxilliary graphical assets to assemble an environment to generate simulated data inexpensively. We also demonstrate that even the usage of spartan simulated data mixed with real-life data can increase the performance of the trained model slightly, given that the ratio of simulated data is suitable for the datasets.**

I. Introduction

Semantic segmentation is one of the emerging fields in computer vision, and deep learning is a most promising approach for it. [1] [2] [3] [4]

However, the challenge with semantic segmentation is the requirement of large volumes of annotated examples. In semantic segmentation, this is a particularly significant challenge, since the required pixel-wise annotation is clearly more labor-intensive than for example bounding-box annotation.

One common solution is to use synthetic images, where the ground truth segmentation maps are immediately available. Several approaches have been proposed for synthesizing realistic views: Building custom simulated environments for specific purpose [5] [6] [7], copy-pasting annotated objects to generate more data [8], and extracting ground truth from popular video games [9].

However, the generation of a complete high-quality realistic synthetic data generator is resource-intensive in itself, and may not always be within the reach of an engineer that quickly needs to generate large volumes of synthetic data for a particular application case. Therefore, we will study how well general-purpose game development platforms are suited for image synthesis. With this, we mean commonly available platforms and their freely available non-tailored graphical assets (*e.g.,* stock models of tree, person, vehicle, etc.). With these components, an experienced developer can construct an environment for data collection within a few days or even hours.

In this paper, we will study these questions with the use case of tree segmentation aided by synthetic data from Unity game environment. We selected trees as objects specifically since it represents an example where human annotations tend to be inaccurate: Humans only annotate the outline of a tree, while a synthetic dataset contains all the small details annotated as well.

The differences in annotation detail are highlighted in figure 1, which illustrates the result of semantic segmentation of trees when trained using human-annotated data (middle) and when using synthetic data (bottom). One can clearly see that the use of synthetic data makes the model learn significantly more detailed segmentation maps.

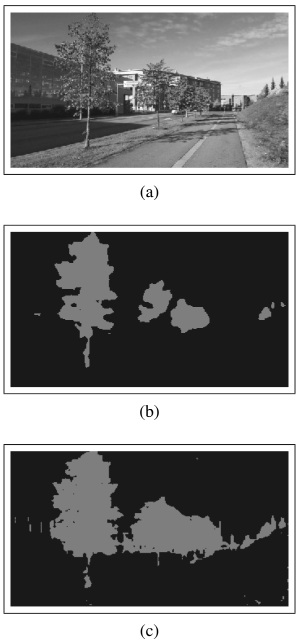


Fig. 1. Semantic segmentation results. The original image (a) segmented with fully human annotated data (b) and with additional simulated data (c). Simulation can achieve better detail accuracy in pixel-wise annotation situations.

II. Data generation and a game environment

Modern game engines and development environments are powerful tools for creating complex and visually compelling virtual environments with relative ease. The availability of ready-to-use artistic assets, such as textured 3D-models, animations, sounds, visual effects, and behavioral logic brings down the skill and time required to put together decent environments. These advances in the field of game development are excellent news to a researcher wanting to create different environments for a variety of tasks, such as reinforcement

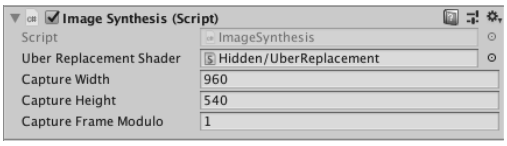


Image Synthesis (Script)

Script	ImageSynthesis
Uber Replacement Shader	Hidden/UberReplacement
Capture Width	960
Capture Height	540
Capture Frame Modulo	1

Fig. 2. ImageSynthesis component in Unity is attached to a Camera object. You can select the resolution for saving images in, and the frequency of saved frames.

learning, stereo vision, visual flow, object detection, semantic segmentation, and many more.

The goal for our data generation experiment was to create data beneficial to the training of semantic segmentation algorithms with minimal cost. To that end, we took a freely available pre-made 3D environment and labeled the 3D objects in the scene, and that was it, as far as the art of crafting the environment is concerned. We did not do any custom art or even layout, as we wanted to specifically see how a very generic, inexpensively gathered dataset could help the training process.

Any game engine will do, but we chose to use Unity for our experiments for several reasons: We were already familiar with it, it is beginner friendly, there is a healthy community for support and all kinds of add-ons and assets, and there are ready made AI-helping components available. The objects in our environment were gathered freely from the Unity Asset Store. The creation of the segmentation maps was done using the freely available Unity package ML-ImageSynthesis with a few tweaks.

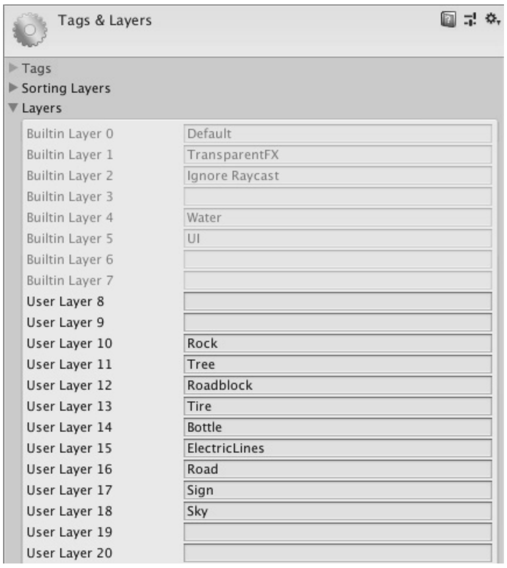
A. Necessary components for gathering data

To gather semantic segmentation data from a game engine, there needs to be a way to export normal RGB camera images with the corresponding ground truth segmentation images. The implementation of this is game engine specific.

In our case, the ML-ImageSynthesis offers the required components to gather semantic segmentation, instance segmentation, depth maps, and optical flow from Unity. We only needed the semantic segmentation. From the plugin, we utilized the camera segmentation renderer component, and a segmentation shader. We also created a script for controlling how often to export renderings. One export in our case consists of two images, one normal RGB camera image and one pixel-wise semantic object segmentation map image. It is also possible to set the wanted aspect ratio and resolution, so you can, for instance, match the real dataset resolution for consistency. In figure 2 the ImageSynthesis component interface is seen as shown in Unity.

B. Environment design

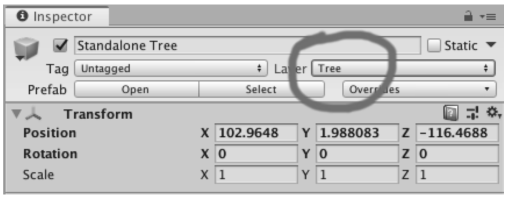
On the artistic side, the scene in the game engine can consist of anything you have assembled and annotated. The annotation process is also game engine specific. The general idea is to have an easy way to tell the game engine which objects belong



Tags & Layers

Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	
User Layer 9	
User Layer 10	Rock
User Layer 11	Tree
User Layer 12	Roadblock
User Layer 13	Tire
User Layer 14	Bottle
User Layer 15	ElectricLines
User Layer 16	Road
User Layer 17	Sign
User Layer 18	Sky
User Layer 19	
User Layer 20	

Fig. 3. Unity game engine has the concept of layers, which we utilized to serve as our semantic class assignment method. Here is a list of layers in our environment. We have added several classification layers, such as the layer 11 for trees, which we are using in our experiments in this paper.



Standalone Tree

Tag	Untagged
Layer	Tree

Transform

	X	Y	Z
Position	102.9648	1.988083	-116.4688
Rotation	0	0	0
Scale	1	1	1

Fig. 4. Assigning objects to belong to a certain semantic class is done by selecting the wanted prefab or an instance object and selecting the layer from the drop-down menu. It is usually beneficial to apply the layer selection to the child objects as well.

to which classes. This could be done, for instance, utilizing different scene hierarchy trees, layers, tagging, naming conventions, or implementing new components to handle the annotation.

For our situation in Unity, we used assigning objects to certain layers to mean different object classes for training. First we added some new layers, as seen in figure 3. Then we assigned the wanted objects to be on those layers, as seen in figure 4. It is useful to note that we do not need to go over our whole environment to assign every instance of a tree to the tree layer, because we are utilizing the prefab-structure to represent our trees. Prefabs are kind of prototype objects in Unity, and might be thought of similarly as classes in programming languages.

There are some caveats when using general 3D environments and objects for gathering datasets for machine learning.

For one, it is up to the original 3D artist creating the environment and the objects to decide how to split the models and textures to match the semantics of the object they are modeling. For example, if you want to teach your segmentation model to instantiate windows and doors from buildings, they need to be technically separate 3D models in the scene, otherwise getting the ground truth for them does not work without additional manual work. In games, developers often like to minimize the polygon count and the texture space to increase real time performance, so things are baked together as much as possible. This means that an otherwise suitable scene may not necessarily offer the technical object level separation to be used as is. Of course, adding invisible 3D pseudo-objects on top of the semantic objects in the models allows you to annotate the 3D objects however you wish, which can still be a lot less effort than labeling the resulting 2D images.

The artistic style used might also be antithetical to real life transfer learning, as in being cartoony or artistically flamboyant.

It might also be hard to find the exact objects or scenes you'd prefer, if you have specific needs. The forests near the Arctic Circle in April look quite different to the stylized lush green forests modeled to please the eye of the global gamer.

Also, simply the texture quality or the polygon count might simply be too low for your use case.

There might also be some technical hurdles, such as in our case, the handling of Unity terrain components. Unity apparently does have a rendering system for segmentating based on terrain textures, but we were not able to get it working properly at the time of writing.

C. Automated data collection

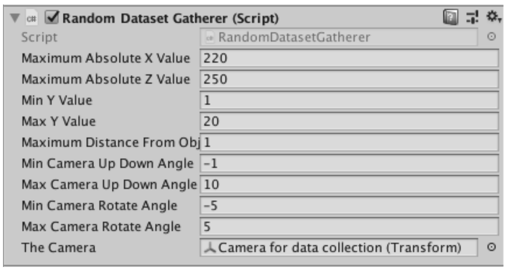
The quick and easy way to gather datasets is to do it automatically. In automated data collection the collecting camera is put into a semi-random position, the view and the segmentation map are rendered, after which a new semi-random position is immediately chosen. This way the gathering of the data is not limited by human guidance or certain camera orientations.

The challenge in automated data collection is the need for some kind of a heuristic to weed out the useless samples, e.g. many images of just sky, close up ground texture, or erroneous views due to the camera being inside of 3D models or beneath the ground.

Our method to combat unusable images was to implement a few very simple rules, which seemed to work well enough on our example environment:

- The camera has to always be at a certain distance above the ground.
- The camera always has to be at least a certain distance away from any 3D model.
- The rotation (α, β, γ) and the position (x, y, z) of the camera has to be within certain minimum and maximum limits.

In our case, these parameters can be tuned to fit the environment and the type of data to be gathered. For our data gathering purposes, we left these parameters quite laxed, serving mostly to make all the randomized views be valid, and limit the amount of pure sky or ground samples. The data gathering component settings can be seen in figure 5.



Random Dataset Gatherer (Script)

Script	RandomDatasetGatherer
Maximum Absolute X Value	220
Maximum Absolute Z Value	250
Min Y Value	1
Max Y Value	20
Maximum Distance From Obj	1
Min Camera Up Down Angle	-1
Max Camera Up Down Angle	10
Min Camera Rotate Angle	-5
Max Camera Rotate Angle	5
The Camera	Camera for data collection (Transform)

Fig. 5. Our random dataset gathering component as shown in the Unity inspector. You can select some parameters to restrict the selection of camera parameters for the automatic gathering. In addition to these, the rules about being above ground and being of certain distance to any 3D models are always enforced.

We did not implement any automated heuristics to maximize the diversity of the dataset, but it would be an interesting further avenue for research.

D. Sequential data collection by driving around

For some learning tasks, such as optical flow and tracking, it might be essential to gather sequential video data mimicking real world spatial progression while moving logically and smoothly. It is also required for collecting test sets to test segmentation consistency and smoothness between frames.

To facilitate this, we added a four wheeled vehicle that can be driven around in the scene similar to an arcade driving game. It can be driven around with cameras attached to it, that save images at constant intervals. Driving manually also allows for the researcher to focus on the part of the environment they want.

The down sides of collecting data via driving around are that it is manual, as we have not implemented automated driving, so it is slow, costly, and the data collected tends to not be as diverse as it could be. The constant data saving rate automatically concentrates the images to be more from the slow parts of the drivable environment, and less from the parts where the scenery changes faster, e.g. a long straight piece of road. The layout of the environment also usually attracts to driving mostly around certain areas (i.e. roads). The cameras are always at a pretty similar orientation and height, and no shots are taken from high up, close up, or from surprising angles.

III. Experimental results

The purpose of our experiment is to see if, and by how much, a very inexpensively collected simulated dataset can improve segmentation performance on a real life dataset.

We also experiment with different ratios of real to simulate data in the training phase. Our intuition is that increasing the amount of simulated data to a constant number of real data would increase the model performance as measured in intersection over union (IoU) and then eventually plateau and finally start to decrease as the amount of simulated data vastly overpowers the amount of real data.

A. Data

In this experiment we specifically studied the tree class, as it is a class where a human doing pixel-wise segmentation is bound to draw the general outline of the tree, and the ground truthed data from the simulated environment correctly assigns every pixel. Therefore, all the images that did not have a tree class were removed from the sets.

The simulate dataset was collected using our custom Unity game engine environment. One thousand images were collected by manually driving around on the vehicle inside the game environment, and six thousand images were collected fully automatically. The images and the segmentation maps were all 960 by 540 pixels. Our environment is extremely spartan for this task, as there is just a single 3D model of a tree scattered around the environment.

For the real life dataset, we used ADE20K [10] from MIT CSAIL. It contains non-sequential photographs of indoor and outdoor scenes, and is human annotated for semantic segmentation. The photographs are of different resolutions. There are over 7300 training images with trees in them. We used the ADE20K validation set purely as our test set, and it was never involved in any of the training or tuning processes. The test set was also cut down to the 700+ images that contained trees, and turned into single class.

The post-processing of the segmentation data for both datasets had three steps:

1) Include only images and segmentation maps where trees were visible at all.
2) Remove other segmentations from the segmentation maps, and leave only the tree class and the not-a-tree class.
3) Convert segmentation map images from normal color RGB to 8-bit single channel grayscale images.

Forming the actual datasets for our different test runs, we chose images from the same randomized sequences for all the set sizes. This means that a bigger dataset always includes all the images chosen for the smaller dataset, too.

The noted size of the datasets is the amount of training images. 20% more was used as a validation set, and chosen from ADE20K and/or the simulated dataset in the same ratio as the training images.

B. Training with simulated data only

DataSet	IoU
Simulated tree data	**0.53311348**
ADE20K tree data	**0.73292613**

TABLE I
EVALUATING ON THE ADE20K TREE CLASS VALIDATION SET, WE CAN SEE THAT SIMULATED DATA BY ITSELF IS PERFORMING BADLY. AS EXPECTED, THE ADE20K TRAINING DATA IS A LOT BETTER FOR THE ADE20K EVALUATION SET.

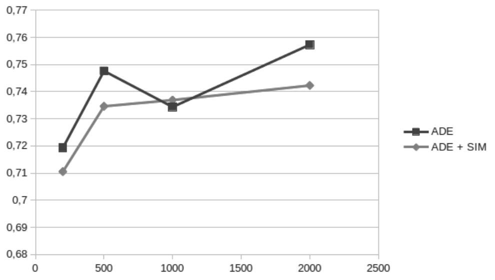


Fig. 6. Results from merging ADE20K dataset with inexpensive simulated data. With 1:1 mix ratio the IoU results get worse, when evaluated on the ADE20K annotated test set.

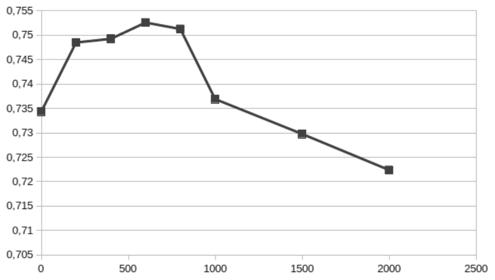


Fig. 7. The IoU results from merging different amounts of simulated data to 1000 ADE20K images. A little bit of simulated data seems to help, while a lot seems to hurt.

To test how the simulated data generalizes to the ADE20K data set, we trained a model on the simulated data only, and evaluated it on the ADE20K trees-only test set. As seen in table I, the performance is pretty quite bad when trained only on the simulated dataset, as indicated by the 53.3% IoU.

DeepLab V3+ [11] was used to the experiments. We randomly chose 2000 images from both sets for training. The independent test set was 730 images. We trained for 50000 steps.

Clearly, our inexpensively simulated dataset with just one 3D model of a low poly tree is not enough to generalize to a proper dataset by itself. Next we move on to see how much of a benefit it still could be.

C. Training with a mixture of real and simulated data

A more realistic scenario is one where we have a small annotated training set from the real world, and wish to append that with synthetic data from a similar scene.

We ran an experiment where we appended our simulated dataset to ADE20K, and compared the results to training only on ADE20K. The appending was done in a one-to-one ratio. We expected the appended set always to be better, but the results were discouraging, as seen in figure 6. It seems that the simulated data mostly just decreases the IoU, except for the peculiar dip when training with 1000 images.

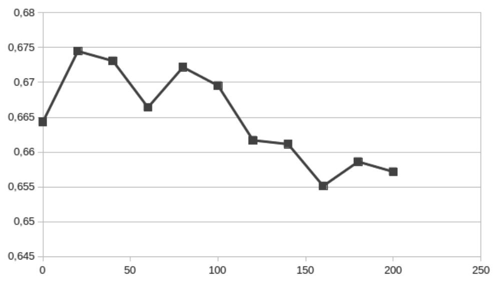


Fig. 8. The IoU results from merging different amounts of simulated data to 100 ADE20K images. These results were averaged from five differently seeded dataset selections, and trained for 30k steps.

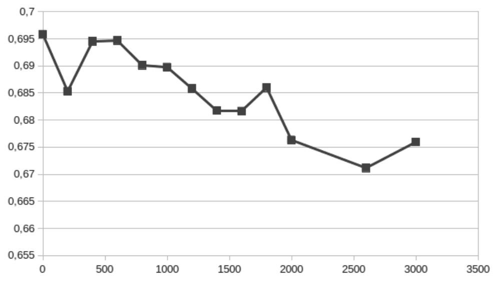


Fig. 9. The IoU results from merging different amounts of simulated data to 2000 ADE20K images. Average of 5 runs for 30k steps.

To see what the optimal ratio of simulated data should be, we took 1000 ADE20K images, and tried adding different amounts of simulated images. The results, as seen in figure 7, were interesting. With the amount of simulated images being less than the amount of real images, the results were better than without simulated images. When the amount of simulated images exceeded the amount of real images, the models performed worse than without any simulated data.

These models were trained for 100000 steps.

We noticed that there was enough randomness in the results depending on the randomly chosen sample images, which obfuscated the process of reliable conclusion drawing. To achieve better stability, we decided to train our other experiments with 5 differently seeded random sequences for the sample selection.

To further explore the best ratio of real to simulated images, we took 100 ADE20K images and added simulated images in 20 image intervals, and trained to 30000 steps with five different seeds. The results averaged over the five random sample sequences are shown in figure 8, and have the same take-away message as in the previous experiment with 1000 real images.

With 2000 ADE20K images and a varying amount of simulated images, as seen in figure 9, it seems that 2000 real images is good enough that additional inexpensive simulated data seems only to degrade the learning process. It would seem there is no need to experiment with higher counts of real images without first adding variance to the game environment.

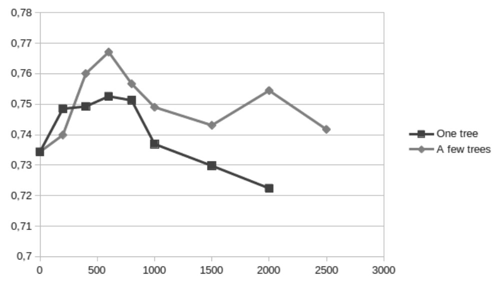


Fig. 10. Results of adding ten more tree models to the game scene, compared to the original one model version. The IoU results from merging different amounts of simulated data to 1000 ADE20K images. 100k training steps.

To see if adding more 3D models of trees to the game scene would help the training, we downloaded 10 more tree models and casually added them all over our game scene. The resulting dataset was used to train with 1000 ADE20K images in different mix ratios for 100k steps. The results are better than with the other simulated dataset, and the mix ratio curve is similarly shaped, as seen in figure 10. With this dataset, the training also seems to tolerate better the higher ratio of simulated images. The performance of the segmentation model is better even with more than half the pictures being simulated than without any simulated data.

In conclusion, it could be said that adding inexpensively gathered data to boost semantic segmentation training is not straight-forwardly the more the better. In our case we only had one or ten 3D tree models in multiple clusters and seen from different angles. This resulted in a very slightly increased IoU when the share of simulated images was suitable for the datasets, meaning less than half the amount of real images. With a bigger mix ratio of simulated images, the IoU dropped with about the same slight amount.

The effect of the simulated data to the IoU is overall low. Our game environment is very spartan, so these results are expected to be somewhere close to the lower limit of what any inexpensive dataset could reach, a sort of a base-line for inexpensive data, if you will. The most logical next research step would be to test with added scene variety, especially adding some variance to the trees, and see how the results scale.

Also, the lower than expected IoU boost from simulated images might have to do with the different way the annotations are drawn by human annotators as opposed to how they are generated by a graphics engine. In our experiments, all the images in the test set were from the human annotated ADE20K dataset. This means that all the test targets were quite low poly blobs drawn over the trees. The simulated data however is much more detailed, as the game engine produces true pixel-wise segmentation of the 3D model. This means adding simulated images to the training set makes the model attempt to segment in a more detailed manner, such as seen in figure 1. This increased attention to detail might cause worse IoU performance when tested against the ADE20K dataset.

All the experiments were ran on Google DeeplabV3+ [11] with an Xception-65 [12] backbone. We always fine-tuned an ImageNet [13] pre-trained model from the Deeplab Model Zoo. The images were squashed to 513x513 resolution.

IV. Conclusion

In this paper we introduced an inexpensive way to gather simulated data with pixel-wise ground truth semantic segmentations. Our results indicate that while using only inexpensive simulated data is nowhere near the level required for training models for real life data, using even very spartan simulated data to augment a real life dataset is a way to slightly improve the model accuracy. The augmentation needs to take into account the quality of the simulated data, and adjust the mix ratio accordingly. For our inexpensively gathered dataset, the best amount of simulated data seemed to be somewhere between 0% and 80% of the amount of real data.

For future work, it would be interesting to explore more thoroughly how the model performance scales in reaction to added scene variance. It would also be interesting to see how well a render randomizer made specifically to improve domain transfer (such as OpenAI Remote Rendering Backend (ORRB) [14]) would as work with our inexpensive data gathering setup.

References

[1] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[3] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 11–19.

[4] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *arXiv preprint arXiv:1802.02611*, 2018.

[5] A. Lehmussola, P. Ruusuvuori, J. Selinummi, H. Huttunen, and O. Yli-Harja, "Computational framework for simulating fluorescence microscope images with cell populations," *IEEE transactions on medical imaging*, vol. 26, no. 7, pp. 1010–1016, 2007.

[6] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[8] M. Wrenninge and J. Unger, "Synscapes: A photorealistic synthetic dataset for street scene parsing," *arXiv preprint arXiv:1810.08705*, 2018.

[9] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision*. Springer, 2016, pp. 102–118.

[10] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[11] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[12] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[14] L. Maciek Chociej, Peter Welinder, "Orrb: Openai remote rendering backend," in *eprint arXiv*, June 2019. [Online]. Available: https://arxiv.org/abs/1906.11633

PUBLICATION
II

Loop-closure detection by LiDAR scan re-identification

Jukka Peltomäki, Xingyang Ni, Jussi Puura, Joni-Kristian Kämäräinen, and Heikki Huttunen

In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 9107–9114

Loop-closure detection by LiDAR scan re-identification

Jukka Peltomäki*, Xingyang Ni*, Jussi Puura†, Joni-Kristian Kämäräinen*, and Heikki Huttunen*
*Tampere University, Finland, †Sandvik Mining and Construction Ltd

***Abstract*—In this work, loop-closure detection from LiDAR scans is defined as an image re-identification problem. Re-identification is performed by computing Euclidean distances of a query scan to a gallery set of previous scans. The distances are computed in a feature embedding space where the scans are mapped by a convolutional neural network (CNN). The network is trained using the triplet loss training strategy. In our experiments we compare different backbone networks, variants of the triplet loss and generic and LiDAR specific data augmentation techniques. With a realistic indoor dataset the best architecture obtains the mean average precision (mAP) above 0.94.**

I. Introduction

Loop-closure is an important sub-problem in robot navigation and mapping since visiting the same location again allows to reduce the map and location uncertainties. This is particularly important for the task of Simultaneous Localization and Mapping (SLAM) where the robot simultaneously builds a map and estimates its location on the map [1]. SLAM is based on *incremental localization* where current location is estimated from the last known global pose [2] and uncertainties accumulate during the process. When the loop-closure is detected the uncertainties can be assigned small values in these locations and propagated to the nearby locations. Visual loop-closure is particularly important indoors where global positioning system (GPS) is not available. Vision-based loop-closure detection is performed by comparing the image of the current location to the images captured from previous locations. In this sense, vision-based loop-closure detection comes down to an online image retrieval task.

A popular method for loop-closure detection is the visual Bag-of-Words (BoW) [3], [4], [5]. The BoW loop-closure detection has been shown to work for large scale data [6] and it has been combined with depth data [7]. However, following the recent trend in computer vision the more recent methods are based on convolutional neural network (CNN) embedding where the feature layers provide a feature vector for image matching. The re-identification architectures adopt the image recognition [8], [9] or the autoencoder structure [10], [11]. Xia et al. provide a comparison of various approaches in [12].

In this work, the type of input data differs substantially from the previous works where conventional RGB images are used. Our input is panoramic intensity images obtained from the intensity channel of a high quality Ouster OS1 LiDAR sensor (Figure 1). The intensity images are of particularly low resolution (64 by 2048 pixels), but are invariant to many imaging distortions such as lighting and shadows since the intensity correlates with material properties. LiDAR scan results to a $360^\circ \times 33^\circ$ view angle image which is suitable for navigation purposes of heavy machinery in indoor work sites and mines. The image matching for loop-closure detection is cast as a re-identification problem. Image re-identification has been successfully used in vehicle recognition [13], [14] and face recognition [15], [16], [17].

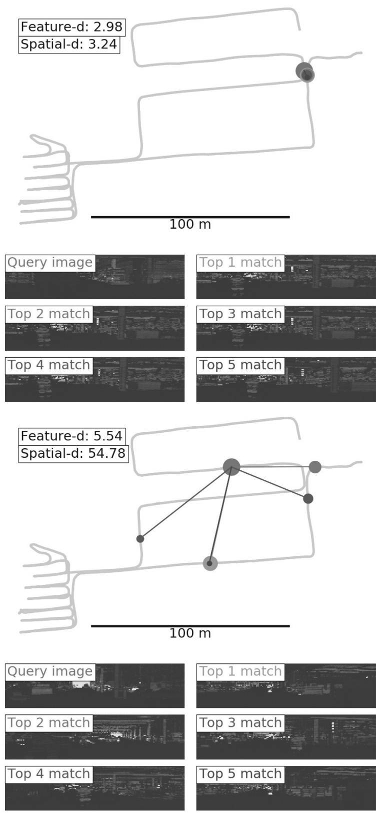


Fig. 1. Examples of loop closure (top) and non-loop closure locations (bottom) where our method effectively detects the loop closure. In the loop-closure location the best five matches (red points) are all near the query spatial location (green point) and within a small feature distance while in the non-loop-closure location the feature distances are large and the matches are found in random spatial locations.

Contributions – The main contributions of this work are:

- We propose feature space embedding for distance based matching of panoramic LiDAR intensity scans. Embedding is computed using a convolutional neural network (CNN) architecture trained using the triplet loss training strategy.
- We provide an experimental comparison of backbone networks, variants of the triplet loss and generic and LiDAR scan specific data augmentation techniques with a realistic data collected from the path of 1,026 meters in industrial environment.

II. Related work

Ability to navigate in known or unknown environments is an essential skills and research challenge in mobile robotics. There are various sensor modalities available, such as sonars, but substantial efforts have been dedicated to vision based navigation since 1970's [2]. Navigation can be sub-divided to map-based navigation and map building, but with the help of spatial uncertainty modeling [18], [19] the simultaneous localization and mapping (SLAM) that combines the two subtasks has become an important technique [1]. A critical task in SLAM is to correctly associate observations of landmarks (locations) with landmarks held in the map. Incorrect association can lead to catastrophic failure of the SLAM algorithm, but successful association helps to reduce uncertainties. Data association is particularly important when a vehicle returns to a previously mapped region after a long excursion, the so-called *loop-closure problem* [20].

Vision-based loop-closure detection – The loop-closure problem can be divided to *loop-closure detection* and *loop closing* where the first refers to detection whether the current observation is from a previously visited location or not and the second to data association where the map and location uncertainties are updated. In our work we focus on the loop-closure detection only. One of the first vision-based SLAM method was introduced by Cummins and Newman [4]. They introduced the Fast Appearance-based Mapping (FAB-MAP) algorithm that was inspired by the Bag of Visual Words [21] image classification approach. First a vocabulary of visual words is established from data and then every scene is represented as a histogram of the found words. Histogram features are matched by a distance function and loop-closure is detected by setting a match threshold. A similar approach during the same time was proposed by Angeli et al. [3] and these both works provide methods for the both loop-closure detection and data association. FAB-MAP 2.0 [6] added an inverted index for sparse approximation which boosted the matching speed and scalability so that they did not anymore restrict the map size. A 3D FAB-MAP 3D was introduced in [7]. In 3D FAB-MAP the camera image is augmented with depth information that gives the visual words a relative 3D position. With the help of depth information they were able to improve the loop closure recall from 0.42 to 0.71 with the same dataset. A number of different approaches were compared for monocular visual SLAM in [22].

In recent works, Convolutional Neural Network (CNN) based feature embedding [23] has replaced the BoW features [24] in image retrieval and in loop-closure detection. Hou et al. [8] compared hand-crafted features to CNN-learned features for loop-closure detection. They found that fully connected layers are not useful for the task by systematically testing features from different layers. In their experiments, the features from the last pooling layer were the best for image matching. Their network was trained for the scene classification task. Unsupervised visual loop closure method was introduced by Merrill and Huang in 2018 [11]. Their method is based on an auto-encoder CNN, where multiple image transformations are created and used for training. One of the image transforms is the training input, while another transform of the same image is used to calculate the histogram of oriented gradients (HOG), which is then used as the ground truth to be learned.

LiDAR intensity images have also been studied for localization by Bârsan et al. in 2018 [25], where they introduced a real-time localization method for self-driving cars, combining LiDAR intensity images with LiDAR scans to a combined embedding.

Image re-identification – Image re-identification is closely related to image retrieval [24], [23] where a query image is matched against a gallery set to find whether the same object or place appears in the gallery. Suitable datasets for robotics are those containing real places and scenes. Gomez et al. [9] explored the CaffeNet CNN with a triplet loss function to train the network for appearance-invariant place recognition. Noh et al. [26] introduced attentive deep local features for learning local feature descriptors. They also released a big Google-Landmarks dataset with over a million images and almost 13,000 different landmarks.

Various other applications and datasets not related to places and scenes also exist. For example, Lou et al. [13] studied vehicle re-identification and proposed a hard negative mining scheme for visually similar images. They released the VERI-Wild dataset of over 400,000 images of over 40,000 vehicles. Kuma et al. [14] provide a solid benchmarking of vehicle re-identification and experiment with different loss functions. They acknowledge that vehicle re-identification is different from, for example, face re-identification, as vehicles are coarser in details and two cars with the same model and color are very hard to distinguish without additional data (such as a visible licence plate). Face re-identification is another popular application. Schroff et al. [15] introduced FaceNet which inspired the network architecture and training setup used in our work. For face re-identification, Ustinova et al. [16] propose a hybrid architecture of a CNN and a bilinear CNN.

III. Method

For loop-closure detection problem, we train a deep neural network for the task of image re-identification. Our images are panoramic (360-degree) images generated from the intensity channel of LiDAR scans. Re-identification is effectively and efficiently solved by learning a mapping from images to a compact Euclidean space where distances directly correspond to semantic similarity, i.e. whether this location is already visited or not.

Training of an effective embedding network requires a suitable architecture, loss function, and hard negative/positive mining. The two popular approaches are the Siamese structure trained with the contrastive loss [27] and a single CNN pipeline trained with the triplet loss [15]. In this work, we adopt the triplet loss approach which is more difficult to implement but is shown to perform well in face and vehicle re-identification [15], [14] and place recognition [28]. The triplet loss has also been shown to outperform many recent loss functions for person re-identification by large margins [29]. Moreover, we experiment variants of the triplet loss: lifted structured loss [30] and Hermans triplet loss [29].

Positive and negative samples - As the main difference to the above works on face, vehicle, and places recognition we need to define the meaning of positive match between two LiDAR scans. In our dataset, this is achieved by setting a spatial distance threshold $\tau_{pos} = 4.0m$ (four meters) which means that each sample s_i within four meters $dist(s_i, s_j) < \tau_{pos}$ is defined as a positive sample for the query scan s_j and all other as negative samples. During the data capture the spatial distances were obtained through an indoor positioning system available in the industrial work site (Section IV).

A. Backbone network

The backbone network used with the triplet loss has a strong impact on the training speed and accuracy and the final test performance. Various backbone networks have been used in literature and they mainly differ in the number and size of layers and the types of pooling layers. We adopt the procedure from other similar works and use only the features from the convolutional part of the network and ignore the final fully-connected layers. We add a global average pooling layer the top of backbones [36] . This gives us the output flattened as a vector, the size of which is dependent on the backbone architecture and input size. We can conveniently control the size of the output vector by having a stack of convolution, batch normalization [37], and ReLU [38] layers between the backbone and the global average pooling layer. For the lifted structured loss and the Hermans triplet loss we also added a batch normalization layer after the average pooling to make the networks perform properly.

After training the backbone network for image re-identification the network is used to extract a global feature vector from a query image (current location) which is matched against the gallery vectors (previous locations). For matching, the closest matches are found using the Euclidean distance which is fast to compute from the query to all gallery vectors. The backbone networks used in our experiments are listed in Table I.

TABLE I
THE BACKBONE NETWORKS USED IN OUR EXPERIMENTS FOR LIDAR SCAN RE-IDENTIFICATION. THE FEATURE VECTOR DIMENSION IS MANUALLY RESTRICTED FOR OTHER EFFICIENTNET VARIANTS EXCEPT B3. THE TOTAL NUMBER OF PARAMETERS IS COUNTED BY SUMMING THE BACKBONE PARAMETERS AND THE PARAMETERS IN EXTRA LAYERS TO ACCOMMODATE THE FEATURE DIMENSIONS.

Backbone	dim(f)	*# of params*
MobileNet V2 [31]	1280	2.3 M
ResNet 50 [32]	2048	23.6 M
SEResNet 50 [33]	2048	26.1 M
DenseNet 121 [34]	1024	7.0 M
EfficientNet B0 [35]	1024	5.4 M
EfficientNet B1 [35]	1024	7.9 M
EfficientNet B2 [35]	1024	9.2 M
EfficientNet B3 [35]	1024	12.4 M
EfficientNet B3 [35]	1536	10.8 M

B. Loss functions

In image re-identification the network topology is coupled with a metric learning loss function that minimizes Euclidean distances in the feature vector space for images that are close in the spatial space, and maximizes for ones further away. Since we selected the triplet loss function and a single backbone architecture we experimented with the three loss functions that can be considered as variants of the triplet loss: *triplet loss* [15], *lifted structured loss* [30] and *Hermans triplet loss* [29].

Triplet loss – Before FaceNet [15] the triplet loss was already used by Weinberger and Saul [39] for clustering and later in a number of other works [40], [41]. Our work is similar to FaceNet in the sense that we apply the triplet loss directly on top of the convolutional part of the backbone network without classification layers.

The triplet loss is based on three data samples selected so that the first is an anchor (a), the second is a positive match (p) (same location as the anchor), and the third is a negative match (n) (different location). The loss function minimizes the squared Euclidean distance from the anchor to the positive sample and maximizes the distance from the anchor to the negative sample,

$$L = \sum_{i}^{N} \left[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha \right]_+ \quad , \tag{1}$$

where α sets the distance margin that the loss function tries to maintain between the locations. The subscript '+' denotes the positive numbers only and therefore the minimum value for the loss is zero when ideally the first term is zero and the second term is $\geq \alpha$.

Hermans triplet loss – Hermans triplet loss is an improvement proposed by Hermans et al. [29] for the face re-identification task. Hermans triplet loss does not require offline hard negative mining which makes the loss function itself more complicated:

$$\begin{aligned} L_{LG}(\theta; X) = \sum_{i=1}^{P} \sum_{a=1}^{K} \Bigg[\log \sum_{\substack{p=1 \\ p\neq a}}^{K} e^{D\left(f_\theta(x_a^i), f_\theta(x_p^i)\right)} \\ + \log \sum_{\substack{j=1 \\ j\neq i}}^{P} \sum_{n=1}^{K} e^{\alpha - D\left(f_\theta(x_a^i), f_\theta(x_n^j)\right)} \Bigg]_+ \end{aligned} \quad . \tag{2}$$

In (2) D is a distance function that does not necessarily need to be the squared Euclidean distance. The function f_θ maps the semantically close points in the data manifold to a metrically close points in the feature space, with the parameters θ. In our case, the f_θ is the neural network. The X denotes the batch of data we are learning from.

Lifted structured loss – Lifted structured embedding loss by Oh et al. [30] implements a similar concept to the triplet loss, but utilizes a combination of all images in each batch. Instead of comparing only three images (A+P+N) with each other, in the lifted structured loss all the images in the same batch are compared to each other. As shown in the following equation all pairings of the negative and positive samples are compared:

$$\begin{aligned} \tilde{J}_{i,j} &= \log \left[\sum_{(i,k)\in N} e^{(\alpha - D_{i,k})} + \sum_{(j,l)\in N} e^{(\alpha - D_{j,l})} \right] + D_{i,j} \\ \tilde{J} &= \frac{1}{2|P|} \sum_{(i,j)\in P} \max(0, \tilde{J}_{i,j})^2 \end{aligned} \quad . \tag{3}$$

C. Data augmentation

Our data augmentation schemes include popular 2D image augmentation techniques and special techniques available for panoramic images. If not otherwise indicated, the experimental results are for all below augmentation techniques enabled.

2D image augmentation – For improving generalization, we employ the standard 2D image augmentation methods during training. Specifically, we use random erasing [42], horizontal flipping as proposed by Simonyan and Zisserman [43], and random cropping proposed by Krizhevsky et al. [44].

Random panoramic rotation – Images constructed from the LiDAR scans span the full 360 degree circle around the sensor (panoramic view). Panoramic view enables a simple augmentation scheme through random rotations. In specific, the Yaw angle of the camera was randomly rotated by $[0°, 360°[$. Rotation was implemented as a simple pixel shift of the image.

Direction flipping – Direction flipping is a variant of the random panoramic rotation. In direction flipping the yaw angle is changed to the opposite direction (180°) to simulate vehicle running to the opposite direction. The probability of direction flipping was set to 50% and it was implemented similar to the rotation augmentation using pixel shifts.

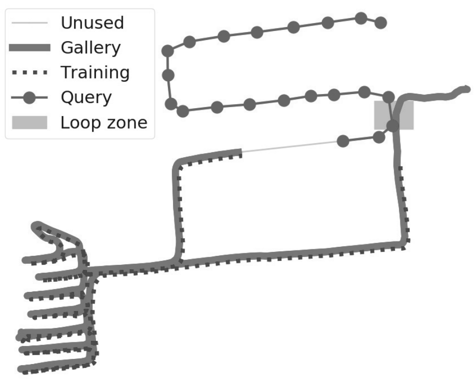


Fig. 2. Dataset split visualized along the spatial path of image capture. Training and gallery sets share significant portion with each other, but the query images as well as substantial zones before and after the gallery images (including the loop-closure zone) are not used in the training set.

D. Training details

Network training and testing were implemented in Python using Keras and TensorFlow. All networks were trained for 200 epochs which was clearly sufficient for convergence for all the networks. The Adam optimizer was used.

The batch size was set according to the available GPU memory (11 GB). The original 2048×64 LiDAR intensity images were scaled down to 512×64 to allow faster and more stable training via bigger batches. For most networks we used 12 anchor images per batch, each with 8 positive examples (within 4.0 m from the anchor) per anchor. For Densenet-121 the anchors per batch was dropped to 10 and for EfficientNets to 8, in order to fit into the memory.

The training samples were batched by taking the specified amount of random anchor points along the training set, and randomly taking corresponding positive match points from within τ_{pos} as defined in the beginning of this section.

IV. Data

LiDAR capturing – For gathering the data, we used a first generation Ouster OS1 LiDAR sensor by Ouster Inc. The Ouster OS1 (1st gen) is a mid-range high resolution imaging LiDAR. The minimum range is 0.8 meters and the maximum range is 120 meters. With an 80% scene reflectivity the sensor can detect to 105 meters with a detection probability of over 90 %. With 10% reflectivity and 90% probability, the range is 40 meters. The sensor captures a 360 by 33.2 degree field of view, and can output 2D images at a maximum of horizontal resolution of 2048 pixels at 10 Hz, and 1024 pixels at 20 Hz. The amount of channels used corresponds to the vertical pixel resolution, and can be chosen to be 16, 32, or 64. The Ouster LiDAR uses intrinsic calibration, has fixed resolution per frame, and boasts a camera-grade ambient and intensity data.

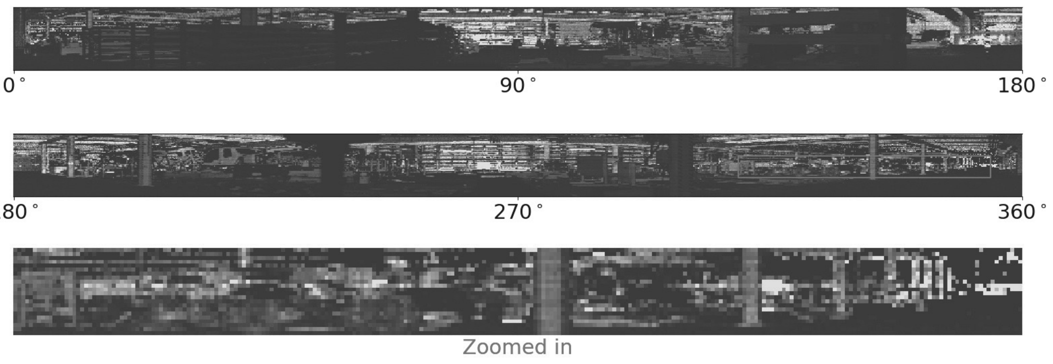


Fig. 3. Example 2D intensity image output from the Ouster OS1 (first gen) LiDAR. It is 2048 x 64 pixels big and greyscale. The image represents the LiDAR measurements from 360 by 33.2 degree angles. It is hard to see details in the unedited format, as in the test scene most details show up quite similar and very dark. The image shown here is colored, brightened up, and split into two 1024 x 64 slices. There is also a zoomed portion (source highlighted in red) to better show off the sensor detail.

We generated our own dataset consisting of a single continuous path in a decently lit indoors industrial environment (a factory). The sensor was mounted to a wheeled stable platform for capture. The height and orientation of the sensor was fixed within the platform. The scale of the environment was well matched to the mid-range specification of the OS1 sensor. We used the highest possible resolution. The path fits into a 173 by 135 meter area, and has less than one meter of vertical range. The total length of the route is 1,026 meters.

The data from the sensor was exported as intensity images. The set was culled to 9,707 images. The corresponding position data are x,y,z coordinates in meters, which is used to generate the positive and negative ground truth labels. The position data was collected via a proprietary high quality SLAM system. The images and the position data was fused together to finalize the dataset. There are multiple loops in the path. The data points are nearly uniformly distributed along the collection path, as there is only slight variance in the distances between data points.

The Ouster LiDAR conveniently outputs LiDAR attribute channels as well as 3D point clouds. In this work we utilized the panoramic intensity images. The sensor is also capable of outputting depth and ambient 2D images, which were not used. The images were directly used as the network inputs. The intensity images are 8-bit greyscale and the resolution of 2048×64.

Evaluation protocol – The dataset was divided into training (6,600 images), gallery (7,200 images), and query (2,100 images) sets visualized in Figure 2. The training and gallery sets are overlapping. There is also an unused section (300 images) between query and gallery set paths to prevent trivially easy matching in the beginning of the query set. Our gallery set includes a prominent loop area that has a corresponding half in the query dataset, but is excluded from the training set. This loop point is manually tagged as being a loop point, while the non-loop points are tagged as being non-loop points. This was thought to be the hardest realistic way to test for loop detection with our dataset.

The mean average precision (mAP) calculation is based on the fact that we manually determined which samples in the query and gallery sets formed a loop and which did not. This gave us a way to determine if the network feature matching gallery images for a given query were either correct (query and gallery points both in the loop zone or both off it) or incorrect (query and gallery points in different zones). This allowed us to use the standard way to calculate mAP.

All the mAP figures presented in Section V are based on a full query set and on the top-1 match (closest in the network computed feature space) from the gallery set. Matches within $\tau_p < 4.0\ m$ are counted as correct recall.

V. Experiments

A. Backbone network

In the first experiment, a number of backbone networks and different loss functions were tested using our data (Section IV). All backbone networks had pre-trained weights using the ImageNet data. The networks were fine-tuned using our training data (Section IV). In these experiments the random panoramic rotation augmentation was applied (Section III-C) with data augmentation procedures of random erasing, horizontal flip, and random cropping. The images were downscaled to 512x64 resolution to allow for bigger batches within available GPU memory. The batch size was 96, but for DenseNet-121 the size was reduced to 80, and for EfficientNet to 64.

The backbone networks from Section III-A were tested: MobileNet V2 [31], Resnet 50 [32], DenseNet 121 [34], EfficientNet [35], and SEResNet 50 [33]. The networks were tested with the different triplet loss variants from Section III-B: lifted structured loss [30], Hermans triplet loss [29] and the triplet loss [15]. The results for all combinations are in Table II.

TABLE II
TOP-1 RESULTS (MAP) FOR DIFFERENT BACKBONE NETWORKS AND DIFFERENT LOSS FUNCTIONS. DIM(f) IS THE FEATURE VECTOR DIMENSION. EFFICIENTNET HAS MULTIPLE IMPLEMENTATIONS DENOTED BY B0-B3 [35]. IN THIS EXPERIMENT ONLY THE BEST MATCHES ARE USED (TOP-1).

Backbone	dim(f)	*Loss function*		
		Lifted Structured	Hermans triplet	Triplet
MobileNet V2	1280	0.711	0.239	0.460
ResNet 50	2048	0.589	0.754	0.640
SEResNet 50	2048	0.686	0.636	0.685
DenseNet 121	1024	0.607	0.604	0.695
EfficientNet B0	1024	0.583	0.756	0.758
EfficientNet B1	1024	0.389	**0.946**	0.732
EfficientNet B2	1024	0.847	0.908	0.765
EfficientNet B3	1024	0.918	0.759	0.734
EfficientNet B3	1536	0.846	0.511	0.822

The EfficientNet variants achieved the best mAP across all three loss functions: 0.918 for lifted structured loss with B3 (1024), 0.946 for Hermans triplet loss with B1 (1024), and 0.822 for conventional triplet loss with B3 (1536). SEResNet 50 and DenseNet 121 performed consistently mAPs ranging from 60% to 70% across all three loss functions. MobileNet V2 performed best with the lifted structured loss, and ResNet 50 with the Hermans triplet. The overall best performance was achieved with EfficientNet B1 (1024) and Hermans triplet loss (mAP 0.946).

To visualize the image re-identification based loop closure the top-1 Euclidean feature space distances are plotted to Figure 4. It is clear the the Euclidean distances are substantially lower in the loop closure region of the test (query) data, which indicates that the learned embedding represents discriminatively the images of different locations in our LiDAR intensity dataset.

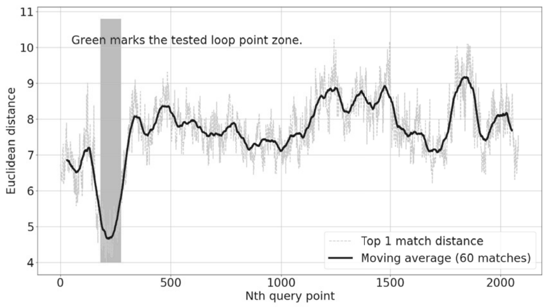


Fig. 4. Query-gallery feature space distances (Euclidean) for the dataset. The green background highlights the loop zone (see Figure 2). The loop-closure is easily detectable from the feature space distances in the network embedding space. The distances in the graph are computed in the space obtained using EfficientNet B1 with Hermans triplet loss and provides mAP of 0.946. (mAP calculation is described in Section IV - "Evaluation protocol")

B. Data augmentation

TABLE III
PERFORMANCE OF THE STRONGEST BASELINE (EFFICIENTNET B1 DIM$(f) = 1024$ W/ HERMANS TRIPLET LOSS) WITH VARIOUS COMBINATIONS OF DATA AUGMENTATION (SECTION III-C).

Augmentation	*mAP*
Baseline - EfficientNet B1 (1024)	0.884
+ random erasing	0.471
+ random crop	0.728
+ horizontal flip	0.402
+ random panoramic rotation	0.825
+ random erasing	0.205
+ all above	**0.946**

We experimented with the image augmentation techniques in Section III-C: random erasing, horizontal flip (50% chance), and random crop. The results are shown in Table III. The results clearly indicate the importance of data augmentation. It is clear that the most effective data augmentation is achieved by enabling all augmentation techniques which yielded to the highest performance (mAP 0.946). Interestingly, the other combinations of the augmentation techniques degraded the performance as compared to the baseline without data augmentation (mAP 0.884).

To study further the complementary nature of the 2D and panoramic augmentation techniques we conducted another set of experiments with various combinations. The results are shown in Table IV. Clear, the only combination that is clearly superior to the baseline using no augmentation is the one that combines all 2D augmentation techniques and the the random panoramic rotation.

VI. CONCLUSION

The main goal of this work was to find the best network architecture, loss function, and data augmentation for CNN-based metric feature embedding so that the embedding can be used in LiDAR image loop-closure detection. Potential applications are heavy machinery localization and SLAM in industrial indoor work sites. For experiments we collected a realistic dataset with an industry quality LiDAR.

We formulated embedding network optimization as an image re-identification problem and adopted the triplet loss as the objective function. The best performance, mAP 0.946, was obtained using EfficientNet B1 as the backbone network, using the Hermans triplet loss function and the following data augmentation techniques: random panoramic rotation, random erasing, random cropping, and random flipping.

Our future work will include collection of large scale public datasets, long-term localization, and integration of the proposed vision-based LiDAR loop-closure detection to a real robot navigation and SLAM.

TABLE IV
THE PERFORMANCE OF DIFFERENT AUGMENTATION SCHEMES AS TESTED ON EFFICIENTNET B1 (1024) BACKBONE AND HERMANS TRIPLET LOSS. THE COLUMNS REPRESENT THE COMMON 2D IMAGE AUGMENTATION TECHNIQUES AND THE ROWS ARE THE SPECIFIC AUGMENTATIONS TECHNIQUES FOR PANORAMIC LIDAR DATA. THE BEST COMBINATION IS RANDOM ERASING, RANDOM CROPPING, AND HORIZONTAL FLIPPING COMBINED WITH THE RANDOM PANORAMIC ROTATION (0.946 MAP).

	None	**Erasing**	**Cropping**	**Erasing & Cropping**	**Erasing & Cropping & Horizontal Flipping**
None	0.884	0.471	0.312	0.728	0.402
Panoramic direction flipping	0.669	0.526	0.847	0.539	0.420
Random panoramic rotation	0.824	0.205	0.493	0.173	**0.946**

REFERENCES

[1] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part i,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, 2006.

[2] G. DeSouza and A. Kak, “Vision for mobile robot navigation: A survey,” *PAMI*, vol. 24, no. 2, 2002.

[3] A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer, “A fast and incremental method for loop-closure detection using bags of visual words,” *IEEE Transactions on Robotics*, pp. 1027–1037, 2008.

[4] M. Cummins and P. Newman, “Fab-map: Probabilistic localization and mapping in the space of appearance,” *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[5] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, “Openfabmap: An open source toolbox for appearance-based loop closure detection,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 4730–4735.

[6] M. Cummins and P. Newman, “Appearance-only slam at large scale with fab-map 2.0,” *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[7] R. Paul and P. Newman, “Fab-map 3d: Topological mapping with spatial and visual appearance,” in *2010 IEEE international conference on robotics and automation*. IEEE, 2010, pp. 2649–2656.

[8] Y. Hou, H. Zhang, and S. Zhou, “Convolutional neural network-based image representation for visual loop closure detection,” in *2015 IEEE international conference on information and automation*. IEEE, 2015, pp. 2238–2245.

[9] R. Gomez-Ojeda, M. Lopez-Antequera, N. Petkov, and J. Gonzalez-Jimenez, “Training a convolutional neural network for appearance-invariant place recognition,” *arXiv preprint arXiv:1505.07428*, 2015.

[10] X. Gao and T. Zhang, “Loop closure detection for visual slam systems using deep neural networks,” in *2015 34th Chinese Control Conference (CCC)*. IEEE, 2015, pp. 5851–5856.

[11] N. Merrill and G. Huang, “Lightweight unsupervised deep loop closure,” *arXiv preprint arXiv:1805.07703*, 2018.

[12] Y. Xia, J. Li, L. Qi, H. Yu, and J. Dong, “An evaluation of deep learning in loop closure detection for visual slam,” in *2017 IEEE international conference on internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*. IEEE, 2017, pp. 85–91.

[13] Y. Lou, Y. Bai, J. Liu, S. Wang, and L. Duan, “Veri-wild: A large dataset and a new method for vehicle re-identification in the wild,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3235–3243.

[14] R. Kuma, E. Weill, F. Aghdasi, and P. Sriram, “Vehicle re-identification: an efficient baseline using triplet embedding,” in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–9.

[15] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[16] E. Ustinova, Y. Ganin, and V. Lempitsky, “Multi-region bilinear convolutional neural networks for person re-identification,” in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2017, pp. 1–6.

[17] Y. Wang, J. Shen, S. Petridis, and M. Pantic, “A real-time and unsupervised face re-identification system for human-robot interaction,” *Pattern Recognition Letters*, vol. 128, pp. 559–568, 2019.

[18] R. Smith and P. Cheeseman, “On the representation and estimation of spatial uncertainty,” *Int. J. of Robotics Research*, vol. 5, no. 4, 1986.

[19] H. Durrant-Whyte, “Uncertain geometry in robotics,” *IEEE J. of Robotics and Automation*, vol. 4, no. 1, 1988.

[20] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part ii,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, 2006.

[21] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, “Automatic differentiation in pytorch,” in *ECCV Workshops*, 2004.

[22] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardos, “A comparison of loop closing techniques in monocular SLAM,” *Robotics and Autonomous Systems*, vol. 57, 2009.

[23] F. Radenović, G. Tolias, and O. Chum, “CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples,” in *ECCV*, 2016.

[24] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Obj. retrieval with large voc. and fast spatial matching,” in *CVPR.*, 2007.

[25] I. A. Barsan, S. Wang, A. Pokrovsky, and R. Urtasun, “Learning to localize using a lidar intensity map,” in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 605–616. [Online]. Available: http://proceedings.mlr.press/v87/barsan18a.html

[26] H. Noh, A. Araujo, J. Sim, T. Weyand, and B. Han, “Large-scale image retrieval with attentive deep local features,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 3456–3465.

[27] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[28] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, “Netvlad: Cnn architecture for weakly supervised place recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[29] A. Hermans, L. Beyer, and B. Leibe, “In defense of the triplet loss for person re-identification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017.

[30] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, “Deep metric learning via lifted structured feature embedding,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4004–4012.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[32] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[34] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[35] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.
[36] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.
[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.
[39] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification." *Journal of Machine Learning Research*, vol. 10, no. 2, 2009.
[40] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1386–1393.
[41] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International Workshop on Similarity-Based Pattern Recognition*. Springer, 2015, pp. 84–92.
[42] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation." in *AAAI*, 2020, pp. 13 001–13 008.
[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

PUBLICATION
III

Evaluation of Long-term LiDAR Place Recognition

Jukka Peltomäki, Farid Alijani, Jussi Puura, Heikki Huttunen, Esa Rahtu, and Joni-Kristian Kämäräinen

In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 2021, pp. 4487–4492

Evaluation of Long-term LiDAR Place Recognition

Jukka Peltomäki⋆, Farid Alijani⋆, Jussi Puura†, Heikki Huttunen⋆, Esa Rahtu⋆, and Joni-Kristian Kämäräinen⋆

***Abstract*— We compare a state-of-the-art deep image retrieval and a deep place recognition method for place recognition using LiDAR data. Place recognition aims to detect previously visited locations and thus provides an important tool for navigation, mapping, and localisation. Experimental comparisons are conducted using challenging outdoor and indoor datasets, Oxford Radar RobotCar and COLD, in the "long-term" setting where the test conditions differ substantially from the training and gallery data. Based on our results the image retrieval methods using LiDAR depth images can achieve accurate localization (the single best match recall 80%) within 5.00 m in urban outdoors. In office indoors the comparable accuracy is 50 cm but is more sensitive to changes in the environment.**

I. Introduction

An autonomous robot that operates in an environment should be able to recognize different places when it revisits them after some time (Figure 1, top). This is important to support reliable navigation, mapping, and localisation. Robust place recognition is therefore a crucial capability for an autonomous robot. Due to its importance place recognition is an important research topic in robotics and computer vision community for which Lowry et al. [1] and Zhang et al. [2] survey the past works.

The problem of visual place recognition gets more challenging if the visual appearance of places change over time. This usually happens due to changes in the lighting conditions, shadows, different weather conditions, or even different seasons. Also, people moving around and items being moved around change the environment. These factors are particularly addressed in *long-term visual place recognition*. Engineered features can be adjusted to be invariant [3], [4], but the recent deep learning methods are prone to overfitting and therefore their performance depends on suitability of the selected training data [5], [6]. These works lack in one or more terms: they focus either indoor or outdoor place recognition, limited variability, unrealistic navigation data or focus on RGB images only.

In this paper, we study deep place recognition using LiDAR sensor. Compared to typical RGB camera, LiDAR is less rich in details but more robust to various sources of imaging distortions such as illumination change and weather conditions. We perform extensive experiments on two largest indoor and outdoor datasets that include LiDAR measurements and are suitable for robot navigation: Oxford Radar RobotCar [7][8] (3D LiDAR) and COLD [9] (2D LiDAR). Our findings are that LiDAR is competitive modality to RGB

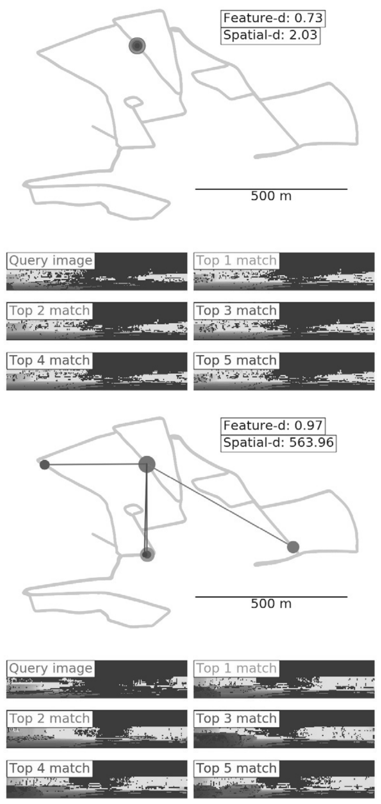


Fig. 1. Examples of successful (top) and failed place recognition (bottom) in the RobotCar dataset. In successful recognition the best five matches (red points) are all near the query place (green point) and within a small feature distance while in the failed case the feature distances are larger and matches are found in random spatial locations.

and especially the range (depth map) measurements are robust to long-term changes. We also show that the state-of-the-art image retrieval method by Radenovic et al. [10] performs well in place recognition even without data specific fine-tuning and is not sensitive to backbone network selection. We publish all code and data to facilitate fair comparisons and future works on place recognition for robot navigation.

Related work – Surveys for RGB-based methods can be found from Lowry et al. [1] (engineered features) and

⋆Tampere University, Finland
†Sandvik Mining and Construction Ltd

Zhang et al. [2] (deep learning). Visual localization combines place recognition and refined localization (inc. 3D pose) using RGB images. Hierarchical Localization using SuperPoint and SuperGlue [11], [12] is the top performing system for visual localization in the recent benchmark by Pion et al. [13].

A number of methods have been proposed for point cloud and LiDAR based place recognition. For example, PointNetVLAD [14] uses global features of point cloud data for place recognition. Steder et al. [15] use LiDAR range images. Guo et al. [16] combine the both LiDAR range (depth) and intensity values and use a probabilistic voting scheme. A network architecture utilizing the combination of RGB camera and LiDAR point clouds for place recognition was introduced by Xie et al. [17]. A place recognition system featuring adversarial training and octree mapping was introduced by Yin et al. [18]. Kim and Kim [19] introduced scan contexts, a type of spatial descriptor, to improve results with point cloud place recognition. Our work focus on LiDAR only place recognition to analyze whether LiDAR depth or intensity images work well in long-term place recognition.

Several datasets suitable for place recognition are publicly available: MulRan [20], The Newer College Dataset [21], COLD [9], NCLR [22], Mapillary Street-Level Sequences [23], and Oxford Radar RobotCar [7]. For our experiments we selected COLD and Oxford Radar RobotCar, as both represent realistic navigation sequences, include LiDAR, and are large long-term datasets.

II. Methods

The two methods experimented in our work are NetVLAD by Arandjelovic et al. [24] and CNN retrieval (CNNRetr) by Radenovic et al. [25]. NetVLAD [24] was selected as it is used in *Hierarchical Localization using SuperPoint and SuperGlue* [11], [12] that won the 2020 Visual Localization Challenge (`https://www.visuallocalization.net/`). On the other hand, the CNNRetr is at the core of the state-of-the-art image retrieval architecture of Radenovic et al. [10], [25], [26]. In the following we briefly introduce these methods and their adaptation to place recognition.

A. Deep place recognition (NetVLAD)

The core idea of NetVLAD [24] is in deep metric learning where the deep architecture learns to produce a representation that encodes the informative content of inputs. The representation is metric in the sense that similarity of inputs, such as RGB or LiDAR range images, can be measured by standard distance functions such as Euclidean distance. In other words, the objective is to learn a function f_θ with its parameters (network weights) defined by θ that maps images I_i to a high (D-)dimensional feature vector space $f_\theta : I \rightarrow \mathbb{R}^D$. The high dimensional representation encodes images from the same place with a small distance value $d_\theta(I_i, I_j) = ||f_\theta(I_i) - f_\theta(I_j)||$ and images from different places with large distance values.

The main building blocks of NetVLAD are 1) the *NetVLAD layer* that implements a differentiable version of the VLAD encoding of SIFT features [27] to replace maximum pooling, 2) *triplet ranking loss*, 3) *Principal Component Analysis (PCA) based dimensionality reduction* and 4) *training procedure using Google Street View Time Machine dataset* that provides multiple close-by images of the same spatial locations captured at different times.

The original VLAD representation is a $K \times D$-dimensional matrix where K denotes cluster centers (visual words) and D is the number of feature dimensions. The SIFT detector provides N descriptors that are VLAD encoded using the following formula:

$$V(j,k) = \sum_{i=1}^{N} a_k(\mathbf{x}_i)(x_i(j) - c_k(j)), \qquad (1)$$

where $x_i(j)$ and $c_k(j)$ are the j-th dimensions of the i-th descriptor and k-th cluster centers. The $a_k(\mathbf{x}_i)$ means that the descriptor x_i belongs to the k-th visual word. In other words, $\mathbf{V}$ encodes feature distances from the visual words that are obtained by clustering all features in the training set. This encoding is more powerful than the original Bag-of-Words (BoW) encoding [27], but with the price of much larger feature vectors ($\mathbf{V}$ can be converted to a vector). NetVLAD layer uses a differentiable version of (1)

$$V(j,k) = \sum_{i=1}^{N} \frac{e^{\mathbf{w}_k^T \mathbf{x}_i + b_k}}{\sum_{k'} e^{\mathbf{w}_{k'}^T \mathbf{x}_i + b_{k'}}} (x_i(j) - c_k(j)), \qquad (2)$$

where $\{\mathbf{w}\}_k$, $\{b\}_k$ and $\{\mathbf{c}\}_k$ are sets of parameters optimized during training. Eq. 2 is obtained from (1) by applying soft-assignment instead of the original hard assignments. The downside of the NetVLAD features is their high dimensionality and therefore Arandjelovic et al. [24] propose a PCA-based dimensionality reduction as a post-processing step. However, in our experiment we found it unnecessary and therefore used the full NetVLAD feature vectors which for 512-dim deep features and 64 clusters have $512 \times 64 = 32,768$ elements.

The typical formulation of the triplet loss [28] is

$$\max\left(||f_\theta(I_A) - f_\theta(I_P)||^2 - ||f_\theta(I_A) - f_\theta(I_N)||^2 + \alpha, 0\right), \qquad (3)$$

where I_A is the "anchor image" (query image from the training set), I_P is a positive example and I_N is a random negative example and α is the margin enforced between the anchor and negative images. Instead of the triplet loss the NetVLAD network is optimized using the triplet ranking loss

$$\max\left(\min_i ||(f_\theta(I_A) - f_\theta(I_P^{(i)})||^2 - ||f_\theta(I_A) - f_\theta(I_N)||^2 + \alpha, 0\right), \qquad (4)$$

that can handle multiple positive candidates $I_P^{(i)}$ and select only the distance to the best match. The triplet ranking loss is needed since the Google Street View Time Machine dataset contains panoramic images that are converted to multiple projective images and only the images viewing the same direction are correct matches. However, since in our case the images come from a LiDAR that can be considered as a projective sensor we adopt the standard triplet loss from [28].

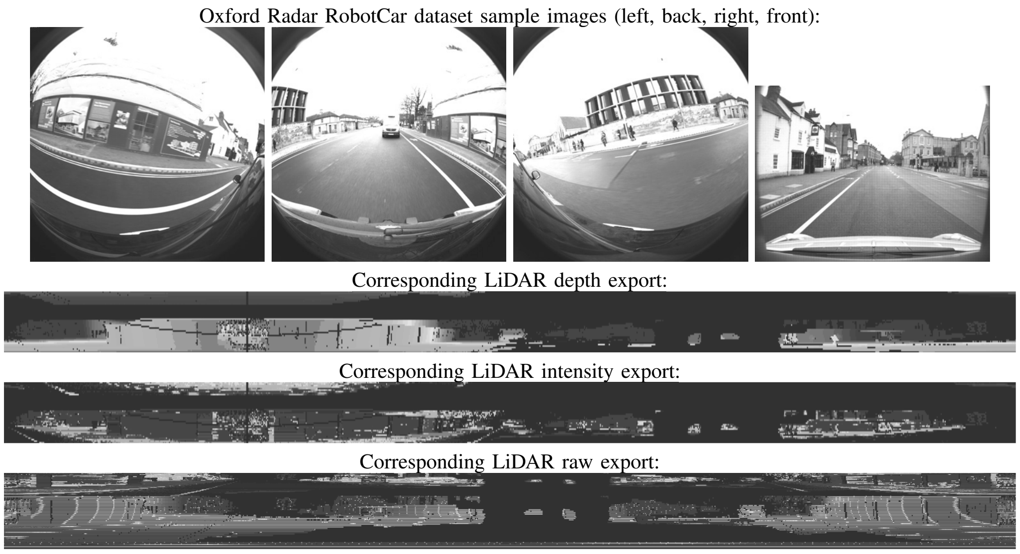

Oxford Radar RobotCar dataset sample images (left, back, right, front):
Corresponding LiDAR depth export:
Corresponding LiDAR intensity export:
Corresponding LiDAR raw export:

Fig. 2. Example from the Oxford Radar RobotCar [7]. The dataset is fully outdoors. The Velodyne LiDAR exports (bottom) are colorized, brightened and cropped for better visualization. The actual LiDAR-exported images fed to the network are 1024 × 41 pix 8-bit greyscale.

B. Deep image retrieval (CNNRetr)

CNN Retrieval, CNNRetr [10][25], is an image retrieval method. The key components of CNNRetr are a fully convolutional backbone network, generalized-mean pooling layer, siamese architecture with contrastive loss, and whitening with dimension reduction. After training, the image retrieval is done by exhaustively comparing the Euclidean distances between the resulting feature vectors.

The fully convolutional neural network used in CNNRetr [10][25] can be of any convolutional architecture, such as the popular VGG or ResNet. The 3D tensor output χ of size $W * H * K$ from the network is fed into a pooling layer.

CNNRetr [10] employs the usage of generalized-mean (GeM) pooling,

$$\mathbf{f} = [f_1 ... f_k ... f_K]^\top, f_k = \left(\frac{1}{|\chi_k|} \sum_{x \in \chi_k} x^{p_k} \right)^{\frac{1}{p_k}}, \quad (5)$$

where the input χ is pooled into a vector $\mathbf{f}$. χ_k is the set of activations for the feature map $k \in \{1...K\}$. The pooling parameter p_k can be learned, as the GeM layer is differentiable.

The GeM-pooled and l_2-normalized feature vector is used for contrastive loss training in a siamese network architecture. Input images are fed as pairs (i, j) with corresponding labels $Y(i, j) \in \{0, 1\}$, where 1 means that the images are matches, and 0 means that the images are not matches. The contrastive loss,

$$L(i,j) = \begin{cases} \frac{1}{2}\|\mathbf{f}(i) - \mathbf{f}(j)\|^2, & Y(i,j) = 1 \\ \frac{1}{2}(max\{0, \tau - \|\mathbf{f}(i) - \mathbf{f}(j)\|\}^2, & Y(i,j) = 0 \end{cases}, \quad (6)$$

decreases the Euclidean distance between matching images and increases between non-matching images. The parameter τ is the enforced margin between the non-matching examples.

After training, feature vector whitening is performed to improve search precision, and dimension reduction is optionally done to improve performance and resource requirements. CNNRetr learns whitening by employing a structure from motion (SfM) pipeline to reconstruct a scene to extract matching points. They use linear discriminant projections, which involves two phases: whitening and rotation. Here, the first part, whitening, is calculated as the inverse square root of the covariance matrix within each matching class, $C_S^{-\frac{1}{2}}$, where

$$C_S = \sum_{Y(i,j)=1} (\mathbf{f}(i) - \mathbf{f}(j))(\mathbf{f}(i) - \mathbf{f}(j))^\top \ . \quad (7)$$

The second part, rotation, of the linear discriminant projection, is the principal component analysis (PCA) of the covariance matrix of the non-matching pairs in the whitened space $eig(C_S^{-\frac{1}{2}} C_D C_S^{-\frac{1}{2}})$, where C_D is basically the same calculation as C_S (Eq. 7) but for non-matches,

$$C_D = \sum_{Y(i,j)=0} (\mathbf{f}(i) - \mathbf{f}(j))(\mathbf{f}(i) - \mathbf{f}(j))^\top \ . \quad (8)$$

The two linear discriminant projection steps are combined as the projection via multiplication $P = C_S^{-\frac{1}{2}} eig(C_S^{-\frac{1}{2}} C_D C_S^{-\frac{1}{2}})$. To apply the projection, the mean GeM vector to perform centering, μ, is taken into account to get the wanted variance. The applied projection finally is

$P^\top(\mathbf{f}(i) - \mu)$, which is also l_2-normalized to get the fully whitened feature vectors to be used in the search process.

III. Experiments

A. Datasets and settings

The experiments were conducted on the two largest and publicly available datasets suitable for outdoor and indoor navigation: Oxford Radar RobotCar [7] (outdoors) and COsy Localization Database (COLD) [9] (indoors).

Radar RobotCar – The Oxford Radar RobotCar dataset [7] is an extension of the original RobotCar dataset [8] and thus follows the original dataset route in Oxford, UK. It consists of 32 traversals in different traffic, weather, and lighting conditions in January 2019. The new dataset contains measurements from three point cloud radars installed on the top of the car and all provide full 360-degree panoramic views around it. In the middle is a Navtech FMCW radar that provides 400 measurements per scan 4 Hz and on its both sides two 20 Hz Velodyne LiDARs of $41.3°$ vertical FoV sensors. For simplicity, we used just one of the two LiDARs, and we randomly selected the *left Velodyne LiDAR* for our experiments. Note that the route is always to the same direction. See Figure 2 for example images. The velocity of the car is moderate and thus the distance between two measurements is rarely more than 0.5 m. We selected the following sequences for our experiments:

- Train: Jan-10-2019-11:46, Cloudy
- Gallery: Jan-10-2019-12:32, Cloudy
- Query 1: Jan-10-2019-14:50, Cloudy
- Query 2: Jan-11-2019-12:26, Sunny
- Query 3: Jan-16-2019-14:15, Rainy

Train and Gallery set images were used to train the two methods and the Gallery was also used as the place recognition database (gallery set). Query sets were chosen from different days with different weather.

COLD – The COsy Localization Database [9] is an indoor navigation dataset. The data has been gathered in 76 sequences across three different locations in Europe. The sequences are varied in lighting conditions such as sunny, cloudy, and night. The sequences also contain dynamic elements such as people moving and rearranged furniture. The room types are annotated, and odometry is used for localization. The sequences are arranged as some being ”standard” or ”extended”. The standard sequences contain rooms that are found in the sequences from the other two locations, as well, and the extended sequences contain location specific room types.

The data is captured with manually driven mobile robots. The robots are equipped with two Videre Design MDCS2 cameras, one in typical perspective mode and the other capturing omnidirectional images. SICK 2D laser scanner is used to capture range information. SICK 2D provides only a single 360-degree line scan of 361 samples that we convert to artificial depth image by expanding it vertically (Figure 3).

For our experiments, we employed datasets similar to the Oxford Radar RobotCar dataset. We had five sequences from

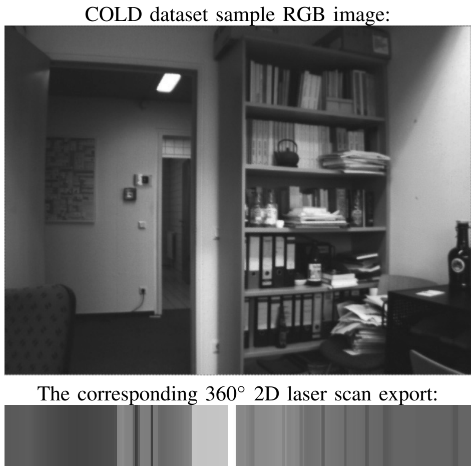

COLD dataset sample RGB image:
The corresponding 360° 2D laser scan export:

Fig. 3. Example from the COsy Localization Database (COLD) [9]. The dataset is fully indoors. The greyscale images below are the expanded SICK laser scans spanning 360-degree around the robot (white encodes distances above 8 m and black is 0 m). The open door on the left-hand-side can be seen as dark gray region, the opposite wall as light gray and the open doorway as a completely white strip in the middle of the scan.

the same office. The train set sequence was sunny. The gallery set sequence was cloudy, and the three tested query set sequences represented all the different light categories: sunny, cloudy, and night time:

- Train: Saarbrücken, Part B, Sequence 4, Sunny 3
- Gallery: Saarbrücken, Part B, Sequence 4, Cloudy 1
- Query 1: Saarbrücken, Part B, Sequence 4, Sunny 1
- Query 2: Saarbrücken, Part B, Sequence 4, Cloudy 2
- Query 3: Saarbrücken, Part B, Sequence 4, Night 3

a) Performance measure: All experiments were conducted using the top-1 retrieval results, i.e. only the best matching image was used. Our performance measure is thus Recall@1 i.e. the number of correctly retrieved locations divided by the number of all query images [24].

b) Settings: If not otherwise mentioned the default parameters of CNNRetr and NetVLAD networks from the original authors were used.

For the both methods we used the triplet loss as that was found performing well and makes comparison between the two architectures fair. The positive P and anchor A samples were selected randomly within the given distance threshold used in training. A list of positive matches for each image was generated prior training. The hard negative mining was conducted according to [24], [29].

B. Results

a) Method comparison: The two methods compared were CNNRetr [25] and NetVLAD [24] described in Section II. In the first experiment, we compared using the methods without fine-tuning to our datasets. CNNRetr [10] is used as is, however for NetVLAD we do not perform the PCA-based dimensional reduction as the results in [25]

TABLE I

COMPARISON OF THE TWO METHODS WITH THEIR DEFAULT SETTINGS (VGG16 BACKBONE AND WITHOUT FINE-TUNING) AND A TRAINED CNNRETR.

	Outdoor dataset - RobotCar [8]				Indoor dataset - COLD [9]			
Method	Rec@1-25m	Rec@1-10m	Rec@1-5m	Rec@1-2m	Rec@1-100cm	Rec@1-50cm	Rec@1-25cm	Rec@1-10cm
Query 1 (same day, 2h later)					*Query 1 (sunny)*			
NetVLAD [24] (no train)	0.899	0.867	0.728	0.130	0.838	**0.779**	**0.464**	**0.008**
CNNRetr [10] (no train)	0.926	0.901	0.774	0.134	0.783	0.723	0.439	0.004
CNNRetr [10] (trained)	**0.986**	**0.977**	**0.869**	**0.155**	**0.847**	0.774	0.462	0.004
Query 2 (next day, same time)					*Query 2 (cloudy)*			
NetVLAD [24] (no train)	0.895	0.863	0.762	0.454	**0.230**	<0.000	<0.000	<0.000
CNNRetr [10] (no train)	0.887	0.856	0.758	0.468	0.100	<0.000	<0.000	<0.000
CNNRetr [10] (trained)	**0.993**	**0.984**	**0.902**	**0.544**	0.091	<0.000	<0.000	<0.000
Query 3 (after 6 days, 2h later)					*Query 3 (night)*			
NetVLAD [24] (no train)	0.527	0.449	0.325	0.049	0.264	0.005	<0.000	<0.000
CNNRetr [10] (no train)	0.678	0.608	0.465	0.060	**0.267**	0.005	<0.000	<0.000
CNNRetr [10] (trained)	**0.918**	**0.856**	**0.642**	**0.089**	0.263	0.005	**0.001**	<0.000

suggest that PCA may degrade the results, which we also found out to happen in our experiments. The results for the two dataset without fine-tuning are in Table I.

These results provide the following three findings: 1) there is no substantial performance difference between CNNRetr and NetVLAD; 2) the accuracy of LiDAR-based place recognition is between 2-5 meters with the RobotCar dataset and 25-50 centimeters with the COLD dataset (top-1 recall above 70%); 3) LiDAR-based recognition fails for the indoor dataset query sequences that are substantially different from the gallery dataset (Query 2 and Query 3). This can be explained by the fact that the 360-degree line LiDAR of the COLD dataset does not provide enough information for place recognition. Since COLD is the only indoor dataset for long-term place recognition and including LiDAR there is obvious need for new indoor navigation datasets.

b) Fine-tuning with training data: The best performing method (CNNRetr [25]) was trained with dataset specific training data (gallery sequence and one training sequence). The top-1 recall (Rec@1) values are shown in Table I. The results clearly demonstrate that dataset specific training improves the results by 10-20%. However, the training did not improve the results for Query 2 and 3 images of the indoor dataset that still failed.

c) Backbone network: The typical image retrieval backbone networks are VGG16 and ResNet-50 which were compared during our experiments. The results are shown only for the Radar RobotCar dataset as the indoor results overall were poor for Query 2 and 3 sets. The results are shown in Table II. Clearly, the selection of backbone has only small impact and thus VGG16 is preferable as it is computationally lighter.

d) LiDAR intensity vs. range: LiDAR intensity and depth scan performance are compared in Table III and as functions of the training epochs in Figure 4. While in Query 1 the LiDAR intensity is slightly better, the depth is clearly better in Query 2 and 3 where the conditions are more challenging. The RGB results with the same network are also added to demonstrate that LiDAR-only accuracy is comparable to RGB. Interestingly the "raw" LiDAR data

TABLE II

BACKBONE COMPARISON USING CNNRETR. THE DETECTION THRESHOLD OF 5.0 M WAS USED IN TRAINING.

	Outdoor dataset - RobotCar [8]			
Method	Rec@1-25m	Rec@1-10m	Rec@1-5m	Rec@1-2m
Query 1 (same day, 2h later)				
VGG16	**0.976**	0.966	**0.866**	**0.154**
ResNet-50	0.970	**0.957**	0.846	**0.154**
Query 2 (next day, same time)				
VGG16	**0.987**	**0.979**	**0.897**	**0.573**
ResNet-50	0.953	0.941	0.869	0.555
Query 3 (after 6 days, 2h later)				
VGG16	**0.832**	**0.777**	0.587	**0.081**
ResNet-50	0.830	0.773	**0.601**	0.080

provided with RobotCar data is worse than the depth channel.

IV. CONCLUSION

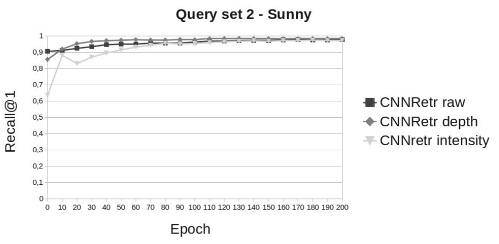
Our experiments provide the following important findings: i) LiDAR is competitive sensor modality (vs. RGB camera) for place recognition, ii) LiDAR depth maps are more robust to long-term changes than LiDAR intensity images, iii) SoTA deep image retrieval architecture "CNNRetr" by Radenovic et al. [10] provides place recognition accuracy of 5 meters urban outdoors and 50 centimeters with recall approx. 80% iv) the backbone network selection is not critical, and v) feature fine-tuning with dataset specific data provides improvement of 10-20%. Two important future directions were also pointed out: a) new indoor navigation datasets with high quality LiDAR are needed and b) complementarity of LiDAR depth, LiDAR intensity and RGB should be further investigated.

REFERENCES

[1] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2016.

[2] X. Zhang, L. Wang, and Y. Su, "Visual place recognition: A survey from deep learning perspective," *Pattern Recognition*, p. 107760, 2020.

[3] M. Shakeri and H. Zhang, "Illumination invariant representation of natural images for visual place recognition," in *IROS*, 2016.



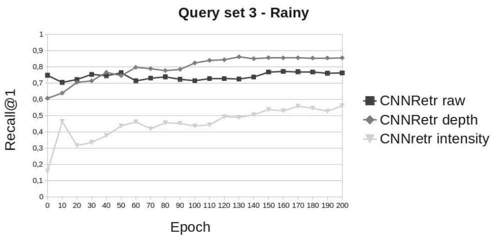


Fig. 4. In sunny weather (top) all the Velodyne LiDAR export modes perform well. In rainy conditions (bottom) the depth mode performs clearly better than the intensity mode and slightly better than the "raw" data.

TABLE III
LIDAR INTENSITY VS. LIDAR RANGE (DEPTH) COMPARISON (CNNRETR, RESNET-50, 5.0M TRAINING THRESHOLD).

	Outdoor dataset - RobotCar [8]			
Method	Rec@1-25m	Rec@1-10m	Rec@1-5m	Rec@1-2m
Query 1 (same day, 2pm50)				
Intensity	0.974	0.964	0.863	0.142
Depth	0.970	0.957	0.846	0.154
Raw	0.944	0.921	0.786	0.130
RGB	**0.984**	**0.974**	**0.958**	**0.598**
Query 2 (next day, 12pm26)				
Intensity	0.940	0.927	0.855	0.542
Depth	**0.953**	**0.941**	**0.869**	**0.555**
Raw	0.938	0.912	0.809	0.519
RGB	0.951	0.934	**0.869**	0.480
Query 3 (after 6 days, 2pm15)				
Intensity	0.560	0.509	0.381	0.056
Depth	0.830	0.773	0.601	0.080
Raw	0.685	0.627	0.493	0.070
RGB	**0.920**	**0.890**	**0.846**	**0.650**

[4] M. Ullah, A. Pronobis, B. Caputo, J. Luo, P. Jensfelt, and H. Christensen, "Towards robust place recognition for robot localization," in *IROS*, 2008.

[5] N. Sunderhauf, S. Shirazi, F. Dayoub, B. Upcroft, and M. Milford, "On the performance of convnet features for place recognition," in *IROS*, 2015.

[6] Z. Xin, Y. Cai, T. Lu, X. Xing, S. Cai, J. Zhang, Y. Yang, and Y. Wang, "Localizing discriminative visual landmarks for place recognition," in *ICRA*, 2010.

[7] D. Barnes, M. Gadd, P. Murcutt, P. Newman, and I. Posner, "The

[9] A. Pronobis and B. Caputo, "COLD: COsy Localization Database," *The International Journal of Robotics Research (IJRR)*, vol. 28, no. 5,

oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset," in *ICRA*, 2020.

[8] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset," *The International Journal of Robotics Research (IJRR)*, vol. 36, no. 1, pp. 3–15, 2017.
2009.

[10] F. Radenović, G. Tolias, and O. Chum, "CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples," in *ECCV*, 2016.

[11] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk, "From coarse to fine: Robust hierarchical localization at large scale," in *CVPR*, 2019.

[12] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperGlue: Learning feature matching with graph neural networks," in *CVPR*, 2020.

[13] N. Pion, M. Humenberger, G. Csurka, Y. Cabon, and T. Sattler, "Benchmarking image retrieval for visual localization," in *Int. Conf. on 3D Vision (3DV)*, 2020.

[14] M. A. Uy and G. H. Lee, "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4470–4479.

[15] B. Steder, G. Grisetti, and W. Burgard, "Robust place recognition for 3d range data based on point features," in *ICRA*, 2010.

[16] J. Guo, P. V. Borges, C. Park, and A. Gawel, "Local descriptor for robust place recognition using lidar intensity," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1470–1477, 2019.

[17] S. Xie, C. Pan, Y. Peng, K. Liu, and S. Ying, "Large-scale place recognition based on camera-lidar fused descriptor," *Sensors*, vol. 20, no. 10, p. 2870, 2020.

[18] P. Yin, L. Xu, Z. Liu, L. Li, H. Salman, Y. He, W. Xu, H. Wang, and H. Choset, "Stabilize an unsupervised feature learning for lidar-based place recognition," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1162–1167.

[19] G. Kim and A. Kim, "Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4802–4809.

[20] G. Kim, Y. S. Park, Y. Cho, J. Jeong, and A. Kim, "Mulran: Multimodal range dataset for urban place recognition," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6246–6253.

[21] M. Ramezani, Y. Wang, M. Camurri, D. Wisth, M. Mattamala, and M. Fallon, "The newer college dataset: Handheld lidar, inertial and vision with ground truth," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[22] N. Carlevaris-Bianco, A. K. Ushani, and R. M. Eustice, "University of michigan north campus long-term vision and lidar dataset," *The International Journal of Robotics Research*, vol. 35, no. 9, pp. 1023–1035, 2016.

[23] F. Warburg, S. Hauberg, M. Lopex-Antequera, P. Gargallo, Y. Kuang, and J. Civera, "Mapillary street-level sequences: A dataset for lifelong place recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: Cnn architecture for weakly supervised place recognition," *TPAMI*, 2018.

[25] F. Radenović, G. Tolias, and O. Chum, "Fine-tuning CNN image retrieval with no human annotation," *TPAMI*, 2018.

[26] ——, "Deep shape matching," in *ECCV*, 2018.

[27] H. Jegou, M. Douze, C. Schmid, and P. Perez, "Aggregating local descriptors into a compact image representation," in *CVPR*, 2010.

[28] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[29] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

PUBLICATION IV

LiDAR Place Recognition Evaluation with the Oxford Radar RobotCar Dataset Revised

Jukka Peltomäki, Farid Alijani, Jussi Puura, Heikki Huttunen, Esa Rahtu, and Joni-Kristian Kämäräinen

In: *Image Analysis: 22nd Scandinavian Conference, SCIA 2023, Levi Ski Resort in Lapland, Finland, April 18–21, 2023, Proceedings*. 2023. Springer. In press