A low-cost Localization Method for Mobile Robots in well-known Environments using Convolutional Neural Networks

Robin Dietrich
School of Electrical Engineering and Computer Science
Oregon State University, Corvallis, Oregon 97331
dietriro@oregonstate.edu

Abstract

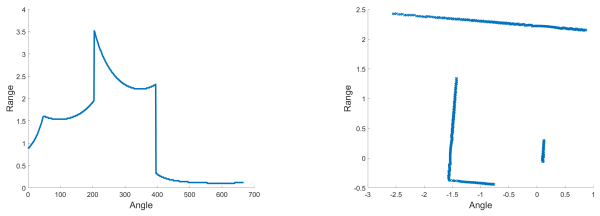
Mobile robot localization based on a 2D laser scan is a crucial problem in robotics and as been an area of research for years. Many existing solutions are based on probabilistic approaches like Monte Carlo or EKF localization. However, these methods are rather cost intensive, since they are usually matching the sensor data directly with the map representation. One approach to decrease the computational time for that is to extract features like lines or corners from the map as well as the sensor data and just match these features. Nevertheless, these features need to be defined and extracted manually which can be difficult with the increasing size of the environment.

This paper therefore presents a novel approach of implicitly learning features of an environment from sensor data using Convolutional Neural Networks. Two different ideas are developed, a 1D CNN that works on the plain scan data (ranges per angle) as well as a 2D CNN that takes a reconstructed image containing the scan data as x, y coordinates as an input. The experimental results showed that the 1D CNN outperforms the 2D CNN clearly and is able to estimate the robots position with an error of about $5cm$ in x and y direction.

1. Introduction

The problem of mobile robot localization has been an area of high interest for researchers ever since the beginnings of robotics. The reason for that is, that a robot has to know it's own position in order to find a goal in its environment. There are two ways to achieve localization, either relative through e.g. the odometry of the robot, or an absolute estimation of the position by incorporation of sensor data gathered from the environment [2]. The first opportunity leads to high uncertainty over time due to the fact, that the noise (e.g. motor malfunction or bumps) that is encountered during one time step is propagated over time. This makes it impossible to get an accurate estimate of the position in the long term, hence this option is usually just used in combination with an absolute position estimation for the robot. The absolute position estimation of a robot is usually done by modeling a probabilistic belief estimation of the robots position throughout the environment [19]. Over time, the methods used for that evolved from Bayes Filters including Markov [9] and Monte Carlo (Particle) [8][7][20] as well as EKF (Extended Kalman Filter) Localization [5]. All of the mentioned approaches maintain a belief state of the robot based on sensor (commonly a laser scanner, LIDAR) and control data executed by the robot, as well as a given map. The scan data is matched with the data from the map to estimate the current position of the robot. Since this can be very cost intensive with larger environments, one approach is to use landmarks instead of or in addition to map data [1]. These landmarks are usually specified by a position (x, y) and sometimes also an orientation (θ). The sensor measures the distance to surrounding landmarks and the robot tries to match the sighted landmarks with the ones stored in a *landmark-map*. Another approach is to extract features from the map as well as the scan data and map those features to obtain a pose estimation [5]. These features usually include lines and corners but can also have various types like e.g. pillars. The problem with all these approaches is that the landmarks as well as the features need to be defined and extracted which can be difficult and noisy. Since the data that is received by a common laser range finder is similar to a flattened image, this paper introduces the idea of extracting features from the sensor data using Convolutional Neural Networks (CNNs). Deep Learning had an increasing success within the last years especially in form of image classification using Deep CNNs [13]. The idea is here, that the Convolutional layers extract features out of an image and the subsequent fully-connected layers learn a matching between features and classes.

The system that is presented in this paper makes use out of these capabilities in order to detect features within the sensor data obtained by the robot. An example of such a



(a) 1D laser scan data. (b) 2D laser scan data.

Figure 1: Sensor data gathered by a Hokuyo URG-04LX-UG01 laser range finder in the ROS STDR-Simulator.

scan can be seen in Figure 1, which shows a scan from a Hokuyo URG-04LX-UG01 laser range finder generated in the ROS STDR-Simulator. Figure 1a shows the 1D raw scan as ranges per angle increment and Figure 1b shows a reconstructed 2D scan around the robot. This paper introduces a novel Deep Neural Network approach for both 1D and 2D feature detection as well as pose estimation from a laser scan based on Convolutional Neural Networks. The rest of the paper is organized as follows. In section 2 a few related work is presented including mostly relative pose estimations based on Recurrent Neural Networks. Section 3 presents the two networks that were developed for feature detection and pose estimation. In section 4 the experimental environment as well as the results of the two networks are presented. Finally, section 5 concludes the results of this paper and section 6 gives an outlook about the various work that the author will be working on subsequently.

2. Related Work

The area of Deep Learning has been of high interest within the last couple of years and due to that there have been a lot of achievements in a variety of applications using e.g. Convolutional or Recurrent Neural Networks (RNNs). The idea of Convolution has been around for decades now, early CNN approaches focused on topics like handwriting [16] and face recognition [14] as well as image and speech detection [15]. The major problem back in the time was the limited computational power as well as the limited amount of the memory that was available during that time. Today where computational power and memory are rather small problems due to high-end GPUs, ideas like handwriting recognition [6] or especially image classification [13] have been improved and extended. Recurrent Neural Networks went through a similar process and are used to solve problems that have a temporal component such as dynamic systems [10] or Natural Language Processing [17].

So far there haven't been a lot of approaches of Deep Neural Networks in the field of mobile robot localization. One of them is presented in [3], where the authors use an RNN to predict the pose (x, y, θ) of the robot for the next time step $t + 1$ by using the controls that are currently (t) applied to it. This approach works just well in case there is no uncertainty in the environment and the robot always ends up in the same state when executing a certain action. Since this is an assumption that is impossible to make about the real world, this approach is not feasible for mobile robot localization because the error introduced by e.g. just a small bump would propagate over time and lead to a completely wrong pose estimation. Another approach based on RNNs is introduced by the authors of [11]. The main idea of this paper is to estimate the robots *room position* based on scan data from a laser scanner. This is basically a 15-Class Classification problem that matches 2D 360-degree scan data with map data from the respective rooms to determine in which room the robot currently is. The authors achieve a high accuracy on this classification which shows that it is possible to work with 2D laser scan data. Nevertheless it is insufficient for mobile robot localization since this system just outputs a room where the robot is and not a pose estimate which is necessary in order for the robot to navigate to a specified goal in the environment and to avoid obstacles. One idea that is rather similar to the attempt made in this paper, is introduced in [18]. Their system uses 3D LIDAR data as an input, converts it into a 2D depth image and uses this to feed a CNN. The CNN detects features in the environment and based on the difference of these features in two subsequent scans, the distance traveled by the robot is estimated. The output of the network is the difference in x and y between time t and $t + 1$. Although the authors were able to achieve a higher accuracy than the standard, Iterative Closest Point (ICP), used for this problem, this is just a relative estimation of the robots position and needs some kind of initial position. Furthermore the orientation of the robot was not incorporated in this system which is however absolutely necessary for a mobile robot in order to interact with its environment.

Due to the drawbacks of the described approaches in DNN based mobile robot localization, the system that will be presented in the next sections is indeed similar to the two latter approaches presented, however it is using 2D instead of 3D LIDAR data and estimates an absolute position as well as orientation of the robot at each time step independently from each other. By using 2D sensor data, the input dimensionality is reduced from about 700000 (Velodyne HDL-32E, 3D LIDAR) to about 667 dimensions (URG-04LX-UG01, 2D LIDAR), however this is still sufficient for an accurate pose estimation as past work in this area has clearly shown by using common approaches such as Monte Carlo [20] or EKF localization [5].

3. CNN for Localization

The two CNNs developed in this paper for learning features from a given map as well as an association between

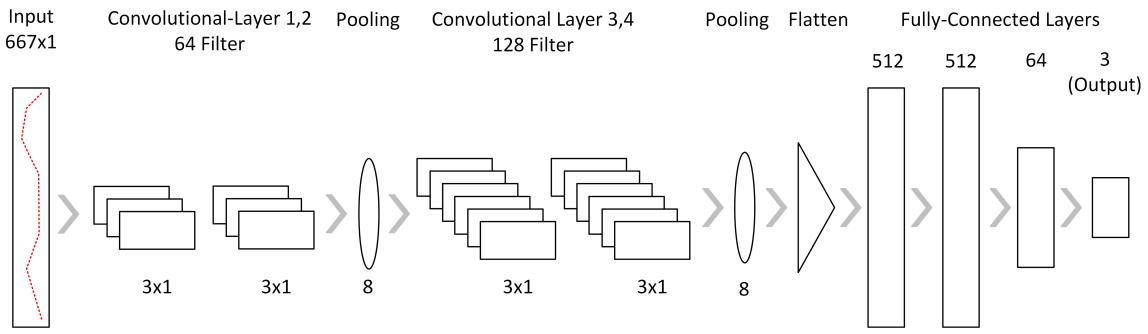


Figure 2: The architecture of the Deep CNN used for the 1D input data.

those features and the pose of the robot are generally working with the exact same input data, a 667 dimensional scan from a Hokuyo URG-04LX-UG01 laser range finder. It has a maximum range of $5.6m$ and a minimum range of $0.02m$, as well as an angular range of $240°$. The difference between the input of the two models is the representation which is described further in the following subsections. One remark that the author wants to make at this point is, that neither of these two approaches explicitly models any uncertainty in the scan or the estimated pose. It was neglected for simplicity at this point but will be part of the future work of the author on this topic, since a probabilistic estimation of the pose is common and important in mobile robot localization to propagate the uncertainty from sensor and odometry data.

3.1. 1D-Input CNN

The 1D-CNN works with the plain data obtained by the sensor. A visualization of this data as range per incremented angle can be seen in Figure 3. This image visualizes the ranges measured by the laser range finder for one time step, it also includes the possible features that could be extracted or found by the CNN. These features could be e.g. lines, corners or other objects with a specific structure like pillars. The Convolutional layers in the network are supposed to detect and learn these features, the subsequent fully-connected layers should then learn a mapping from these features to the pose of the robot (x, y, θ).

The structure that was chosen for the network is visualized in Figure 2 and includes 2 times 2 Convolutional layers, each pair followed by a max-pooling layer of size 2. The Convolutional layers include 64/128 filters of size 3x3. The flattened output of these layers is then used as an input to 4 fully-connected layers, the last one being of size 3 since it outputs the estimated pose of the robot as x, y, and θ. The initial input as well as the output of each fully-connected layer is normalized. As an activation function for the Convolutional layers was ReLU chosen, for the fully-connected layers Sigmoid. Initially a dropout layer after each Convolutional pair and each fully-connected layer was found useful but with increasing size of input examples the difference between the training and validation loss was so low that they weren't helpful anymore. This will be discussed further in the experimental results in Section 4. The proposed architecture was found to be the most promising one since it led to the best results, although a lot of other structures were considered, especially using more convolutional layers with a smaller pooling size. However, none of these architectures led to better results compared to the one shown in Figure 2, although this doesn't mean that there is no improvement possible in the chosen structure of the CNN.

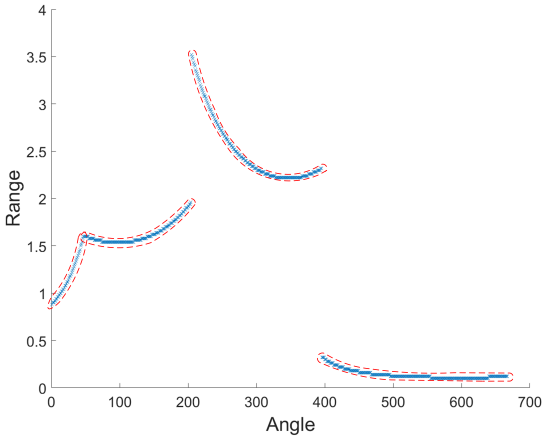


Figure 3: The 1D raw input data for the 1D CNN, where the measured ranges are visualized as blue crosses and the possible features (lines) are surrounded by dashed red lines.

3.2. 2D-Input CNN

In addition to the previously described 1D-CNN, the author proposes a second approach based on the idea to reconstruct a local 2D m̈apäround the robot based on the 1D sensor input. This m̈apïs then fed into a 2D-CNN, which treats it as a normal image except for the fact that it just contains values of either 0 or 1. To implement this, an im-

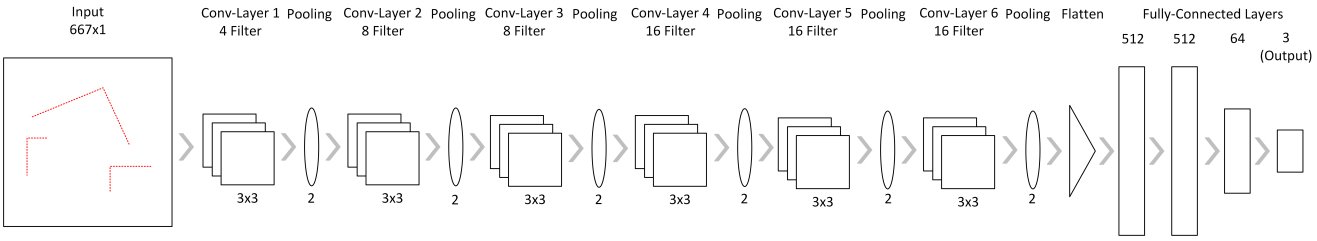


Figure 4: The architecture of the Deep CNN used for the 2D input data.

age size of 384x384 was chosen. This is a common size for CNNs, since it keeps a valid size after pooling layers without loosing information until it finally reaches a minimum size of 3x3. Alternatively a smaller image size with the same properties, like e.g. 224x224, could be chosen. However, a smaller image size makes the reconstruction harder and more inaccurate since this is more or less a discretization of the input. The values of the images are determined by first calculating the respective Euclidean coordinates x and y from the ranges r and angles α. This is done using simple geometry:

$$x_i = sin(\alpha_i) * r_i \tag{1}$$

$$y_i = cos(\alpha_i) * r_i \tag{2}$$

Where x_i/y_i denotes the x/y value for the i-th example and r_i the range measured for the respective angle α_i. Since this just creates coordinates surrounding the robot at the origin, these x and y values need to be normalized into the images' pixels coordinates with the origin being at the center of the image. The normalization works as follows:

$$imgX_i = iround(x_i * \frac{size}{2 * max} + \frac{size}{2}) \tag{3}$$

$$imgY_i = iround(y_i * \frac{size}{2 * max} + \frac{size}{2}) \tag{4}$$

Here the image coordinates $imgX_i/imgY_i$ are calculated based on the euclidean coordinates x_i/y_i, the $size$ of the image (384) as well as the maximum range max of the sensor. The result is then rounded to the next integer to fit a pixel in the image. This normalizes all values in a range of [0-383] to fit in the image. The result of such a transformation can be seen in Figure 1b. This image shows not only the scan points but also the respective features that could be fount by the CNN, in this case lines.

In order for the CNN to detect these features the network contains 6 Convolutional layers as shown in Figure 4. The amount of filters per layer was here chosen a lot less than in the previously described 1D case, since the input dimensionality is now about 200 times higher than before and a high number of filters led to a tremendous increase of the runtime. Other than that the architecture is pretty similar,

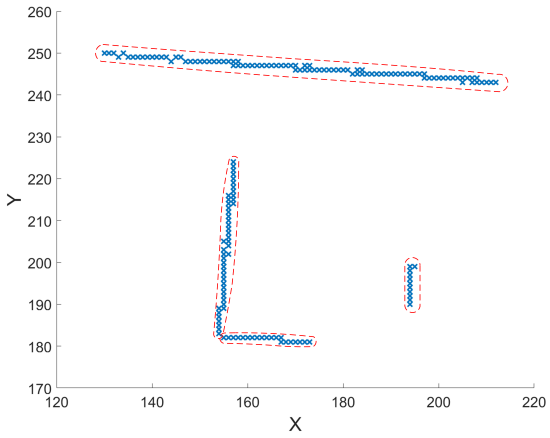


Figure 5: A reconstructed 2D image of the 1D range data obtained by the laser range scanner, where the measured ranges are visualized as blue crosses and the possible features (lines) are surrounded by dashed red lines.

except for the filter size which changed from 3x1 in the one dimensional case to 3x3 for two dimensions.

4. Experiments and Results

This section evaluates the results obtained by training the presented networks. First, the used environment including simulator, robot and map are specified. Afterwards a comparison between a simple fully-connected network and the 1D/2D CNN developed in this paper is presented. Finally the author elucidates the capabilities of the CNNs by adapting the network to another map.

4.1. Setup and Environment

The simulation environment that was used for collecting the networks input (sensor/pose) data is the STDR Simulator that is part of the Robot Operating System (ROS), since it is an easy to use 2D simulator. The map used for the experiments is shown in Figure 6, a simple map with some sparse obstacles and a size of 15.50x$14.92m$. The black parts of the image denote obstacles in this case and the white parts identify the free space of the environment. This map is part of the STDR package and was chosen because

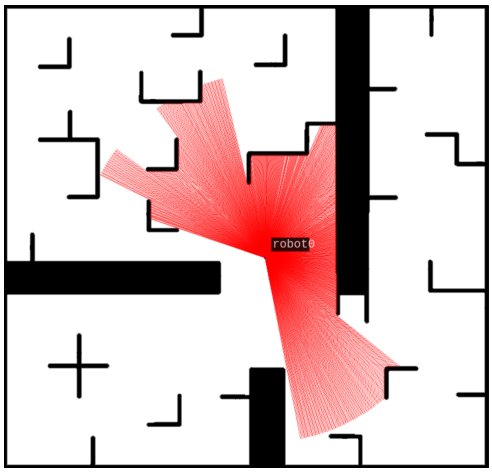
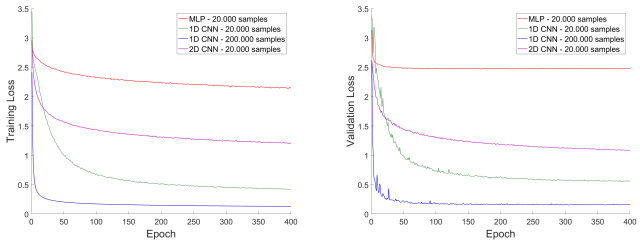


Figure 6: The map and robot used for the experiments, with the red lines being a visualization of the robots laser scanner.

of its structure compared to other maps that contained e.g. just simple rooms. The robot used for the experiments is based on a simple robot model which is also part of STDR. It is equipped with a Hokuyo URG-04LX-UG01 laser range finder that outputs a 667 dimensional laser scan of its environment with a frequency of $10Hz$. It has a minimum range of $0.02m$ and a maximum range of $5.6m$, as well as an angular range of $240°$. The robot with its respective sensor reading is shown in Figure 6 as well. To generate the input data for the network, the robot was spawned randomly at a position sampled uniformly from the environment. The data is then split into training (90%) and test data (10%). All networks are learned using the Adam optimizer and the mean absolute error as loss.

4.2. MLP vs. 1D/2D CNN

In the first experiment, the basic functionality of the CNNs is shown by comparing it to a standard Multi-Layer Perceptron (MLP) [12], since this is known to be a able to universally approximate data. The MLP in this case consists of 3 fully-connected layers with 512 hidden units, 1 layer with 64 and the output layer with 3 hidden units corresponding to x, y and θ, respectively. Batch normalization as well as a dropout of 0.25 were implemented for the MLP, since its biggest problem was to generalize and not overfit the training data, however this decreased the convergence rate. The *ReLU*-function worked best as activation and The MLP is compared to a 1D CNN learned with a dataset of 20.000 samples, one with 200.000 samples as well as a 2D CNN. The structure of the 1D CNNs is here the same, the difference is just the sample and batch size. The parameters used for the training of the networks are listed in Table 1 and the results of the training/validation error can be seen in Figure 9.



(a) Training Loss. (b) Validation Loss.

Figure 7: A comparison between a MLP, the 1D CNN trained with 20.000 and 200.000 samples as well as the 2D CNN.

The results show that both CNNs clearly outperform the MLP. The MLP has huge problems with the generalization, after about 50 epochs the validation error gets stuck around a value of 2.5 and does not decrease any longer. Both 1D CNN variants on the other hand outperform the 2D CNN by at least a difference of 0.7 in the training and 0.5 in the validation loss. The 2D CNN turns out to be much harder to learn than the 1D CNN, this could be due to the higher dimensional input. Sadly it was not possible to run the 2D CNN for longer than 400 epochs, since one epoch took around 60-100 seconds. Since already the 1D CNN version with 20.000 samples worked significantly better than the two previously mentioned networks, the author focused on the 1D CNN for further improvements and therefor learned the network as well with a dataset of 200.000 samples. Although one full epoch takes now instead of 5 about 50-60 seconds, the increased sample size by a factor of 10 really led to a significant improvement of the networks performance. The network now converges towards a validation error of about 0.15 after 400 epochs and since the loss here is the mean absolute error, this means, that the error in the position is less than $0.15m$ because the error is composed of the error in x, y and θ. This is a very good result for mobile robot localization without the incorporation of the robots odometry information. Especially because the mean differ-

	MLP	1D CNN	1D CNN	2D CNN
Epochs	2000	5000	400	400
Batch Size	100	100	400	100
Learn Rate	0.01	0.006	0.006	0.01
LR-Decay	0.005	0.0	0.0	0.005
Dropout	0.25	0.1	-	0.1
Samples	20.000	20.000	200.000	20.000

Table 1: The training parameters for the four different models that are part of the first experiment.

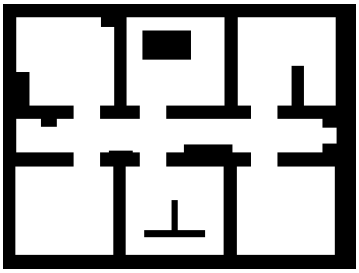
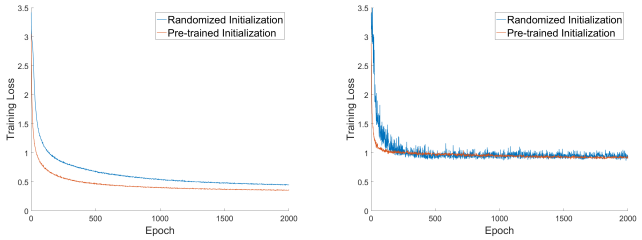


Figure 8: A map with simple rooms and fewer obstacles used for the second experiment.



(a) Training Loss. (b) Validation Loss.

Figure 9: The training (*a*) and validation (*b*) of the 1D CNN learned with randomized and pre-learned data on the map shown in Figure 8.

ence between the training loss in Figure 7a and the validation loss in Figure 7b is 0.0131, so almost zero. This shows that it is indeed possible to learn an implicit representation of a map using Convolutional layers and with their output learn a mapping to the pose of the robot in the environment.

4.3. Adaption to a 2nd map

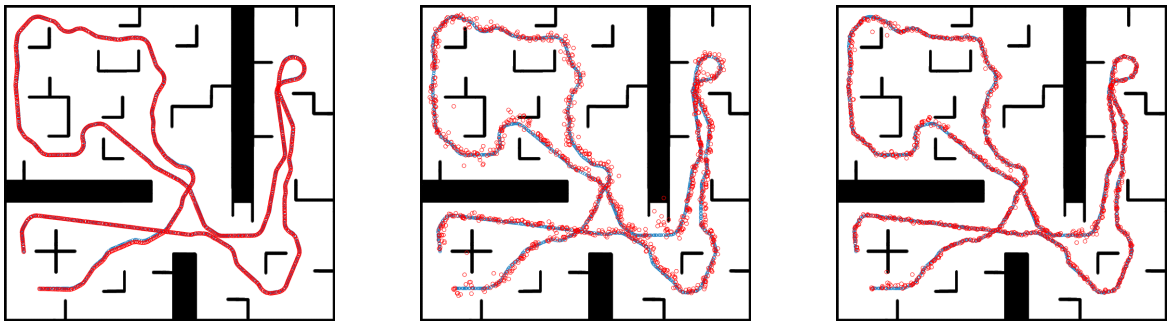
The second set of experiments that the author conducted for this paper includes the use of the proposed 1D CNN for learning of a new environment which is pictured in Figure 8. The map is a lot simpler than the previously introduced one. There is not as much structure or obstacles in the environment as it was the case before. This was chosen on purpose to see the influence of the map shape on the learning process. For this experiment two 1D CNNs of the same architecture are compared. The only difference between them is, that one of them is initialized with randomized weights while the other one loads the weights from a model that was pre-trained for 2000 epochs on the first map using a dataset of 20.000 samples. This experiment should show on the one hand if the representations or features learned on one map are actually useful when learning a new environment and on the other hand it should show if the network architecture is just specifically designed for one map or if it is generalizable for more. The network is exactly the same as the 1D CNN that was introduced in the previous Subsection and learned with a dataset 20.000 samples. The training loss of the networks can be seen in Figure 9a while the validation loss is presented in Figure 9b. These plots clearly indicate that the network is able to learn a pose estimation not only for the first map but also within this new environment, although the loss is not as good as in the first map. This could be due to the lack of structure and obstacles. To investigate these impacts of the environment on the learning progress will be part of the future work for this paper. Furthermore the results unveil an improvement of the networks convergence with pre-initialized weights compared to randomly initialized ones. Especially in the beginning, both the training and validation loss decrease faster. This shows, that the network actually learns a features representation from one environment that can be transferred to another one.

4.4. Real Time Localization

For the last experiment in this paper, the initial sparse obstacle map is used again, since it provides better structural properties than the map introduced in the Subsection before. In the following the performance of the 1D CNN is compared against a standard implementation of the Monte Carlo localization [19] from the ROS package amcl. To decrease the error in the pose estimation, another approach, *Learning Ensembles* [4] are introduced for this experiment. In addition to using 1 trained model of a 1D CNN for the prediction, multiple models are trained and evaluated. The overall prediction is then simply the mean between the predicted values for one time step. This should make the pose estimation more robust and makes it also possible to calculate a covariance from the predictions. The CNNs used for this experiment are all based on the architecture presented in Subsection 4.2 and trained with a dataset of 200.000 scans. To generate the input data for the CNNs as well as the pose estimation using amcl, a robot was steered through the environment for about 1500 time steps.

The results are presented in Figure 10, which illustrates the position estimations by amcl (10a), the single model CNN (10b) as well as the multi model CNN (10c). The quantitative results of the experiment are shown in Table 2 and include the mean as well as the standard deviation of the pose estimation error (x, y, θ) for all three methods. The error in the orientation θ is here calculated as the difference between the *sinus* values of the true orientation and the predicted one.

The Monte Carlo (MC) localization from amcl definitely outperforms both CNN approaches. This is clearly visible in the visualization of the pose estimates in Figure 10a, where the amcl version is almost exactly following the underlying true data. It can be also seen in Table 2, where the difference in the mean error of the position (x, y) between the Monte Carlo localization and the single CNN model is about $0.075m$. The multi model CNN on the other hand



(a) Monte Carlo (b) Single 1D CNN (c) Multi 1D CNN

Figure 10: The resulting position estimations in the map based on amcls Monte Carlo localization (a), a single (b) as well as multiple 1D CNNs trained with 200.000 samples. The underlying blue circles indicate the true position of the robot while the red circles illustrate the position estimation of the respective technique.

improves the single model version significantly. The difference in the positions mean error compared to Monte Carlo localization is here just around $0.018m$. Figure 10c illustrates this as well, there are just very few pose estimations really far away from the true path compared to the single model approach in Figure 10b.

This shows that the CNN approaches are indeed comparable with a state of the art solution like the Monte Carlo localization. Especially the results of the multi model CNN are impressive, since the positional error in x and y direction is just around $5cm$. That is a really good value for a robot. It also indicates that the *Ensemble Learning* decreases the error of the system.

Although the predictions for this experiment were calculated off-line, the 1D CNN was also tested as an on-line localization for a robot in a simulation. The results of this test run showed, that it is possible to use this localization method in real-time applications, since neither the single nor the multi model approach had a problem to maintain an output frequency of $10Hz$.

	Monte Carlo	Single CNN	Multiple CNNs
Mean x	0.023	0.097	0.052
Mean y	0.037	0.113	0.044
Mean θ	0.009	0.052	0.047
Std x	0.017	0.083	0.063
Std y	0.028	0.206	0.056
Std θ	0.011	0.127	0.155

Table 2: The mean absolute error as well as the respective standard deviation (std) of the pose estimation in meters.

5. Conclusion

This paper introduced a novel approach for mobile robot localization in well-known environments based on a CNN. A very promising 1D CNN was presented and the results have shown that is comparable with a probabilistic state of the art localization method like Monte Carlo. Especially the combination of the predictions from multiple CNNs of the same architecture increased the performance significantly. In a test run the positional error of the prediction was found to be less than $5cm$, compared to the Monte Carlo localization that achieved an error of around $3cm$, this is a really good result for mobile robot localization. Furthermore the results of the experiments also showed, that the features learned by the network are applicable not only for one environment but improve the convergence rate when being applied to a new domain. This was demonstrated by comparing a pre-trained network with a randomly initialized network on a new map. This experiment also revealed, that the architecture of the 1D CNN is generalizable with regards to the environment.

In addition to the 1D CNN, a two dimensional version was also presented that takes a reconstructed 2D image of the sensor data as an input. However the results were not as good as the author initially expected, especially compared to the high computational cost of learning the network due to the high dimensional input (384x384).

Nevertheless the presented CNN approaches for mobile robot localization were close to a method like Monte Carlo localization and this shows, that it is possible to learn a pose estimation for a robot from the environment data.

6. Future Work

There are several different topics the author would like to address in the future, but probably the most important one is a profound breakdown of the network to analyze its

strengths and weaknesses. To achieve this, the interim results of the convolutional layers will be examined. The networks will be also trained on several other maps with different structural properties as well as obstacle densities to detect possible difficulties the network has with these environments.

Another approach to decrease the convergence time of the network will be to use an Autoencoder to pre-train the Convolutional layers and learn a representation of the environment seperately from the association between features and the output pose.

In mobile robot localization it is common to combine sensor data with the odometry information of the robot to obtain a more accurate pose estimation. So far, the networks presented in this paper are neither considering the odometry information of the robot nor the time component. This will be an area of future research for the author in form of the integration of the presented work into a RNN or LSTM to improve the learning of the network dependent on the time and additional odometry information received by the robot.

The last important extension that needs to be developed and implemented is a model to incorporate and propagate the uncertainty from the sensor data to the pose estimation. This is a very important part of common probabilistic localization approaches, since no sensor or odometry information is perfect this uncertainty propagates to the pose estimation of the robot. Usually multiple state estimations of the robots pose and covariance are maintained and updated simultaneously. The author considers a similar approach in form of learning an association not only from features to a specific pose but also to a covariance. Another approach for this would be to learn a mapping from features to a Probability Density Function over the environment.

References

[1] M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, 13(2):251–263, Apr. 1997. 00707.

[2] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe. Mobile robot positioning-sensors and techniques. Technical report, DTIC Document, 1997. 00726.

[3] H. Brahmi, B. Ammar, and A. M. Alimi. Intelligent path planning algorithm for autonomous robot based on recurrent neural networks. In *2013 International Conference on Advanced Logistics and Transport*, pages 199–204, May 2013. 00008.

[4] L. Chen. Learning Ensembles of Convolutional Neural Networks. 00000.

[5] L. Chen, H. Hu, and K. McDonald-Maier. EKF Based Mobile Robot Localization. In *2012 Third International Conference on Emerging Security Technologies*, pages 149–154, Sept. 2012. 00008.

[6] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional Neural Network Committees for Handwritten Character Classification. In *2011 International Conference on Document Analysis and Recognition*, pages 1135–1139, Sept. 2011. 00186.

[7] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1322–1328 vol.2, 1999. 01481.

[8] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2, 1999. 01164.

[9] D. Fox, W. Burgard, and S. Thrun. Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 25(3):195–207, Nov. 1998. 00513.

[10] K.-i. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993. 00596.

[11] A. Frster, A. Graves, and J. Schmidhuber. RNN-based Learning of Compact Maps for Efficient Robot Localization. In *ESANN*, pages 537–542, 2007. 00008 bibtex: forster_rnn-based_2007.

[12] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 13852.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 10372.

[14] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, Jan. 1997. 01375.

[15] Y. LeCun, Y. Bengio, and others. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. 00929.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, Dec. 1989. 02195.

[17] T. Mikolov, S. Kombrink, L. Burget, J. ernock, and S. Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531, May 2011. 00399.

[18] A. Nicolai and G. A. Hollinger. Deep Learning for Laser-Based Odometry Estimation. 2017. 00000.

[19] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005. 06654.

[20] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99–141, May 2001. 01867.