LiDAR-Based Robot Pose Estimation with Deep Learning

Dennis Lindt
University of Oldenburg
dennis.lindt@uol.de

Svenja Schuirmann
University of Oldenburg
svenja.schuirmann@uol.de

Sven Schultze
University of Oldenburg
sven.schultze@uol.de

1 INTRODUCTION

In many robotic systems, knowing the position of the robot is an essential requirement. It helps the system to understand how close it is to complete a task, and what to do next. There are two main approaches to robot localization: 1) using indoor positioning systems (IPS), and 2) using robot-mounted sensors such as light detection and ranging sensors (LiDAR) and odometry. IPS's make use of different technologies, such as signal beacons with fixed positions to triangulate the current position [7], or magnetic sensors that react to abnormalities in the magnetic field inside buildings [4].

While these IPS's can reach high accuracy, they are often expensive and difficult to install. Because of this, robot-mounted systems are more commonly used. For example, a combination of LiDAR data and odometry can yield information about relative position changes over time, which can be used to create a map of the robot's environment.

An example for such a localization system is Simultaneously Localization and Mapping (SLAM). SLAM is stateful, meaning that it uses previous odometry, location and sensor data to make predictions about the robot's current location. Over time, the system generates a map of its surroundings, which allows it to recognize places it has visited before. However, this approach can lead to problems and errors when the robot is disturbed. For example, if a robot is moved, picked up or shifted in an unexpected way. Another source of error could be its dependency on odometry. Odometry is often inaccurate because it relies on assumptions about physical properties of the robot and the underground [1]. This make measurements unreliable and can falsify the position of the robot on the map. This error can get incrementally worse [8].

A stateless localization system is not prone to these errors. By using nothing but LiDAR data as input, the system does not consider its previous states to estimate the robot's location. Thus, the robot can be moved without issue. Further, the system does not depend on odometry, which is often faulty and leads to errors. This paper presents a stateless localization system based on LiDAR data that uses a convolutional neural network as its backbone. The network extracts features from the LiDAR data, detecting walls, corners, and objects. Then, it estimates the robot's location based upon its findings.

2 RELATED WORK

There are several ways to estimate a robot's location based on robot-mounted sensors. One such technique is Simultaneously Localization and Mapping (SLAM). SLAM generally uses data from optical sensors (LiDAR or depth cameras) and odometry to localize the robot. While doing so, it slowly generates a map. Additionally to the optical data, SLAM uses odometry as an initial guess how the position has changed. Then, changes in data from the depth sensing systems yield information to find a more accurate estimation based on the generated map. One of many different implementations of SLAM is GMapping [3].

Very recently, a paper from Zhao et al. [10] proposed an approach of localization based on a 3D model and deep learning. With this method, a convolutional regression network was trained with simulated LiDAR data from a 3D model. The network was tested on real and synthetic LiDAR data with the result that the real images achieved lower accuracy. The network was trained with known location and used this to estimate the position of the robot later on. This approach is very similar to our own proposed system, but uses more expensive hardware, and generates an image from the LiDAR data, which is then processed by a two-dimensional convolutional network, instead of processing the LiDAR data directly.

Another approach is the usage of the Monte Carlo Localization (MCL) and a deep-learned distribution [9]. The use-case of this project is robot localisation in an large scale environment. The method uses learning-based and filtering-based localization with MCL. The localization is calculated with a deep-probabilistic model. Then, a precise recursive estimator processes the estimated pose conferring to the geometric alignment. A case study was done to verify the method, showing that it can localize a robot in 1.94s with an accuracy of 0.75m in an large environment up to 0.5 km^2.

A different paper [2] uses landmarks to localize a robot. It assumes that the robot can recognize landmarks and calculate relative distance. The position of the robot is then estimated with a known map of landmarks. This approach does not use any odometry data. Tests of this algorithm showed promising results.

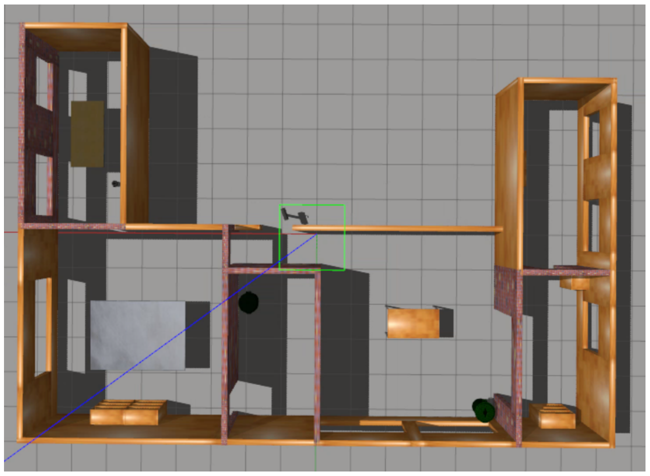


Figure 1: Simulated apartment in Gazebo

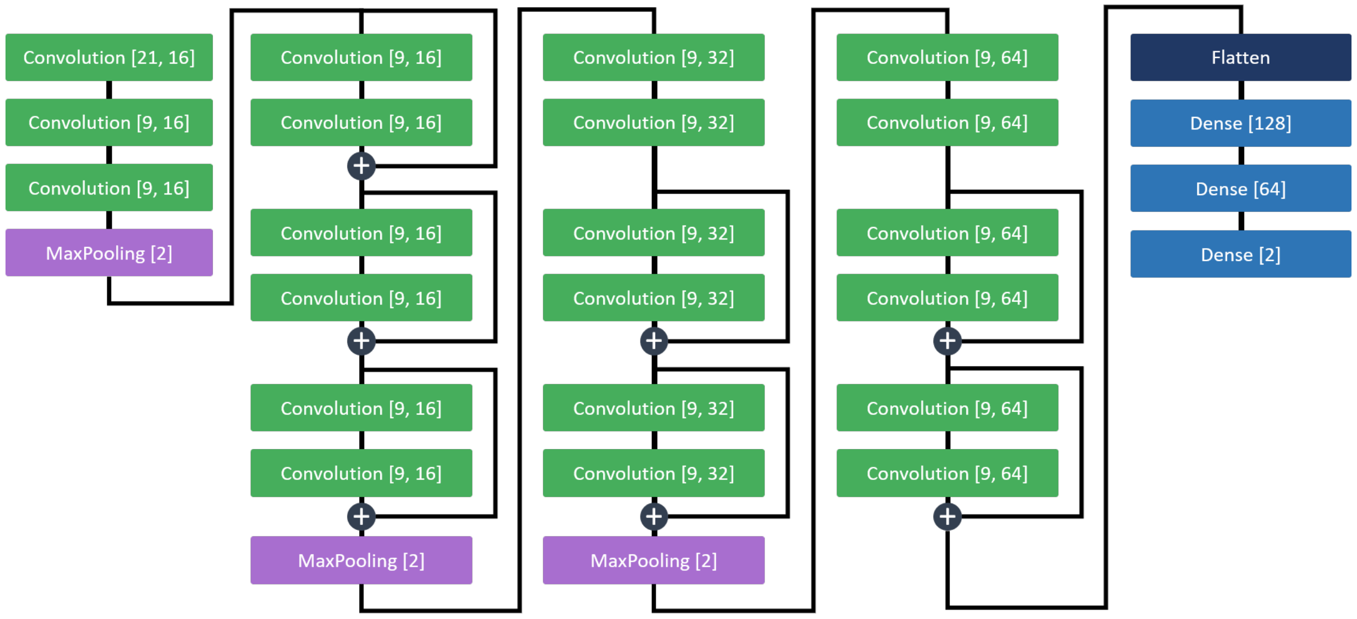


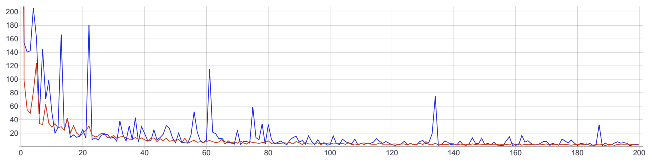
Figure 2: Residual Architecture. A Convolution [S, K] layer has K kernels of size S, a MaxPooling [S] layer has a pooling size of S, and a Dense [K] layer has K neurons. The one-dimensional convolutional network architecture follows the idea of residual connections in two-dimensional convolutional networks like ResNet [5].

3 OUR APPROACH

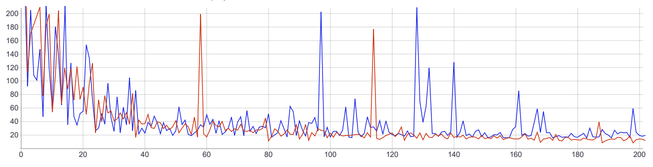
To start off, we created a dataset with a virtual apartment in the robot simulation environment Gazebo (Figure 1). The appartment measures roughly 12 by 16 meters. By probing random coordinates in this apartment with a simulated LiDAR system, we obtained 360 distance values each. Another possible approach might be the use of a grid-based probing pattern, however we do not believe that performance is impacted by this. Then, the distance vectors are rotated in such a way that each coordinate has one unambiguous representation. In the end, we collected 50000 distance-coordinate-pairs.

Through experimentation, we arrived at a convolutional neural network architecture inspired by ResNet [5]. The network comprises seven residual blocks, and three normal convolution blocks. Each convolutional layer employs a cyclic padding, meaning that values from the other end of the vector are concatenated as padding. This ensures that the values at the ends are processed in the same way as in the middle, which is possible because the distance values from the LiDAR themselves are cyclic. Additionally, each convolutional layer is followed by a batch normalization instead of biases before the activation ReLU. This allows the network to converge faster during training, and provides regularization to prevent overfitting.

Figure 3 compares the residual architecture (3a) to a more shallow architecture (3b) with only four convolutional layers. 3a shows a faster and more stable convergence than 3b, and ends up at a lower error after 200 epochs. Additionally, the residual network performed better in inference, partially due to the lower number of trainable params, being 652946 compared to 857218 in the shallow network.



(a) Residual Architecture



(b) Shallow Architecture

Figure 3: Convolutional architectures in comparison (Mean Squared Error over 200 epochs). Training loss in red, validation loss in blue.

We trained the residual architecture with the previously generated dataset of the simulated Gazebo apartment. We used the GPU-accelerated Environment Google Colaboratory with a Tesla T4 for the process. For this, we split the dataset into 40000 samples for training, and 10000 for validation. Then, we used the optimizer Adam [6] at a learning rate of 0.001 and a batch size of 50 to train it for 500 epochs. Afterwards, the network arrived at a mean squared error of roughly 0.0006, a mean average error of 0.0183, and a mean absolute percentage error of 2.9%. In real units, this amounts to an average error of 2.8cm, a top 10% error of 6.5cm, and a top 1% error of 11.2cm.

Figure 4 visualizes the worst performing coordinates from our dataset on this model. It shows that the model performs worse in specific areas of the apartment, especially close to walls. We argue that this is a result of the generation method of our dataset. By randomly teleporting the robot to random coordinates within the apartment, it might have occasionally glitched into walls or objects, which would insert corrupt LiDAR scans into the dataset. Another interesting phenomenon are errors in the top-left outside of the apartment, at roughtly the same distance to the wall as the width of the smaller rooms. This shows that the network estimates the position based on the distance to detected walls, and maybe even more notably on the distance and position of corners. This decision process seems to slightly alter the prediction at this spot, confusing it with rooms with similar distances to walls and corners.

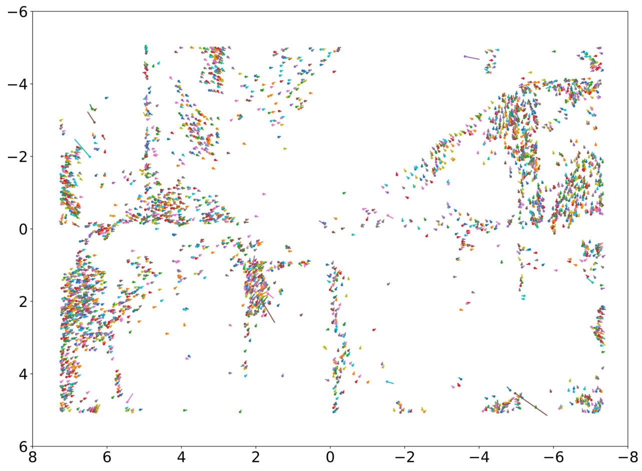
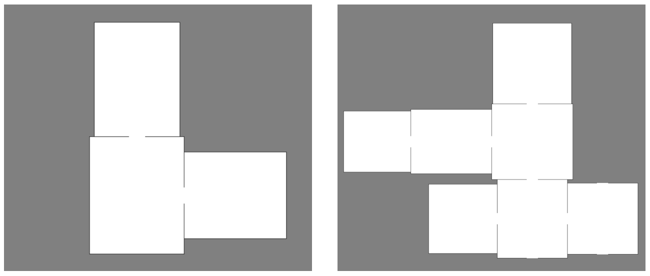


Figure 4: Coordinates with highest error (top 10%) from our dataset. Ground truth is represented by a small point, a line represents the prediction as a vector.

The performance on the training and test dataset was not significantly different, which suggests that overfitting is not a problem that occurs in our scenario. Since both datasets are shuffled together and intersect spatially, their samples are very similar, so the network has no problem to generalize.

4 EVALUATION WITH DIFFERENT APARTMENT LAYOUTS

In order to test the performance of the neural network in different environments, one virtual apartment does not suffice. Therefore, we created a 2D apartment layout generator that also calculates new datasets for the neural network. Our self-written script generates 2D images of basic random apartment layouts with a set number of rooms of slight varying sizes. Inside the apartment, we can set a robot object which emits ray lines similar to a LiDAR system. Like in gazebo, the robot is placed in a random position inside the apartment, and 360 distance values around the robot are collected. Because of the different layouts and therefore increasing complexity, we simulate new situations for the robot and check their influence on our system's accuracy. For this example, we created two apartments, one with 3 rooms (Figure 5a) and one with 7 rooms (Figure 5b), and compared the precision between the two. Both apartments were drawn on an image with 1920 width and 1080 height in pixels, so the position and distance calculation stays the same. The white and black pixels are corresponding to drive-able area and walls, respectively. The LiDAR is simulated with a ray-marching algorithm, measure distances in 360 different directions from the robot. In the end, we generated a dataset containing 50000 distance-coordinate-pairs for both generated apartments.



(a) Small apartment (3 rooms) **(b) Large apartment (7 rooms)**

Figure 5: Images of generated apartment layouts

In Figure 6, we compare the mean absolute percentage error between the two apartments for 500 epochs. The red graph shows the error of the small apartment which converges faster than the large apartment. It can be said that complexity and size of an apartment affect the learning speed and final performance of our neural network. Nevertheless, after long enough training, the neural network manages to achieve good results for an apartment with increased size and complexity. In this comparison, the rooms are empty and no objects are present.

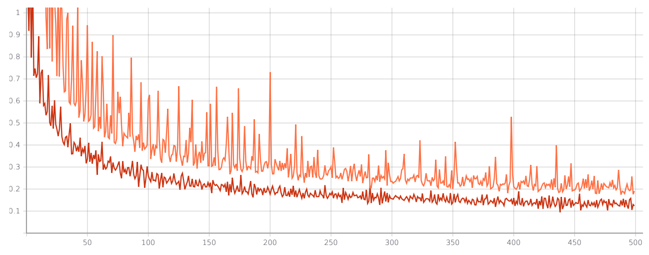


Figure 6: Mean absolute percentage error of small (red) and large (orange) apartment over 500 epochs of training

5 APPLICATION IN THE REAL WORLD

In this paper, we present an approach to robot pose estimation based on deep learning. This approach is able to estimate the position of a robot in a two-dimensional space based on two-dimensional distance values obtained via LiDAR. When training the neural network, the presence of matching ground truths in the form of absolute coordinates is crucial. However, in the real world, this is generally not the case.

To solve this problem, we used the SLAM implementation of GMapping in ROS to generate a map (see Figure 7). Then, we applied our dataset generation algorithm from section 4 to generate

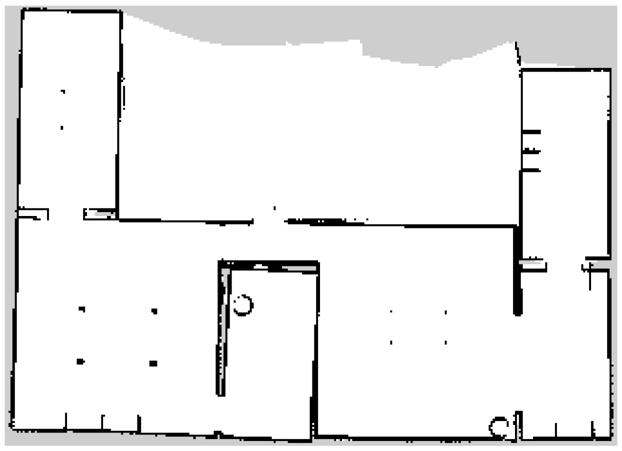


Figure 7: Map generated with the ROS implementation of GMapping [3]

LiDAR-coordinate pairs, and trained a model based on that data. We achieved similar performance as before, though this model was obviously flawed because it carried the inaccuracies of the GMapping algorithm additionally to its own. However, it might be possible to achieve better real-world performance by manually cleaning up the by GMapping generated map, or creating it from scratch. For example, one could fix obvious inaccuracies such as tilted rooms or doorways.

6 DISCUSSION AND CONCLUSION

Most robot localization methods are stateful and often rely on odometry or temporal data of previous states to estimate the robot's position and movement. This approach works well as long as sensor data is accurate, and the robot is not moved in unexpected ways. However, if the robot is picked up, for example, the algorithm might not be able to recover and will continue to predict very inaccurate poses.

With this problem as our initial motivation, we created a stateless approach to pose estimation, using a convolutional neural network to extract features from LiDAR data and to predict the robot's location subsequently. To find its limitations, we tested this approach on several apartment layouts of varying complexity and implemented a visualization technique to reveal its shortcomings.

Since the approach requires a dataset of absolute spatial coordinates paired with corresponding LiDAR scans, we proposed a way to apply it in the real world, using the GMapping algorithm to initially create a map of the apartment. Based on this, we were able to create a dataset with absolute coordinates and train a model. While this model does not beat the accuracy of the GMapping localization, it could be used to find the robot's location after it was picked up and moved, or disturbed in any other way. Also, the first step of our approach could be performed on a machine with a better hardware configuration, for example, more expensive sensors to generate a very accurate dataset. Then, the model could be trained, and deployed on a cheaper system, with only a LiDAR and a minimum of computational capability as a requirement.

On the other hand, our approach also has several shortcomings. For example, it is strongly affected by dynamic changes in the environment, like moved furniture. Whenever a change occurs, the model has to be arduously retrained with an augmented dataset to ensure high accuracy. Further, the model is bound to one specific environment, and cannot be transferred to another apartment with little effort, like most SLAM algorithms can. Also, we have constrained our model to a specific robot orientation to improve performance. This means the LiDAR data needs to be rotated according to rotational sensors. However, this problem might be overcome with a more complex model architecture and a larger dataset.

In the future, it might be interesting to apply our approach to the real world. There is still much work to be done in the practical realization. For example, different LiDAR types can be evaluated, and mapping or 3D-scanning approaches can be experimented with to generate the initial dataset. Maybe, a highly accurate IPS can be used instead, for example, based on Bluetooth beacons or surveillance cameras that are temporarily installed in the beginning. Also, the robustness of our approach has to be evaluated further. One question is, whether perturbations like humans walking in the rooms disturb the model's estimations, and how this can be fixed.

REFERENCES

[1] Mordechai Ben-Ari and Francesco Mondada. 2017. Robotic Motion and Odometry. In *Elements of Robotics*. Springer International Publishing, 63–93. https://doi.org/10.1007/978-3-319-62533-1_5

[2] Margrit Betke and Leonid Gurvits. 1997. Mobile robot localization using landmarks. *IEEE transactions on robotics and automation* 13, 2 (1997), 251–263.

[3] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. 2007. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics* 23, 1 (2007), 34–46.

[4] Janne Haverinen and Anssi Kemppainen. 2009. Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems* 57, 10 (2009), 1028–1035.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[6] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[7] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. 2007. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 6 (2007), 1067–1080.

[8] Scott Martin. 2019. *What Is Simultaneous Localization and Mapping? SLAM is a commonly used method to help robots map areas and find their way.* https://blogs.nvidia.com/blog/2019/07/25/what-is-simultaneous-localization-and-mapping-nvidia-jetson-isaac-sdk/

[9] Li Sun, Daniel Adolfsson, Martin Magnusson, Henrik Andreasson, Ingmar Posner, and Tom Duckett. 2020. Localising Faster: Efficient and precise lidar-based robot localisation in large-scale environments. *arXiv preprint arXiv:2003.01875* (2020).

[10] H Zhao, D Acharya, M Tomko, and K Khoshelham. 2020. Indoor LIDAR Relocalization Based on Deep Learning Using a 3d Model. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 43 (2020), 541–547.