Albert Ludwigs University of Freiburg

Master project

Localization with Deep Learning

Authors:
Vladislav Tananaev
Denis Tananaev
Oksana Kutkina

September 5, 2016

Supervisors:
Prof. Dr. Wolfram Burgard
Dr. Joschka Boedecker
PhD. Manuel Watter

Contents

1 Introduction

Deep learning has been extremely successful in Computer Vision, but its applications in Robotics is still limited. In this work we investigate potential uses of deep neural networks in robot localization problem. It was traditionally solved using various implementations of the Bayes Filter, most prominent of which are the Kalman Filter and the Particle Filter.

To our best knowledge there is very little activity in the area of robot localization with deep learning. The most similar to our work is Förster et al. (2007). In this work authors apply a bidirectional Long Short-Term Memory (LSTM) recurrent neural network (RNN) to solve robot localization problem. They use a simple robot simulator with an ideal map, where the robot's positions are not continuous trajectories, but rather disconnected points captured in different parts of the room. The simulated laser scan from these positions serves as the input to the neural network. We use similar approach in our work. However, we first test different approaches on toy example in case of 1D localization, where the robot's position is represented by one scalar value. We then proceed with a 2D case, where we use a robot operating system (ROS) implementation of the robot simulator to generate smooth trajectories of the robot for both synthetically generated and captured in the real-world grid maps.

We investigate two different approaches. In the first, naïve approach, we feed observations directly as input to the network and the output of the network is the absolute robot's position. This allows to re-localize robot since the network implicitly stores the map, but this is also subject to the catastrophic forgetting, when fine-tuning on the new maps leads to the loss of performance on the previous. In the second, correction approach, we use the network to output odometry corrections based on the real observation and a simulated laser scan at noisy robot's position calculated from raw odometry. This approach is able to generalize well to different environments, but cannot re-localize if the true trajectory is far from the computed one. This is very similar to the currently existing Bayesian filters which also cannot recover from filter failure.

Finally, in our work we investigate different deep learning architectures like recurrent neural network (RNN), feed-forward neural network (FFNN) and convolutional neural network (CNN).

A detailed description of the naïve and correction approaches is provided in section 2. Section 3 contains the results of our experiments with 1D localization in discrete and continuous settings. We explore both naïve and correction approaches with recurrent neural networks and feed-forward neural networks. In section 4 we evaluate performance of different architectures in a 2D case, when the data for training and test sets is generated using a custom made ROS simulator. As before, we test both the naïve and the correction approaches.

2 Outline

In this section we describe the two approaches that we evaluated in our work. The first approach takes a machine learning view on the problem when we cast the localization as a prediction problem. All of the available data at a certain time step is treated as the input of the network and ground truth is presented as the desired output. Therefore, the network learns complex mapping and is subject to the typical problems in machine learning - overfitting and catastrophic forgetting. We call this approach "naïve", since it is a straightforward way of solving a problem using deep learning. We refer to the second approach that we evaluated as "correction", since the main goal in it is to correct odometry. In this case, we use a more conventional view on the robot localization problem. We device an algorithm that is supposed to correct noise, which is similar to Kalman and Particle filters. However, it is also subject to the same limitation. When the computed trajectory diverges significantly from the true positions, it is impossible for the correction approach to re-localize.

2.1 Naïve approach

The general localization problem can be described as follows: at every time step we receive a laser sensor reading z_t, which is a noisy observation, we have x_{t-1} position of the robot computed in the previous time step and u_{t-1} the noisy odometry measurement which describes how the robot moved from $t-1$ to t. Additionally, we have m - the map of the environment. Our goal is to find the position of the robot at the current time step x_t. If we apply a motion model directly, the computed x_t will deviate from the true position due to the odometry noise. Therefore, the noisy observation should also be used in order to gain additional information and compute a better estimate.

In this approach, we feed all available information as input to the network (omitting the map), and set x_t as the desired output. Thus, the network have the opportunity to model Bayesian Filter implicitly. The summary of the inputs and output of the network is described below.

Given:

- x_{t-1} – previous position of the robot
- u_{t-1} – noisy odometry that led to x_t
- z_t – noisy observation

The goal is to find:

- x_t – current position of the robot

Additional experiment that we carried out is the direct mapping from the observations z_t to the position x_t. The rationale behind it is that in the known environment a person can recognize his precise place just by a single glimpse. Thus, the whole map is stored implicitly in the network's memory and it should query the robot's position using the received observation.

2.2 Correction approach

This approach comes from the conventional view on robot localization. In theory, in case of the ideal odometry sensor, the readings from it will allow to localize robot precisely by simple application of the motion model. Unfortunately, there are no ideal sensors, so the traditional way of coping with this is to use a Recursive Bayesian Filter (Equation 1). It allows to recursively estimate a probability density function over time from which it is possible to infer the most likely position of the robot. In this case, the uncertainty is reduced by using additional information from observations.

$$bel(x_t) = \eta p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \tag{1}$$

Similarly to this, in correction approach we use additional laser sensor information. The network takes as input the observation which is a noisy sensor measurement concatenated with a simulated observation taken from the position reported by noisy odometry. The output of the network is odometry correction Δu. By adding odometry correction to the odometry received at the current time step we should get a better estimate of the current position. In this case the motion model is modeled explicitly and computed outside of the network. The algorithmic description of the approach is presented below.

Algorithm 1 Correction approach

1: Input: x_{t-1}, u_{t-1}, z_t
2: $\hat{x}_t = motion\_model(x_{t-1}, u_{t-1})$
3: $\hat{z}_t = robot.lookup(\hat{x}_t)$
4: $\Delta u_{t-1} = net(\hat{z}_t, z_t)$
5: $\bar{u}_{t-1} = u_{t-1} + \Delta u_{t-1}$
6: $x_t = motion\_model(x_{t-1}, \bar{u}_{t-1})$
7: Return x_t

Therefore, in correction approach the sole purpose of the network is to learn the noise distribution of the odometry to be able to cancel it out using noisy observations. Conceptually it is similar to the correction step of the Kalman filter

(Algorithm 2), where the difference between true and predicted observation is weighted with Kalman gain and is used to correct the position which is computed from noisy odometry (line 7). In our case, however, the Kalman gain is implicitly learned by the network (Algorithm 1, line 5).

Algorithm 2 Kalman Filter

1: Input: μ_{t-1}, Σ_{t-1}, u_t, z_t
2: Prediction:
3: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
4: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
5: Correction:
6: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + R_t)^{-1}$
7: $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
8: $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
9: Return μ_t, Σ_t

Additional advantage of the correction approach is direct use of the motion model which can be non-linear along with the neural network that can also approximate non-linear functions.

3 One-dimensional case

We first test our ideas with a toy 1D localization in discrete and continuous settings.

3.1 Discrete case

In discrete case the robot lives in a grid world where its state is represented by a single natural number, thus $x_t \in \mathbb{N}$. To every world position there is a corresponding landmark which is also described by a natural number $z_t \in \mathbb{N}$. In our simulation, the robot can move only one cell forward, but odometry u_{t-1} may either show true move or with a certain probability say that the robot stayed in the same cell. Therefore, the neural network cannot always rely on the odometry information. Observations in our simulation z_t can be binary {0,1} or have some diversity {0, 1, ..., N}. The illustration of the discrete simulation is shown in Figure 1.

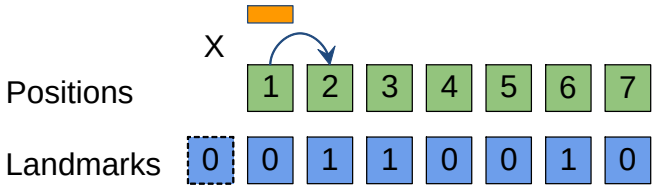

Figure 1: Discrete 1D simulation.

The landmarks outside map are all assumed to be 0. The noise in odometry is modeled probabilistically. For example, odometry is correct with probability of 0.8 and with probability 0.2 it is incorrect. The observations in this case are assumed to be perfect and the field of view (FOV) of the robot is 1 cell. Therefore, in Figure 1 from $x_1 = 2$ the robot senses $z_1 = 1$, the correct odometry that led to the current position is $u_0 = 1$.

3.1.1 Naïve approach

In the naïve approach we feed previous robot's position, odometry and observation directly to the network. In case when all of the landmarks are unique a simple feed-forward neural network would be suffice. Thus, we investigate the case when identical landmarks can be repeatedly encountered during map traversal. Thus, it is required to exploit sequential information in order to localize. At the extreme, the robot observes either 0 or 1 and should make sense of the movement based on these observations. To exploit temporal consistency that is required in this case we use a Long Short-Term Memory (LSTM) recurrent neural network (RNN) (Hochreiter and Schmidhuber (1997)) with sequence length of 4. In Figure 2 a graphical representation of the inputs and output is shown.

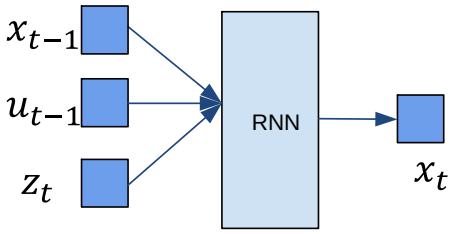

Figure 2: Inputs and the output in the discrete one-dimensional case RNN.

In fact, we use a tensor of size $batch\_size \times sequence\_length \times input\_dimension$ as input. In our case it is $N \times 4 \times 3$, where N is a batch size and can be freely chosen during training. We concatenate horizontally observation, odometry and previous position. After this, we stack this vector in time dimension vertically. In Figure 3 the recurrent neural network unrolled in time is depicted. At every time step the input tensor is updated in the First In First Out (FIFO) manner.

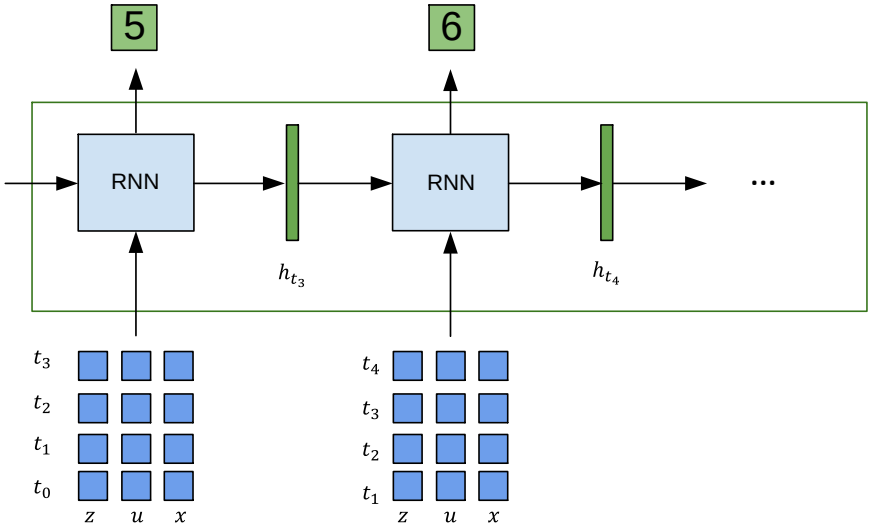

Figure 3: Discrete 1D case with RNN unrolled in time. The input is a tensor of size $batch\_size \times sequence\_length \times input\_dimension$.

We carried out an experiment with a 3 layers RNN, 64 hidden units each. During training, the objective robot's position is represented using one-hot encoding. As a result, the output layer is equal to the map size which in our case was 100. Finally, the activation used in the output layer is softmax function (Equation 2)

$$o_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}}, for j = 1, ..., K \tag{2}$$

where K is a number of neurons in the output layer and j is the output neuron number.

The results of the experiment are depicted in Figure 4 where each image corresponds to one time step. The position with the highest probability is chosen as

the final answer. The predicted initial position of the robot is incorrect, but as soon as the robot starts to move it quickly localizes itself correctly.

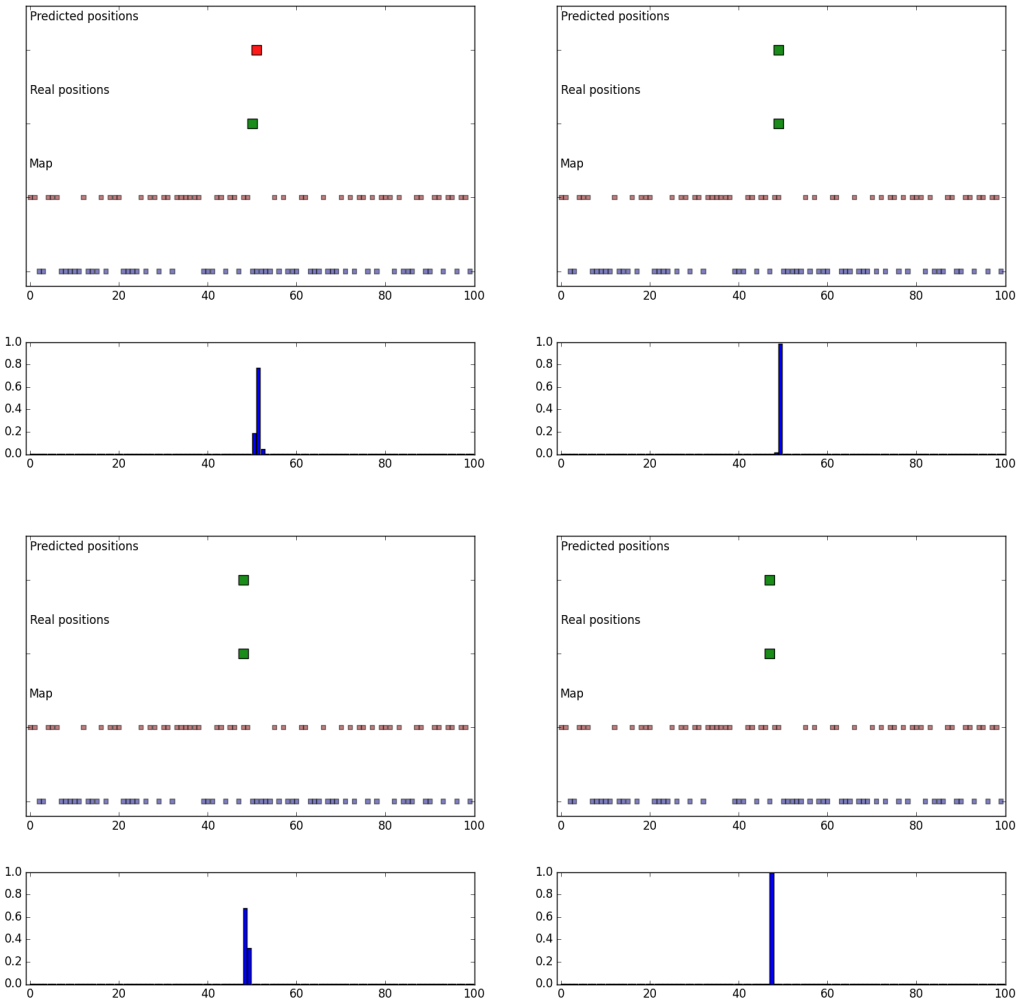

Figure 4: 1D localization experiment using RNN (layers 64, 64, 64, 100), sequence length 4 and map size 100, first four time steps is depicted. The upper row of each image shows predicted robot position, the row below is ground truth. The map is depicted at the bottom of the image. Squares laying higher in x-axis represent value 1, and lower 0. At the very bottom of each image the histogram shows the network output probability of being in every position.

However, in this approach the network should be retrained for each map. The number of hidden layers and units in each layer should always be scaled with the map size. In Figure 5 the failure sequence is shown. In this case the map of size 1000 is used while architecture of the network is the same. Since the output layer

indicates the absolute robot's position, it is possible that the network's capacity is reached.

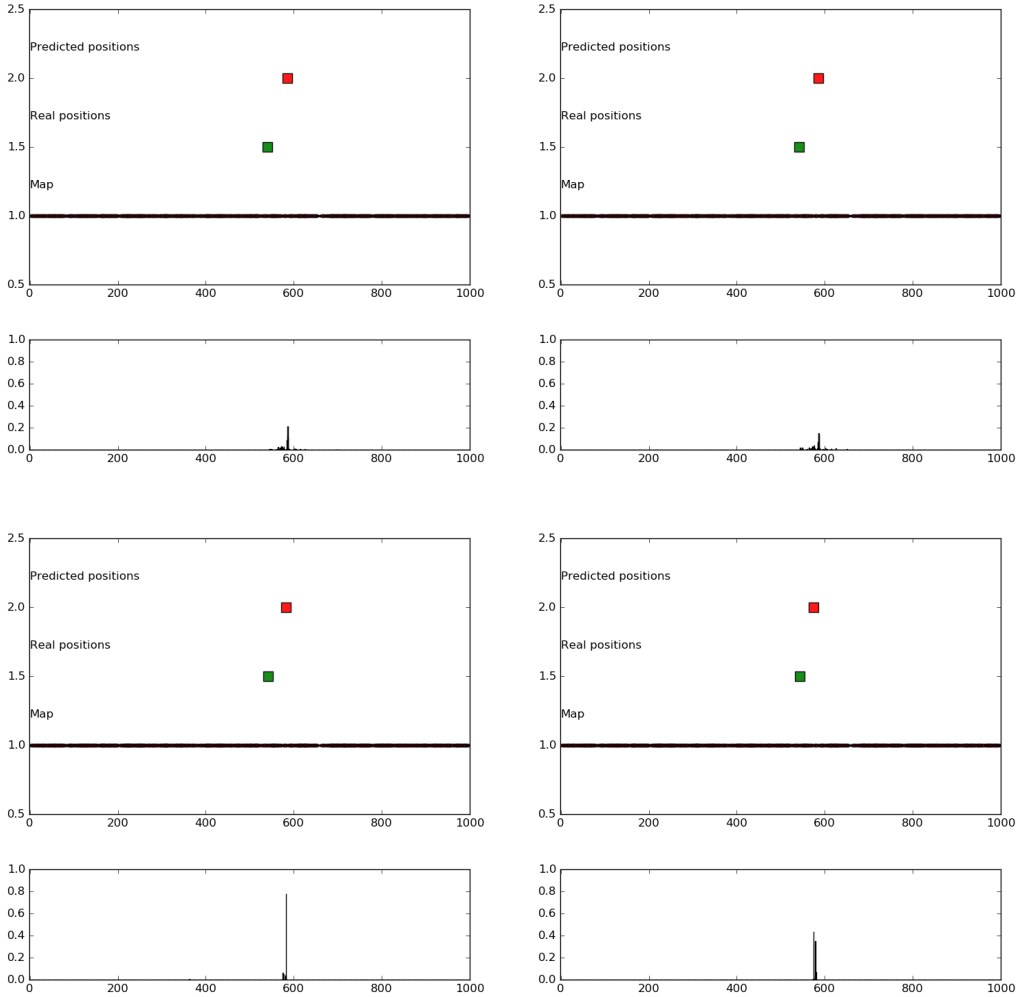

Figure 5: Failure case using bigger map, it is possible that the network's capacity is reached. The experiment is carried out with RNN (layers 64, 64, 64, 1000), sequence length 4 and map size 1000.

All in all, the naïve approach can learn to predict the robot's position, however it does not generalize to different maps. The network should be retrained or even changed. Another downside of this approach is that RNN are harder to train and due to big input tensors require more memory during training.

3.1.2 Correction approach

Main idea behind correction approach is that in order to generalize we must not use absolute positions. Instead of predicting absolute positions we can correct odometry noise. The network then needs to compare the true observation received from the sensor and simulated observation from the position from odometry. The illustration of this approach is shown in Figure 6.

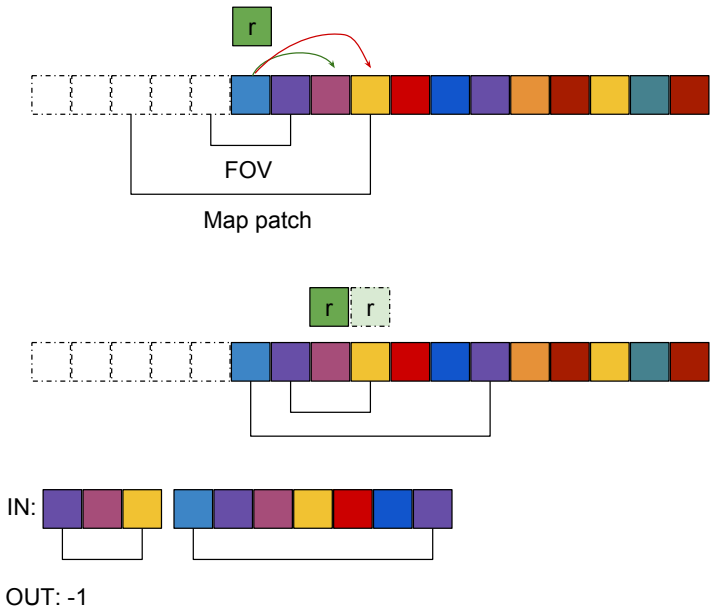

Figure 6: Correction approach for 1D case. Green square is the real robot's position, pale green is the position from odometry. Green arrow shows the real move carried out by the robot, whereas red is the move reported by odometry. Robot senses landmarks in its field of view (FOV), and can simulate observation from odometry position - the map patch. The concatenated vector of FOV and the map patch is the input to the network, and the output is the odometry correction -1.

In this illustration the robot takes a step of 2 cells forward, but the odometry reports that it moved 3 cells. The real observation that robot receives is "field of view (FOV)", and "map patch" is the simulated scan from the odometry position. The pale green square indicates the position where the robot should be according to the odometry. However, the real position of the robot is one cell before. The neural network takes as input a concatenation of the observation and the map patch. It should predict the output, which in this case should be -1. Therefore, $\bar{u}_{t-1} = u_{t-1} + \Delta u_{t-1}$ would be the corrected odometry by using which we will arrive to the true or better estimate of the robot's position. Therefore, the network does not have any notion about its environment and can generalize to different maps. At the same time, once the robot diverges from the true trajectory, the odometry

error can no longer be corrected. In Figure 7 the inputs and output of the neural network are shown.

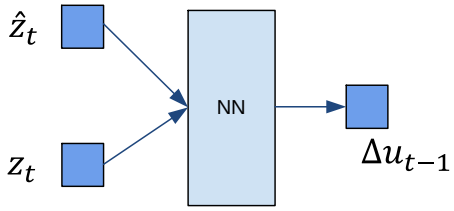

Figure 7: Inputs and the output of the network in correction approach. It takes a concatenated vector of simulated $\hat{z}_t$ and true z_t observations and outputs odometry correction Δu_{t-1}.

We carried out experiments with FFNN and LSTM RNN. We used rectified linear unit (ReLU) (Equation 3) activations after every layer in FFNN, since it performed better than classical $sigmoid(x)$ (Equation 4) or $tanh(x)$ (Equation 5).

$$f(x) = max(x, 0) \tag{3}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5}$$

Additionally, we investigated several gradient updates schemes such as *gradient decent with momentum*, *RMSProp* and *Adam*. The fastest convergence rates with lowest value of the loss function has been achieved with *Adam*. Thus, we used it in all subsequent experiments.

The achieved accuracy with feed-forward network (layers 64, 5) is 100%, and with RNN (layers 64, 64, 5) is 96%. The networks can generalize to different and bigger maps. They also work well for maps with landmark diversity which is subset of what they have been trained on. For example, if the network is trained on a map with landmark diversity $l \in \{0, ..., 5\}$ it can also perform well on maps with $l \in \{0, ..., 4\}$, but not when $l \in \{0, ..., 6\}$.

Figure 8 shows the relation between accuracy and landmark diversity for the experiments with feed-forward neural network. As soon as landmark diversity is 2 ($l \in \{0, 1\}$), the network is near perfect in correcting noisy odometry.

In Figure 9 the performance on the same problem of the LSTM RNN is shown. Since FFNN showed better accuracy with faster training times we do not use RNN in out later experiments.

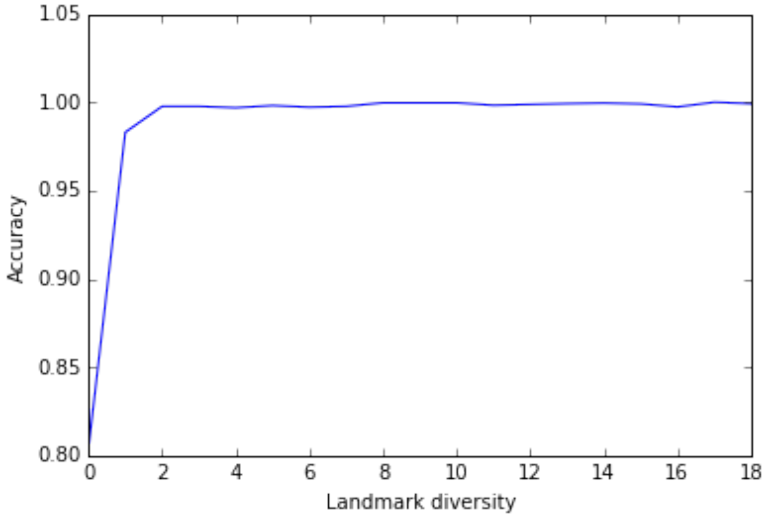

Figure 8: Relation between landmark diversity and accuracy for correction approach using a feed-forward neural network.

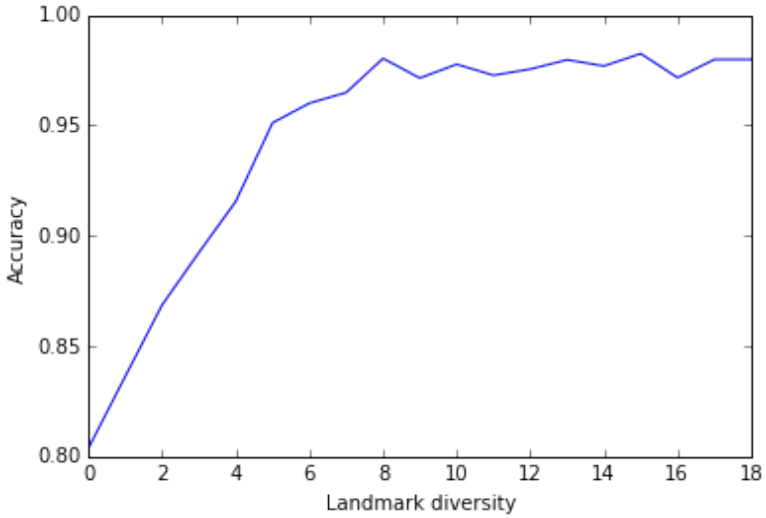

Figure 9: Relation between landmark diversity and accuracy for correction approach using a recurrent neural network. It can be seen that RNN generally performs worse than FFNN in terms of accuracy.

3.2 Continuous case

Continuous setting is very similar to the discrete. However, in this case the localization is cast not as prediction, but as a regression problem. The loss function used during training is the Mean Squared Error (MSE), shown in Equation 6

$$MSE = \frac{1}{N}\sum_{x \in X}^{N}(x_p(x) - x_g(x))^2 \tag{6}$$

where x_p is the predicted position, x_g is ground truth and N is a number of time steps.

The robot's position and odometry are represented by the real numbers $x \in \mathbb{R}$ and $u \in \mathbb{R}$, but the observation is discrete. In this experiment we assume that landmarks are like photons and the robot uses CCD camera which measures the quantity of photons in a certain interval. Essentially, the sensor is modeled by the histogram (Figure 10).

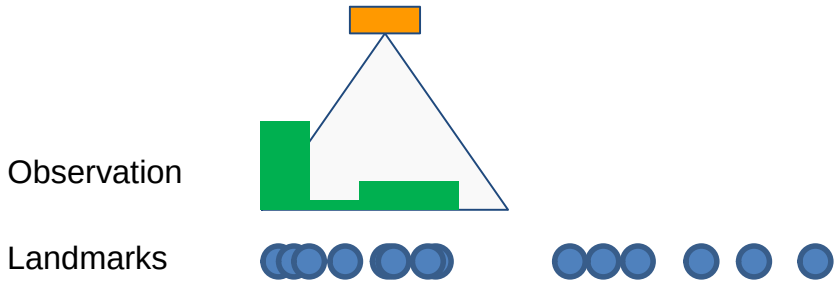

Figure 10: Continuous one-dimensional case.

In continuous case we carried out experiment only with feed-forward neural network (layers 128, 64, 1).

A few time steps of the experiment are shown in Figure 11. The corrected position of the robot stays near the true position, while trajectory computed by using odometry diverges. Here the map is represented by the histogram to depict how the environment is sensed by the simulated robot.

The inputs and outputs in the continuous settings are similar to the ones we use later in 2D case. The input of the network is the vector of the discrete observations (laser scans in 2D) and the architecture regresses odometry correction. We continue further with the exploration of the naïve and correction approaches in 2D case.

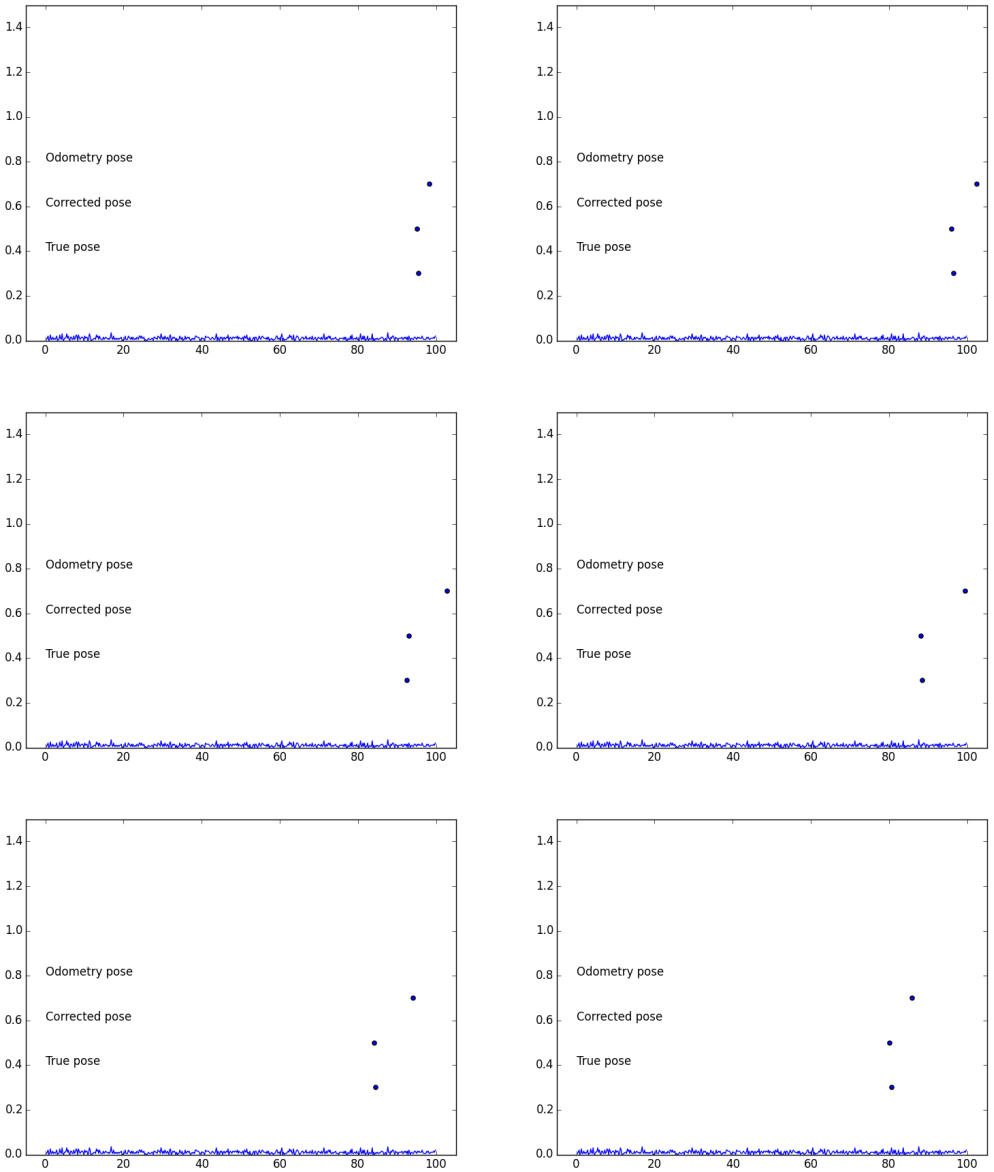

Figure 11: Correction approach with Feed-Forward Neural Network. The corrected position of the network stays close to ground truth while the position computed via odometry accumulates error and diverges.

4 Two-dimensional case

In this section we explore the possibility of application of the deep neural networks to the robot localization problem in 2D. The robot's pose is described by three parameters (x, y, θ), where x and y are the coordinates of the robot in 2D plane and θ is the orientation angle. The occupancy grid map and the initial position of the robot are given. Additionally, we have odometry information from the wheel encoders which can be represented as linear v and angular ω velocities of the robot and 2D laser scans of the environment with 181 (for the view of 180 degrees) or 362 (for the view of 360 degrees) distance measurements. Our goal is to estimate the robot's pose in each moment using observations (2D laser scans), odometry information and the map. To achieve this goal, we tried feed-forward and convolutional neural networks with different architectures, gradient updates and training schemes. We did not make experiments with recurrent neural networks, because they showed poor performance compared to the FFNN in 1D case.

This section describes full pipeline of the deep learning localization starting from the simulation of the robot and its environment, datasets generation and concluding with working results of different neural networks on the maps of variable sizes. We also provide the full description of advantages and disadvantages for each approach and discuss possible future improvements.

4.1 Simulation

As the environment for the robot simulation we use Robot Operating System (ROS). For the map and motion simulation we use the packages of Navigation Stack Marder-Eppstein (2016) (map_server for map and move_base for navigation). As the maps for the training and test set generation we use the current benchmark maps for robot localization (University of Bonn (2016)). For all experiments we use the map resolution 0.01 (10 cm) with threshold for occupied and free cells 0.55 and 0.196 respectively. We simulate the odometry noise using the following odometry model (Thrun (2002)):

$$\delta_{rot1} = atan2(\overline{y}' - \overline{y}, \overline{x}' - \overline{x}) - \overline{\theta} \tag{7}$$

$$\delta_{trans} = \sqrt{(\overline{x} - \overline{x}')^2 + (\overline{y} - \overline{y}')^2} \tag{8}$$

$$\delta_{rot2} = \overline{\theta}' - \overline{\theta} - \delta_{rot1} \tag{9}$$

Here $\overline{x}$, $\overline{y}$, $\overline{\theta}$ and $\overline{x}'$, $\overline{y}'$, $\overline{\theta}'$ are the coordinates before and after robot motion and δ_{rot1}, δ_{trans}, δ_{rot2} are the relative translation and rotations between two poses (see Figure 12).

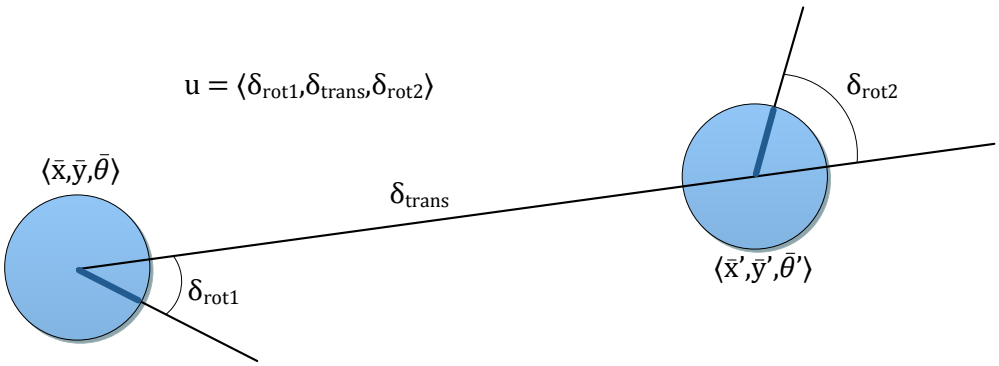

Figure 12: The odometry model. Here $\overline{x}$, $\overline{y}$, $\overline{\theta}$ and $\overline{x}'$, $\overline{y}'$, $\overline{\theta}'$ are the coordinates before and after robot motion and δ_{rot1}, δ_{trans}, δ_{rot2} are the relative translation and rotations between two poses.

Given relative transformation between two poses we add zero-mean Gaussian noise

$$\hat{\delta}_{rot1} = \delta_{rot1} + Gaussian(\alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2) \tag{10}$$

$$\hat{\delta}_{trans} = \delta_{trans} + Gaussian(\alpha_3\delta_{trans}^2 + \alpha_4\delta_{rot1}^2 + \alpha_4\delta_{rot2}^2) \tag{11}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} + Gaussian(\alpha_1\delta_{rot2}^2 + \alpha_2\delta_{trans}^2) \tag{12}$$

where α_1, α_2, α_3, α_4 are the parameters which we set 0.01, 0.01, 0.005 and 0.005 respectively.

And finally, we calculate the noisy position from odometry.

$$x' = x + \hat{\delta}_{trans}cos(\theta + \hat{\delta}_{rot1}) \tag{13}$$

$$y' = y + \hat{\delta}_{trans}sin(\theta + \hat{\delta}_{rot1}) \tag{14}$$

$$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \tag{15}$$

In order to simulate laser scans we use fast ray cast algorithm provided by Amanatides et al. (1987). The general idea is to split the space into the regions. Then we shoot the ray and check the regions for intersections between the ray and obstacles starting from the closest to the ray's origin region. When the closest intersection is found, this point becomes the end point of the ray. Let's consider how ray cast algorithm works in more details (see Figure 13).

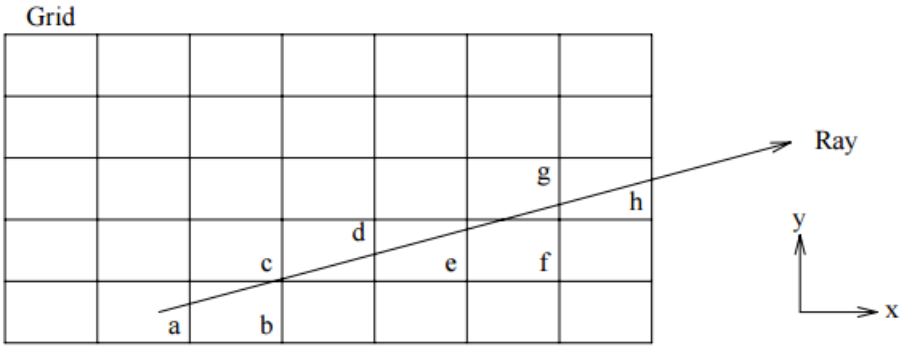

Figure 13: The example of the fast ray cast algorithm.

In order to correctly traverse the grid the algorithm should go through the regions a, b, c, d, e, f, g, h exactly in that order. The algorithm splits the ray into time t intervals. Each time interval is related to it's own region. We start at the ray's beginning and go through each region in the order of time intervals.

It is possible to split the traversal algorithm in two steps: initialization and incremental traversal. In initialization step the region of the ray origin should be defined. In case when the ray origin lays outside the map, we find the point where the ray enters the map and take the region related to this point. The starting point of the ray: x, y coordinates are initialized. Additionally, the variables step by x and y axis are set to the 1 and -1. The step variables indicate whether x and y incremented or decremented as the ray crosses the region boundaries. Then we find the value t at which the ray crosses the first vertical region boundary and store that variable as $tMaxX$. Then the same computations performed for y and we store the variable as $tMaxY$. Finally, we compute $tDeltaX$ and $tDeltaY$ which indicate how far along the ray we must move for the horizontal and vertical directions. The operation is repeated until we find the closest point of the intersection between the ray and an obstacle.

In our simulation we generated noisy observations with sensor noise variance 0.0001.

We model the robot with size 2×1 m. The position of the laser sensors is set relative to the robot's reference frame and for the front sensor the offset is 1.2 m along x-axis from the center of the rotation of the robot, and 0.4 m along both x and y-axis for the rear sensor. We put the laser sensors in different positions like it could be on the real robot in order to make simulation closer to reality. We also introduce new system of tf transforms for the ROS for training set generation (see Figure 14).

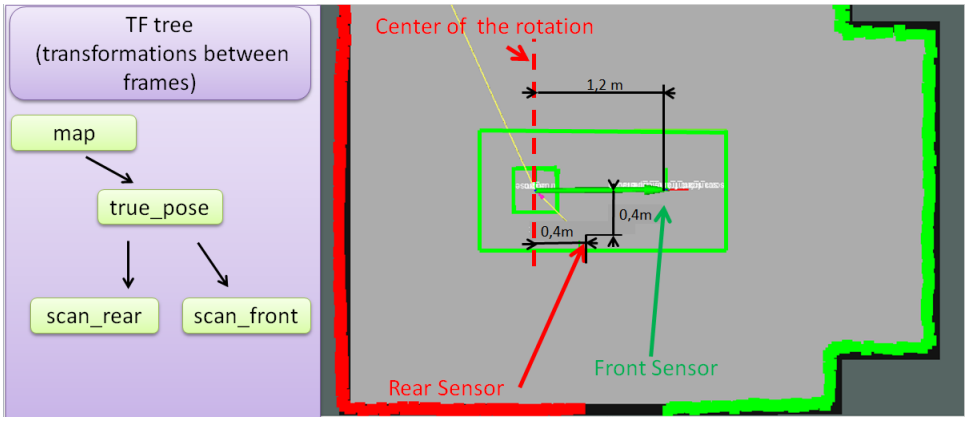

Figure 14: On the left - the tf tree transformation for datasets recording; on the right - the size of the robot and positions of the laser sensors visualized in Rviz.

The tf tree consists of the transformations between frame *map* and *true_pose* which gives the real position of the robot in the map and transformations between *true_pose* and laser sensors *scan_front* and *scan_rear*. Given the positions and observations from the laser scans we generate noisy odometry for each data sample using odometry model, which was described before.

4.2 Datasets generation

We generate datasets for training and test. The datasets includes the perfect positions of the robot x, y, θ as the labels, then 362 range measurements from the laser scan sensors (181 for front and same for the rear sensor), δ_{rot1}, δ_{trans}, δ_{rot2} - the perfect odometry between the positions and $\hat{\delta}_{rot1}$, $\hat{\delta}_{trans}$, $\hat{\delta}_{rot2}$ - the noisy odometry.

We use seven different maps with different sizes and complexity for datasets generation. During the recording we get the datasets samples with the frequency of 10 Hz.

The first dataset was recorded on the map with the size 21×27.7 m. The robot on this map was allowed to move only on the ellipse-like trajectory. We generate random point on the trajectory as the goal and wait until the robot achieves it and then sample the next goal. Robot is allowed to move both forwards and backwards (see Figure 15).

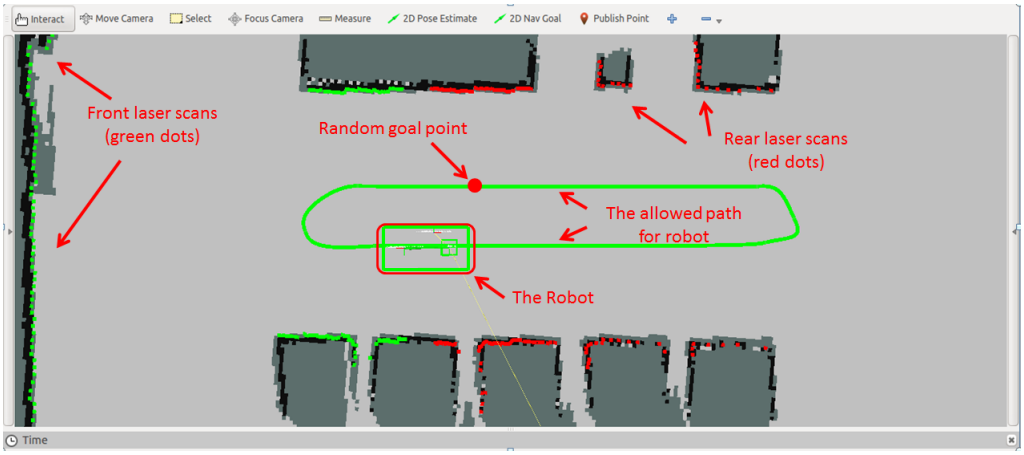

Figure 15: The first dataset generation. The green line is the trajectory where the robot is allowed to move; red dot is a randomly sampled goal position. After robot reaches it, the new goal is sampled; green and red dots are the laser scans from front and rear sensors respectively.

For motion implementation we used DWA motion planner from move_base package.

The total time of the recording is six hours (or approx. 200 000 data samples). Additionally, we changed the motion trajectory from ellipse to the straight line and recorded one hour of the data for the test set.

The second dataset was recorded on the completely symmetric map. The size of the map is 16×60 m (see Figure 16).

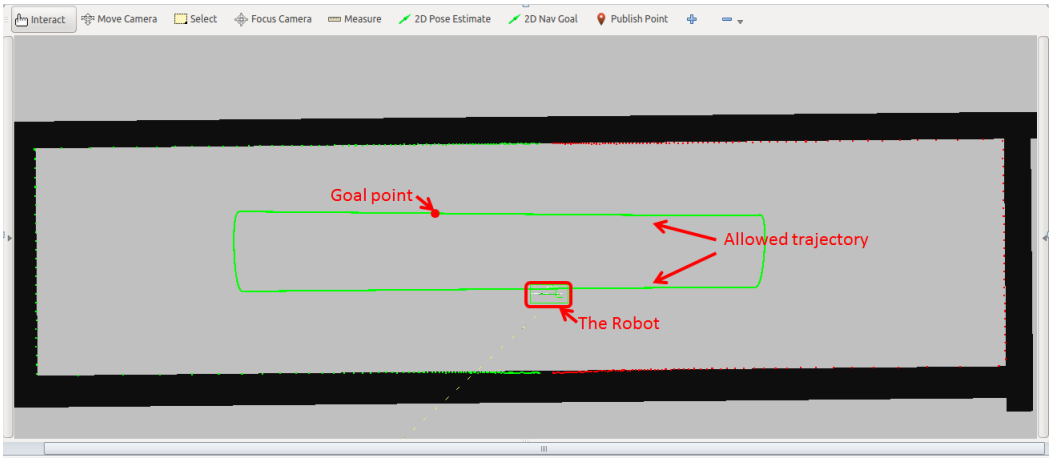

Figure 16: The second dataset generation. The green line is the trajectory where the robot is allowed to move; red dot is a randomly sampled goal position. After robot reaches it, the new goal is sampled.

The total time of the recording is six hours. The third dataset was generated at the map of the size 50×38 m (see Figure 17).

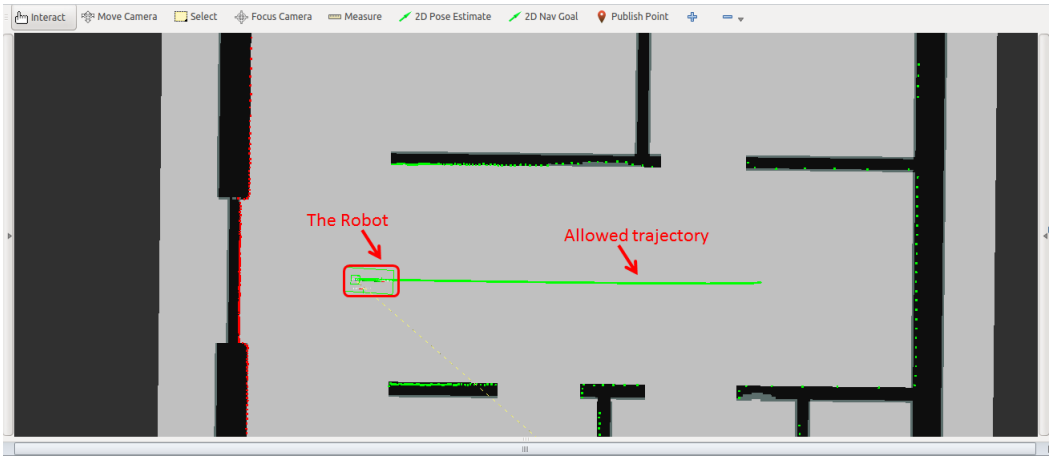

Figure 17: The third dataset generation. The green line is the trajectory where the robot is allowed to move.

The total time of recording is one hour (approx. 4000 data samples).

The next datasets were recorded without restriction of the robot's trajectory. We randomly sample point in free space of the map. Then the robot moves to the point using trajectory planner and DWA controller from Navigation Stack with a purpose of reaching the goal. The global and local cost maps from Navigation stack were used in order to avoid collisions (see Figure 18).

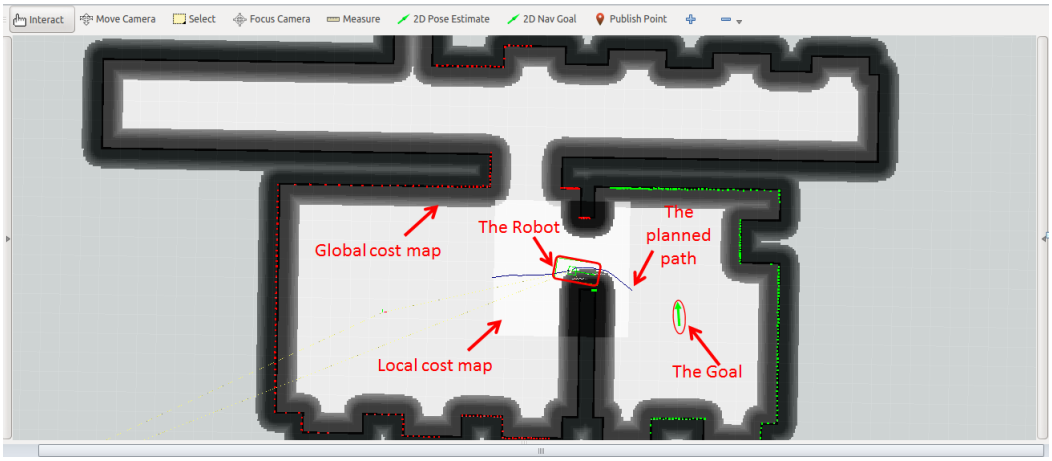

Figure 18: The fourth dataset generation.

The size of the map is 41×52 m. The total time of recording is 8 hours (approx. 260 000 data samples).

The last three datasets were recorded the same way as the forth dataset.

For the dataset five we used the map recorded by Cyrill Stachniss in building 079 at the University of Freiburg (see Figure 19). The total time of the recording is four hours (approx. 130 000 samples).

Figure 19: The fifth dataset map: the building 079 of the University of Freiburg.

For the sixth and seventh datasets we used the map recorded by Dirk Hähnel at the Intel Research Lab in Seattle and the map of the offices (see Figure 20). The total time of recording is 4 hours for each map (approx. 260 000 samples).

Figure 20: Left - the sixth dataset map: the Intel Research Lab in Seattle; right - the seventh dataset map: offices.

In total, we recorded 1 054 000 data samples (total time is 35 hours) on seven different maps with different levels of difficulties with and without restriction for robot motion.

4.3 Scan matching network

The first series of tests we made using the naïve approach. The idea of the naïve approach is to simply create neural network which implicitly memorizes the map. In order to do that we use a simple feed-forward neural network with three hidden layers, 362 neurons each (see Figure 21).

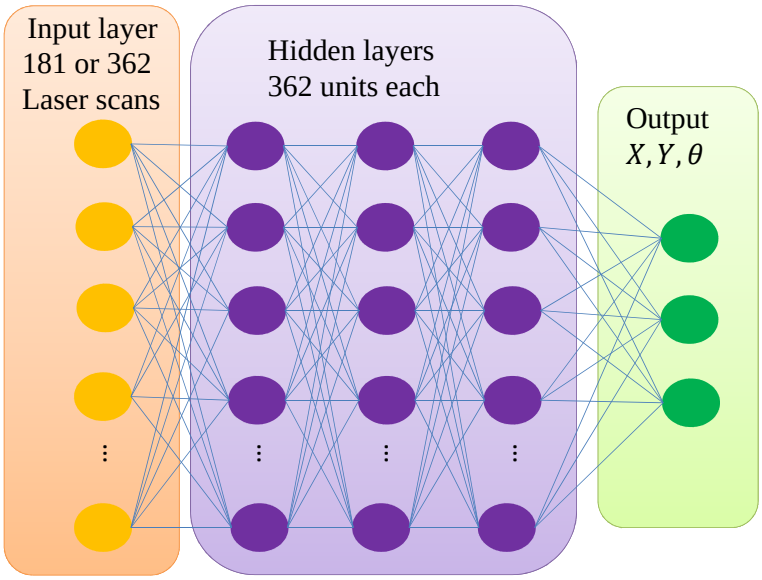

Figure 21: The feed-forward neural network for naïve localization approach trained on the first dataset.

In our experiments we train the neural network with input of the size 181 (only the data from the front laser sensor) and 362 noisy distance measurements from the front and rear laser sensors. As output we receive the x, y, θ corresponding to the global position on the map. We found out that varying the size of the input vector does not improve the performance of the neural network on the small map. We used the first dataset for training and test (90% of the data is used for training and 10% for tests).

The goal of the first experiment is to overfit the neural network in order to check how good it could memorize the positions and map. For this reason we did not use the dropout layers.

We trained the neural network using $ReLU(x)$ as the activation functions. We also explored $tanh(x)$ and $sigmoid(x)$ functions, but the loss did not drop in these cases. As the loss function we used MSE.

On the earlier stage of the experiments we made a comparison between different learning algorithms like *gradient descent with momentum*, $RMSProp$, *Adam* and the last one showed the best results. Thus, in all our future experiments we used *Adam* with weight decay.

We also checked the performance of the neural network for localization of the robot by calculating MSE. As the result we received the perfect localization of the robot on the test set which was similar to the train set with MSE 0.000125 (see Figure 22).

Figure 22: The first experiment on the first dataset. Green arrow is ground truth; red arrow - prediction from neural network; MSE = 0.000125.

As the next experiment we tried the same network on the test set which was recorded on the same map, but the trajectory for robot motion was the straight line (for the train set the trajectory of motion was ellipse, see Figure 15). As the result, the MSE for this experiment becomes 0.4567. This is because the neural network not only memorized the map, but also the robot's trajectory from the training set. Because of this, even given the different laser scans, the network tried to localize the robot on an ellipse instead of a line (see Figure 23).

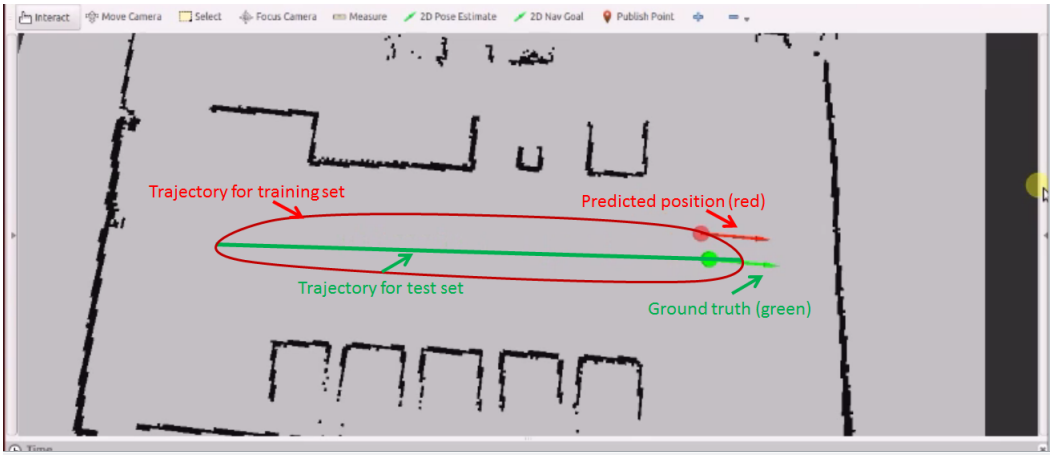

Figure 23: The second experiment with different trajectory for the test set. Green arrow is ground truth; red arrow - prediction from neural network; MSE = 0.4567.

In order to check how good this approach generalizes on different maps, we tested our neural network on a new map (see Figure 24).

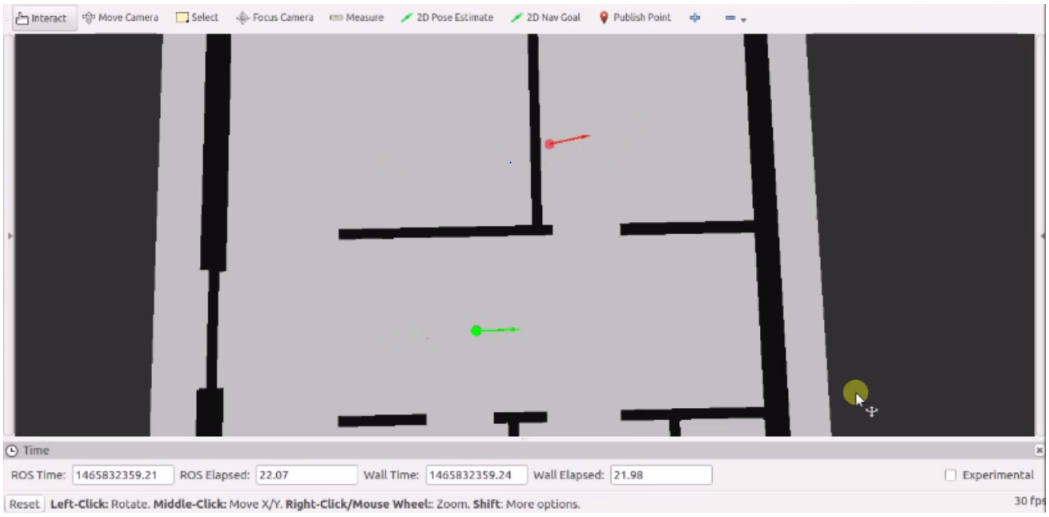

Figure 24: The third experiment with different map for the test set. Green arrow is ground truth; red arrow - prediction from neural network; MSE = 5.32.

It is possible to see that the drawback of using the global positions for the training of the neural network is that it will be able to only localize the robot on the exactly the same map where it was trained. Moreover, if in ROS the map was located at (0,0) coordinates and then it was shifted in some other place in the same system of coordinates the network will output wrong localization (see Figure 25).

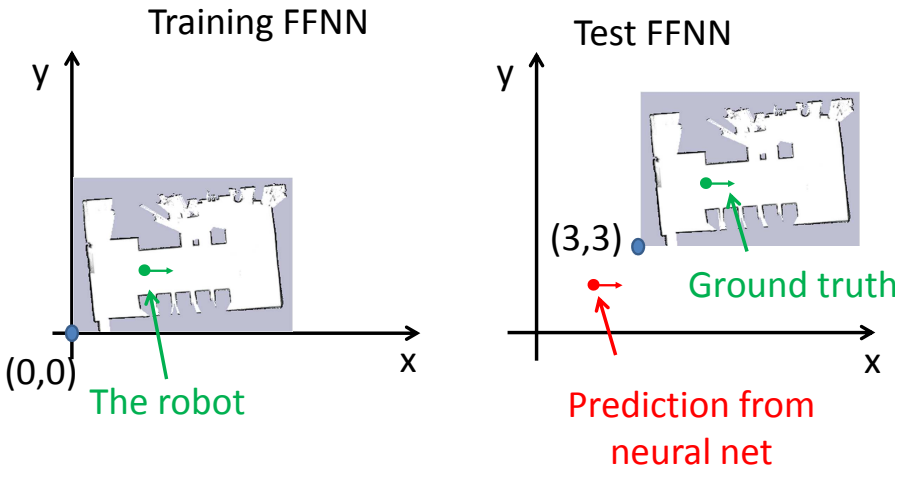

Figure 25: The example of wrong localization, because of the training FFNN on the global positions of the robot.

In order to check robustness of this neural network to the symmetric environ-

ment we trained and tested it on the second dataset. The neural network fails to localize the robot in symmetric environment because of ambiguity of the observations (see Figure 26).

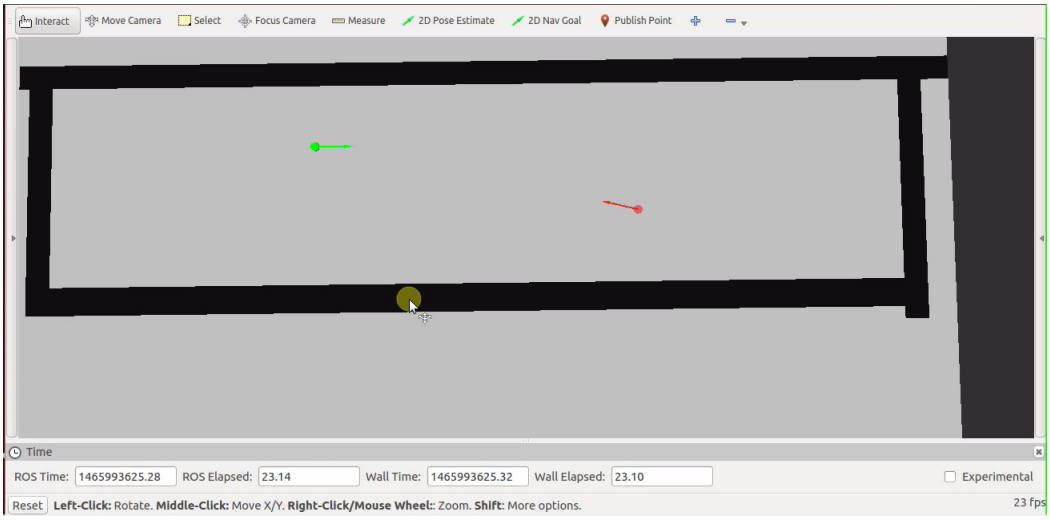

Figure 26: The example of the wrong localization with the neural network in the symmetric environment. Green arrow is ground truth; red arrow - prediction from neural network.

The next set of the experiments we made in order to generalize the positions on the map. For this purpose we used the dataset four. The size of the map is bigger than in our previous experiments. For this reason, we increased the size of the neural network and we also introduced dropout layer in order to avoid overfitting (see Figure 27).

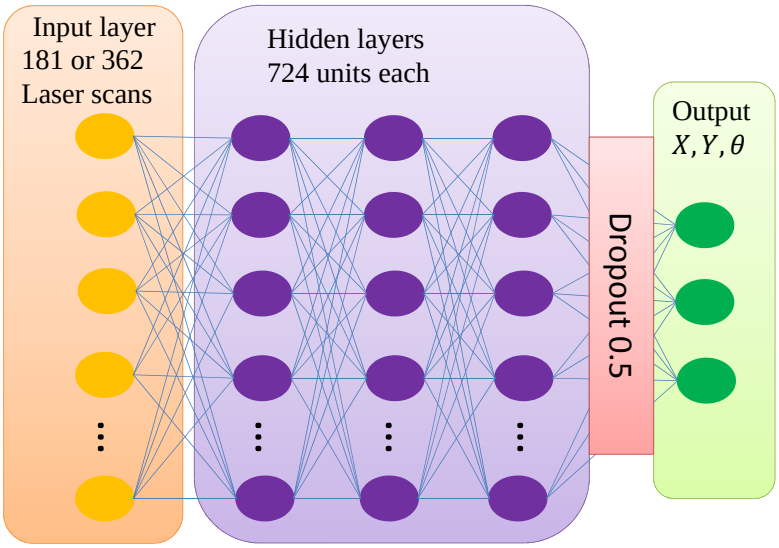

Figure 27: The architecture of the feed-forward neural network for training on the fourth dataset.

Like with the previous experiment, varying the size of the input vector of observations did not give valuable increase of performance. The training procedure was implemented the same way as for the first network. We used *Adam* as learning algorithm, *ReLU* activation functions and MSE loss.

The resulting MSE on the test set is equal to 0.25 which is worse than the results on the previous datasets. But in this case robot is able to move through the whole map with random trajectories (see Figure 28).

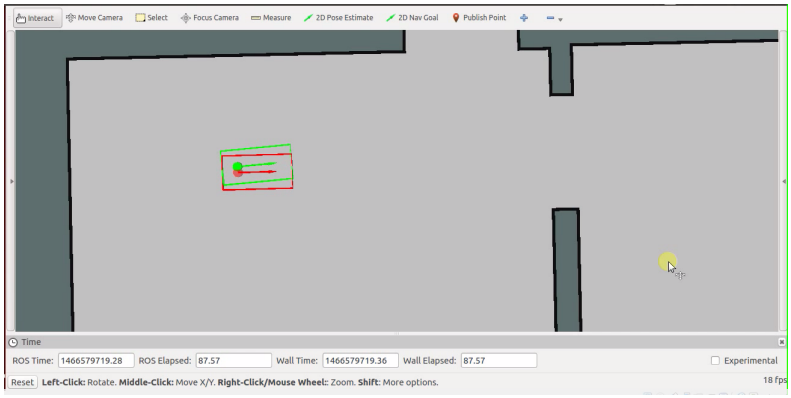

Figure 28: The result of the fifth experiment. The robot can move everywhere in the map using random trajectories. Green arrow and contour - ground truth; red arrow and contour - prediction from the neural network; MSE=0.25.

In order to improve the received result we add as input to the network the previously predicted position (see Figure 29).

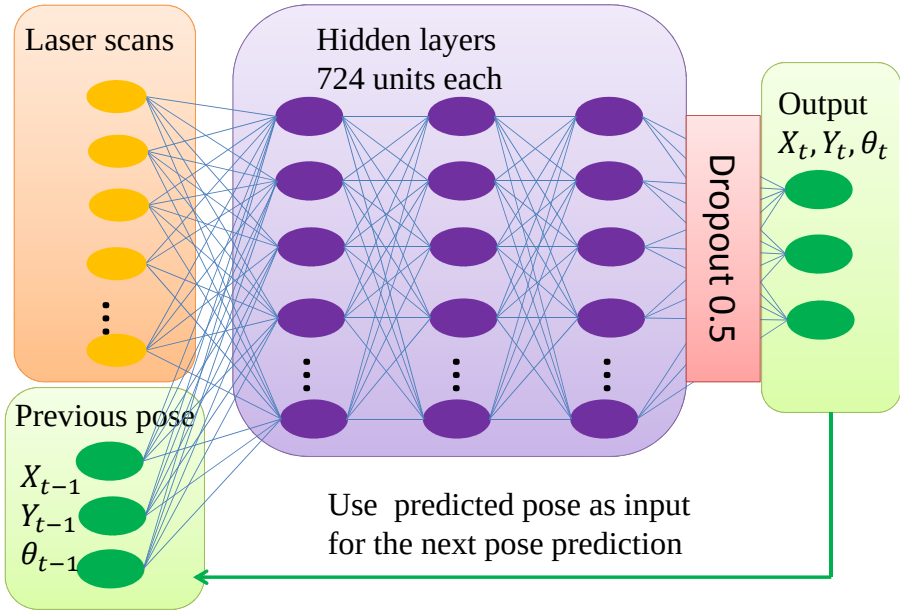

Figure 29: Use predicted position from previous state as input for computing next state.

Surprisingly, the previously predicted pose did not improved the localization, but instead made it even worse. The MSE becomes 2.95. The reason why this happens is that the neural network makes the prediction with some small error, and this error accumulates over time. We also tried to add as input raw odometry, but this experiment showed worse results compared to the laser-scan-only localization.

The naïve approach showed good results on the small maps without symmetry in the environment, but it fails in symmetric environments. Also, it is necessary to train a particular neural network for a particular map in order to localize the robot. But the interesting findings is that the neural network doesn't need a map information (unlike traditional methods like Kalman Filter and Particle filter localization) and after the training it can use only laser scans for localization, because the map is saved in the weights of the neural network.

4.4 Map regeneration

In order to check how well neural network can memorize the map, we do the opposite of what we did in the last section. We increase the number of layers of the neural network and swap the input and output (see Figure 30).

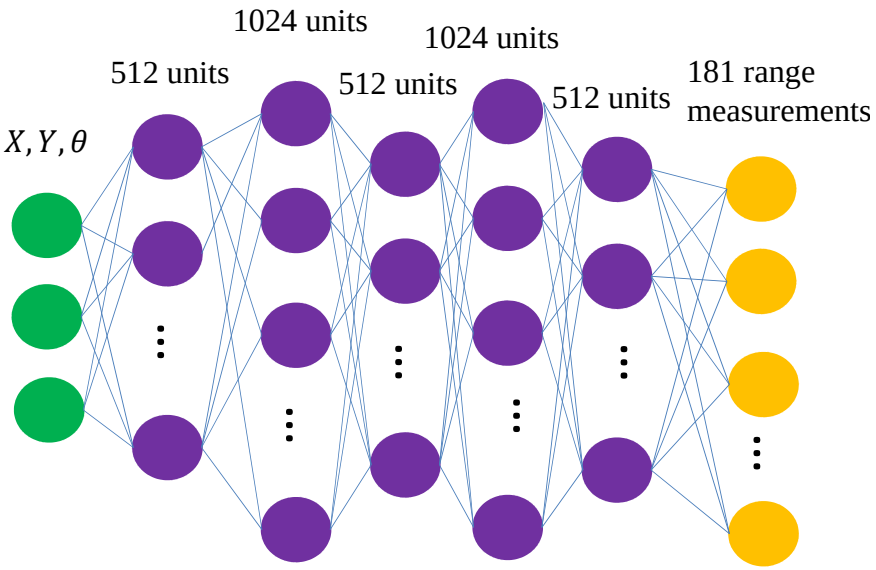

Figure 30: The feed-forward neural network architecture for map regeneration.

With feed-forward neural network we were able to achieve MSE=0.4 (see Figure 31).

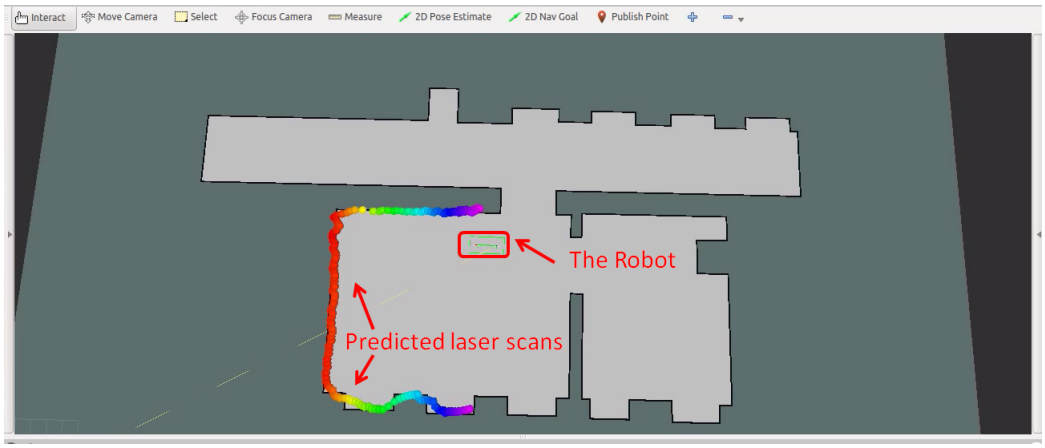

Figure 31: The example of laser scans generation with neural network given x, y, θ position of the robot.

We found out that even without tuning neural network it has capabilities to keep the whole map in the weights. The size of the map should be proportional to the size of the neural network. Some iterations over the architecture of the neural network should be preformed in order to achieve better MSE.

4.5 Correction network

In order to generalize to new maps we evaluate the correction approach, where the network is used to correct odometry noise. In this case, the network takes an observation vector z_t (size 181) concatenated with the simulated laser scan $\hat{z}_t$ (size 181) which results in one input vector of size 362. The output that network should learn is the correction for odometry u_{t-1} which is a tuple of three values $(\Delta\delta_{rot1}, \Delta\delta_{trans}, \Delta\delta_{rot2})$.

Ideally, we would like to receive $(\Delta\delta_{rot1}, \Delta\delta_{trans}, \Delta\delta_{rot2})$ such that

$$\delta_{rot1} = \hat{\delta}_{rot1} + \Delta\delta_{rot1} \tag{16}$$

$$\delta_{trans} = \hat{\delta}_{trans} + \Delta\delta_{trans} \tag{17}$$

$$\delta_{rot2} = \hat{\delta}_{rot2} + \Delta\delta_{rot2} \tag{18}$$

where $(\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ is the ideal odometry and $(\hat{\delta}_{rot1}, \hat{\delta}_{trans}, \hat{\delta}_{rot2})$ is the noisy odometry. Thus, by adding corresponding odometry corrections predicted by the neural network we should be able to get the ideal odometry. From the equations 10, 11 and 12 it follows that the network objective is to learn the inverse of the additive noise. Therefore, the network objective is $(\Delta\delta_{rot1}, \Delta\delta_{trans}, \Delta\delta_{rot2})$, where

$$\Delta\delta_{rot1} = -Gaussian(\alpha_1\delta^2_{rot1} + \alpha_2\delta^2_{trans}) \tag{19}$$

$$\Delta\delta_{trans} = -Gaussian(\alpha_3\delta^2_{trans} + \alpha_4\delta^2_{rot1} + \alpha_4\delta^2_{rot2}) \tag{20}$$

$$\Delta\delta_{rot2} = -Gaussian(\alpha_1\delta^2_{rot2} + \alpha_2\delta^2_{trans}) \tag{21}$$

The training set is generated from pairs of true consecutive positions which are treated independently $(\{x_0, x_1\}, \{x_1, x_2\}, ...)$. To the ideal odometry, which can be computed between every pair $(\bar{u}_{01})$, the noise is added. This noisy odometry (u_{01}) is used to produce a new robot's position which is slightly different from the true pose $(\hat{x}_1 = motion\_model(x_0, u_{01}))$. From $\hat{x}_1$ position the simulated scan $\hat{z}_1$ is generated.

Therefore, every time step is an independent training sample, where z_t is a noisy laser scan received from the true position x_t and $\hat{z}_t$ is a simulated laser scan which is generated from the noisy position $\hat{x}_t$.

We carried out experiments on the fourth dataset which was generated using DWA controller from Navigation Stack, the fifth dataset from map of building 079 of the University of Freiburg and the seventh dataset "offices".

In all experiments with FFNN the loss function did not converge and the localization performance was poor. Thus, we devised a convolutional neural network similar to VGG (Simonyan and Zisserman (2014)).

Our CNN consists of 5 convolutional layers and 3 fully connected layers. We used padding 0, kernel sizes for convolutional layers 3×1 with stride 1 and for

max pooling 2×1 with stride 2. In Figure 32 a convolutional neural network used in our experiments is depicted. We refer to it as the "Correction Net".

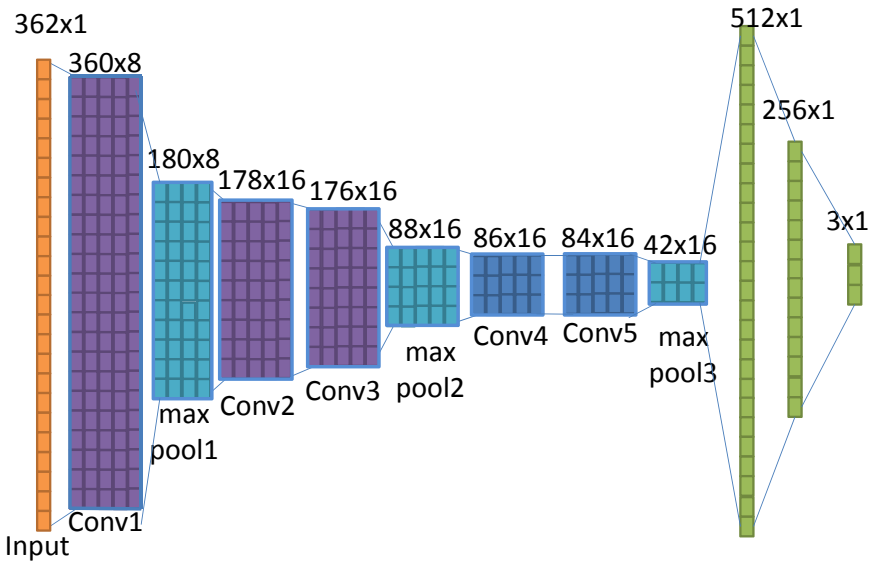

Figure 32: The Correction Net.

Architecture of the Correction Net (Width x Height x Depth):
[362x1x1] **INPUT**
[360x1x8] **CONV1**: 8 3x1 filters at stride 1, pad 0
[180x1x8] **MAX POOL1**: 2x1 filters at stride 2
[178x1x16] **CONV2**: 16 3x1 filters at stride 1, pad 0
[176x1x16] **CONV3**: 16 3x1 filters at stride 1, pad 0
[88x1x16] **MAX POOL2**: 2x1 filters at stride 2
[86x1x16] **CONV4**: 16 3x1 filters at stride 1, pad 0
[84x1x16] **CONV5**: 16 3x1 filters at stride 1, pad 0
[42x1x16] **MAX POOL3**: 2x1 filters at stride 2
[512] **FC6**: 512 neurons
[256] **FC7**: 256 neurons
[3] **FC8**: 3 neurons

After each convolutional layer and in the fully connected layers *ReLU* is used as activation function.

In the first experiment we use the ideal observations and test the network on the training set. While FFNN has failed even in this experiment, the Correction Net achieved MSE of 0.0003 (see Figure 33).

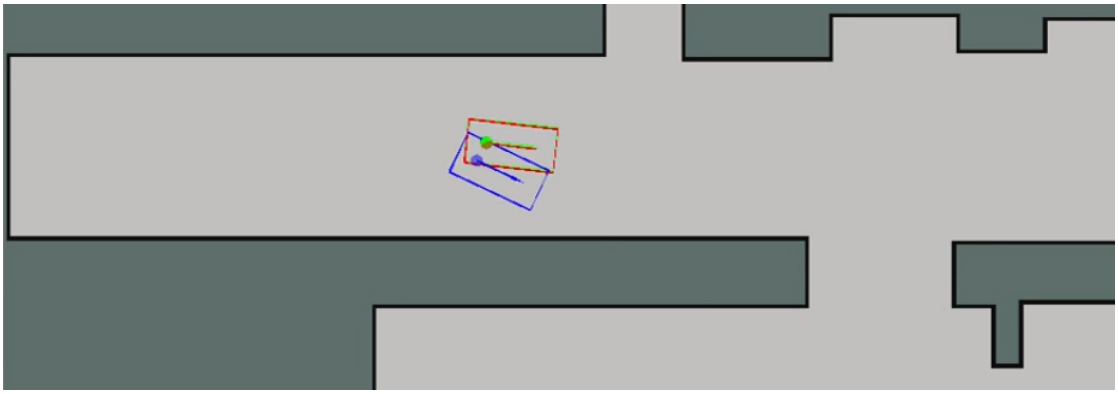

Figure 33: Performance of the Correction Net on the training set. Achieved MSE is 0.0003. In blue is the position of the robot computed from odometry, green - ground truth, red - position corrected by the Correction Net.

In the next experiment we tested the network on the same trajectory, but with the new noisy odometry which yields MSE of 0.00089.

The network was also able to cope with different trajectories on the same map, however it failed on a new, more complex map of building 79 of the University of Freiburg (Figure 34).

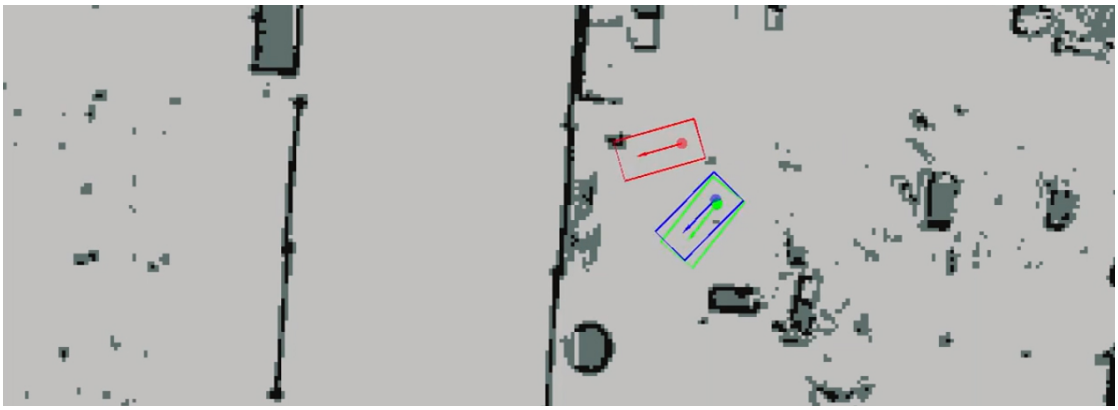

Figure 34: Performance of the Correction Net on the map of building 79 of the University of Freiburg. The network fails to localize since observations on the map are more challenging than in training set.

However, when the network was trained on a challenging map and used after in a simpler, unseen map, it was able to localize much better. In this case the network was able to tolerate some noise in observations (zero-mean additive Gaussian noise with $\sigma = 0.001$). The result is shown in Figure 35.

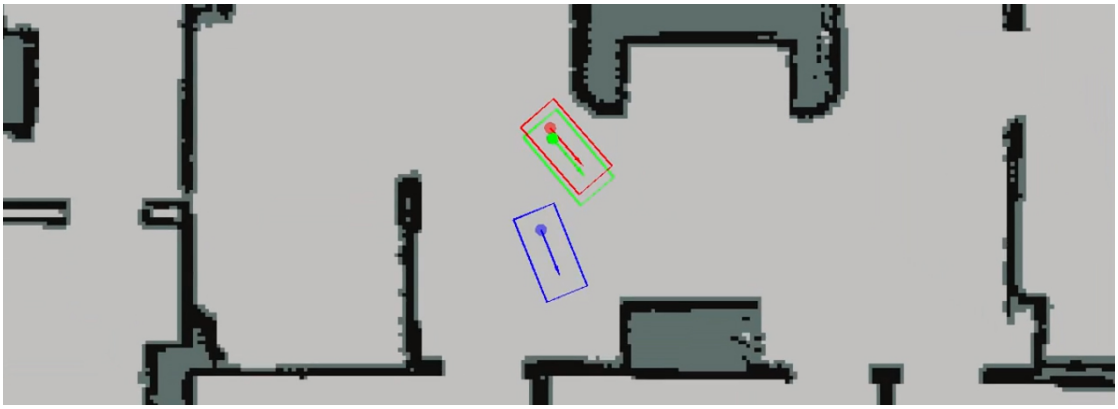

Figure 35: Performance of the Correction Net on a new map with noisy observations. The robot's corrected position is much better than the raw odometry, however there is an error which leads to divergence later.

However, in some cases the network diverges, see Figure 36.

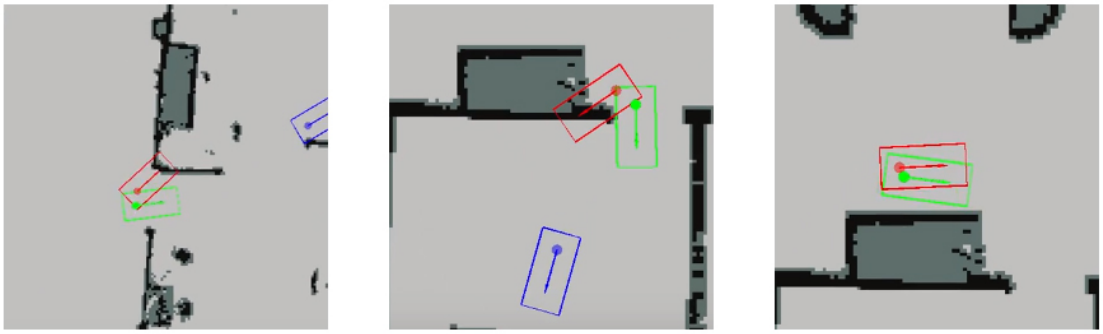

Figure 36: Correction Net failure cases. In the first two cases the possible reason for failure is that simulated scan "sees through" the wall. In the last case the possible reason is ambiguity in the observation.

To sum up, the Correction Net is able to localize on new maps with noisy observation which is similar to the real-world case. However, it fails in some cases which lead to divergence without chance of recovery. There are three possible reasons for those failures.

The first one is that the network did not see enough training examples to be robust. In general, more training time and diverse maps seems to be helpful to improve localization.

The second possible reason is that the network was trained only on small errors without outliers. Therefore, in practice error accumulates, which sometimes results in the situation when the position computed from the corrected odometry "sees through walls". In this case, the difference between real and generated observations is significant and the network output is usually a large and incorrect odometry correction which makes the robot swerve into completely wrong direction after

which the recovery is impossible. One possible remedy to that problem could be to train network to treat such observations as outliers and ignore them.

The third possible reason of the divergence is the commitment to one hypothesis. In ambiguous situation there could be several plausible corrections. However, the network always makes hard decisions at each time step. An interesting direction to investigate in order to solve this problem is to add some memory to the network. One simple way is just stack several observation vector in 2D matrix and another is to use additional RNN which will keep track of the decisions.

5 Conclusions

In our work we investigated the capabilities of the deep neural networks for the robot localization.

We explored several deep neural network architectures using developed 1D and 2D simulations for the robot localization problem. We evaluated performance in both discrete and continuous settings. In our experiments we used RNN, CNN and feed-forward neural network and explored two approaches for solving the problem: naïve and odometry correction.

The idea of the naïve approach is to give the observations and any other information that we have like odometry in order to output the global position. We showed that the naïve approach works very well on the map of small size and fails when the size of map grows and the map has a lot of symmetric elements. It is also impossible to make a general localization mechanism based on this approach, because for each new map we should train our neural network from the beginning. But the interesting findings is that the neural network is capable to save the map in its weights and use only observations to localize the robot (which is not true for traditional approaches like Kalman filter and Particle filter).

The idea behind the correction approach is based on the fact that if we would have the odometry without noise we can use it to perfectly localize the robot. So we developed FFNN for 1D case and CNN for 2D case which reduces the odometry noise. As a result, we achieved stable localization. This approach allows to localize the robot on completely different maps without necessity to retrain neural network. The one drawback of the approach is that training set should be consistent and it should include several maps with different complexity; otherwise, it can fail and diverge on the hard and unseen parts of the environments. While the Kalman Filter uses the raw odometry when the observations are not reliable, the correction approach cannot handle these cases and it uses incorrect observations. However, handling these cases using deep neural networks can be a possible direction for further work. And one possible solution is to train neural network on the concatenated sequence of observations from previous time steps, it will allow CNN to detect outliers in observations.

In general, we can conclude that deep learning holds huge potential for robot localization problem. However, further research is needed to solve some of the limitation highlighted in this work.

References

Amanatides, J., Woo, A., et al. (1987). A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10.

Förster, A., Graves, A., and Schmidhuber, J. (2007). Rnn-based learning of compact maps for efficient robot localization. In *ESANN*, pages 537–542.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Marder-Eppstein, E. (2016). Navigation. `http://wiki.ros.org/navigation`. Accessed: 2016-06-17.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, 45(3):52–57.

University of Bonn (2016). Map benchmarks. `http://www.ais.uni-bonn.de/~holz/spmicp/`. Accessed: 2016-06-18.