NEURAL NETWORK FOR THE IDENTIFICATION OF A FUNCTIONAL DEPENDENCE USING DATA PRESELECTION

V. Hlavac[*]

Abstract: A neural network can be used in the identification of a given functional dependency. An undetermined problem (with more degrees of freedom) has to be converted to a determined one by adding other conditions. This is easy for a well-defined problem, described by a theoretical functional dependency; in this case, no identification (using a neural network) is necessary. The article describes how to apply a fitness (or a penalty) function directly to the data, before a neural network is trained. As a result, the trained neural network is near to the best possible solution according to the selected fitness function. In comparison to implementing the fitness function during the training of the neural network, the method described here is simpler and more reliable. The new method is demonstrated on the kinematics control of a redundant 2D manipulator.

Key words: *neural network, feedforward neural network, function approximation, fitness function, feasibility function, data preselection, data filtering, 2D manipulator, redundant manipulator, redundant manipulator kinematics, inverted kinematics, obstacle avoidance*

Received: May 13, 2019 **DOI:** 10.14311/NNW.2021.31.006
Revised and accepted: April 30, 2021

1. Motivation

A typical use of a feed-forward neural network is the identification of a given functional dependency. A feed-forward neural network (also known as a multilayer perceptron) can perform the interpolation of a functional dependency for most "reasonable" functions. The best behavior is for functions which are continuous including their first derivation. In this case, a feed-forward neural network can interpolate even very noisy data. If the problem is not uniquely determined (it has more degrees of freedom) and if for any combination of potential inputs there are an infinite number of possible solutions, it is necessary to convert the problem to a determined problem by adding other conditions.

A special case in which a neural network is typically used is the inverted problem. The functional dependency from the input to the output (result) is known.

[*]Vladimir Hlavac; Czech Technical University in Prague, Faculty of Mechanical Engineering, Department of Instrumentation and Control Engineering, Technicka 4, Prague 6, 160 00 E-mail: hlavac@fs.cvut.cz

©CTU FTS 2021

The inverted function should calculate which input is required to obtain the required output. Many functions can be analytically inverted. For others, a neural network can be used for their identification. After this, the trained neural network can be used to generate the most probable input for the given output. In reality, the feedforward neural network interpolates the source data using a smooth (multidimensional) function. While not necessary accurate, this feature is important when noisy data are interpolated.

Any undetermined problem can be identified with the use of a feed-forward neural network if there is a condition which compares which solution is better. This condition corresponds to a fitness function used in genetic algorithms (a feasibility function determines which individual is not feasible and should be erased from a population; a fitness function allows the sorting of individuals to select the best). In other methods of artificial intelligence, this function is known as a penalty or cost function. To implement this classification of a potential solution, a reinforced learning algorithm can be used [1]. The problem is that this approach completely denies the use of a batch training method.

In the following text, another approach is proposed and its implementation to solve a redundant manipulator kinematics problem is described. In this case, it is easy to generate a huge amount of data in the possible range of joint angles using a geometric virtual model. A large amount of data allows the implementation of fitness and feasibility functions, as known from the field of genetic algorithms. The proposed method applies the fitness function directly to sort and select the data before the training of the neural network, instead of modifying the network training method. After this data filtering, a standard batch training method can be used.

2. Redundant manipulators

The arm of most 2D manipulators has only two links (segments). This is sufficient to reach any point in the working space. More links cause worse stiffness. There are still many cases where the stiffness and thus the precision with respect to reaching an accurate position are not so important, and, at the same time, there is some limitation on the movement of the arm. A typical example is the car body painting robot, where the car body itself represents an obstacle. The task is to avoid the obstacle (by the whole arm) while keeping the required end-point trace. The typical solution is still to use more robots, usually equipped with an arm with only three links, each of the robots passing a part of the planned trace.

For some tasks for example, car body welding less stiffness would require some advanced real position measurement and control, such as the use of image recognition. However, obstacle avoidance could still be important and creates a very complex problem when using standard methods of position control.

A robotic manipulator with more degrees of freedom than is necessary to reach any required position is called a redundant manipulator [2, 3]. The additional degrees of freedom can be used to form the robot arm into a particular shape, which allows the arm to avoid touching an obstacle. The advantage of a redundant manipulator is its universality.

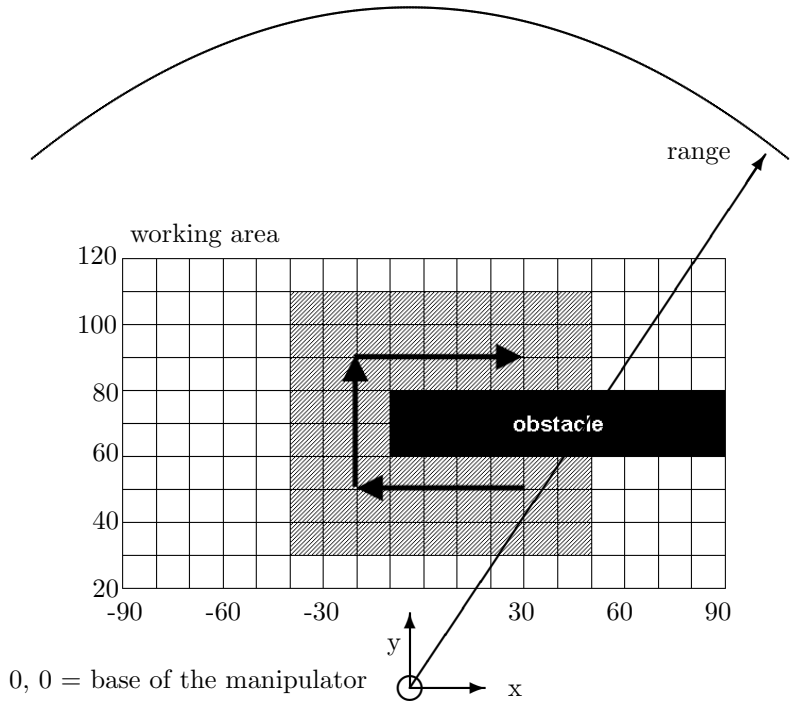

Fig. 1 *Schema of the task. Range of the manipulator* $r = 25 \times 7 = 175$. *The working area is situated in the range of coordinates* $x = (-90, 90)$, $y = (20, 120)$, *where the zero of the coordinates is at the base point of the arm.*

3. Neural network with data preselection

The invention of the described method is based on an innovative form of data preparation. While data are evaluated and filtered, the standard Matlab neural network toolbox is used to prove the functionality of the method.

3.1 Model of a redundant manipulator kinematics

To demonstrate the new method, a very simple model was chosen: a planar manipulator with seven degrees of freedom (example of visualization in Fig. 12). For this test, all of the links of the arm have the same length, **a**, and all joints have a possible angle range $\pm 120°$, denoted by $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_7\}$. Zero coordinates are defined as the position of the first joint (the base of the robot, Fig. 1). The position of the second joint is:

$$\begin{aligned} x_1 &= a \cos \beta_1, \\ y_1 &= a \sin \beta_1, \\ \alpha_2 &= \beta_1. \end{aligned} \tag{1}$$

Using this convention, the position of the i-th joint is

$$\begin{aligned} x_i &= x_{i-1} + a\cos(\alpha_i + \beta_i), \\ y_i &= y_{i-1} + a\sin(\alpha_i + \beta_i), \\ \alpha_{i+1} &= (\alpha_i + \beta_i). \end{aligned} \tag{2}$$

This simple kinematics can be easily programmed. All the problems of obstacle or other link avoidance can be solved in the next step. For testing the algorithm, all the links have the same length, 25 (the program uses no unit).

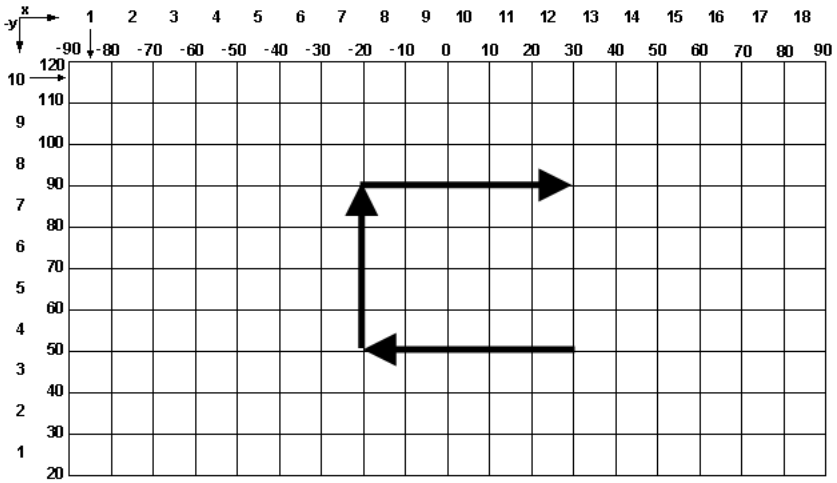

Fig. 2 *Working area, divided into tiles, and the proposed path. Index numbers (the outer scale) represent indices in Matlab, while the inner scale represents the coordinates of the end-point of the manipulator.*

3.2 Generation of the data

A random vector of position, $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_7\}$, is generated in the required range. Because the final use of a neural network can cause some extrapolation, the real possible range of all the actuators should be bigger.

The method in which many possible solutions are randomly generated and only the best is kept is known as the Monte-Carlo method [4], and can be used when finding another approach is problematic, or because it is more reliable (because of its simplicity) than deterministic approaches that are too complex. Monte Carlo can also be used to test the final model [5].

Many artificial intelligence methods are dependent on the quality of the used random generator (function), but, in this case, where data are used for a neural network, which provides interpolation and noise reduction, a simpler generator can be used [6].

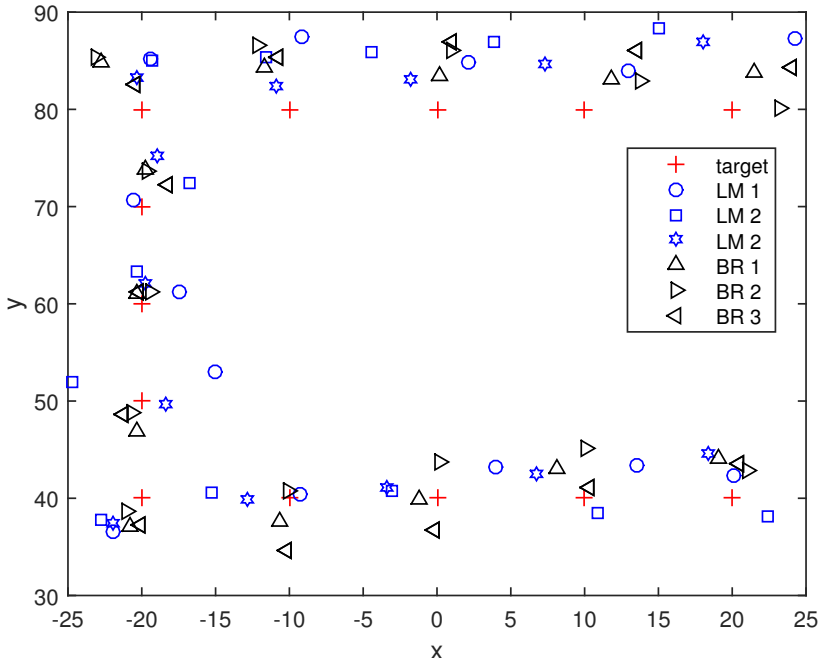

Fig. 3 *Results of the first training. BR (Bayesian regularization, marked in black) fits better than LM (Levenberg-Marquardt, blue). Red crosses represent required position.*

3.3 Feasibility functions

First, a set of feasibility functions should be applied. All impossible settings should be discarded. The first feasibility function disallows any position where one arm will cross another, or pass through an obstacle. The second condition is that the end point of the manipulator must be in the working area. If a final readjustment is required (when using camera control, or just when the exact setting should be adjusted after positioning by the neural network), then the absolute value of the last joint angle (β_7) should be bigger than some value (i.e. 20°) and the previous joint angle should be the same value from its limits (in the described example, in the range of $\pm(120° - 20°) \Rightarrow \pm 100°$).

3.4 Data grouping

The working area was split into many tiles. The goal was that in each of the tiles the best solution (in reality, the three to ten best solutions) would be used to train the network. In this testing example, the working area starts 20 length units above the robot base and contains 10 rows vertically, each 10 units in height. Horizontally, 9 columns to the left and to the right of the axis are defined, representing ±90 length units (Fig. 2). Random combinations of the joint angles are generated and

assigned to Eq. (1) and (2). If the resulting end-point coordinates fall outside this area, the combination is abandoned. Otherwise, it is recorded to the data structure of the indicated individual tile.

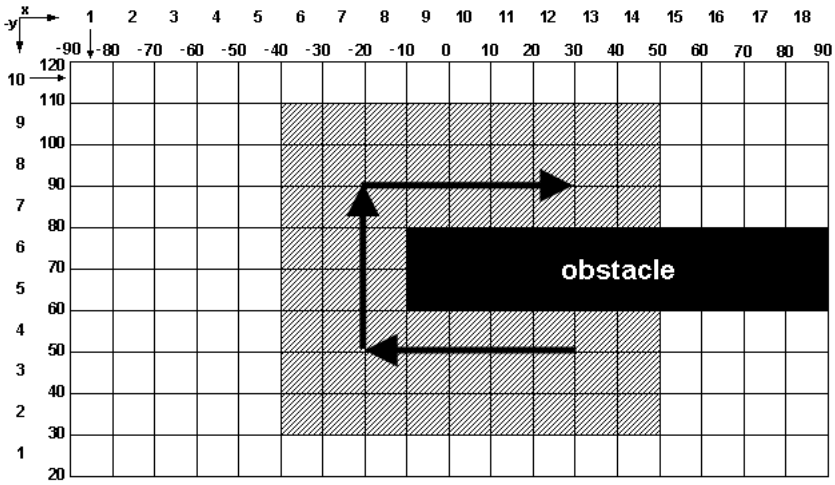

Fig. 4 *Trace (path) with an obstacle. The endpoint of the manipulator should move along this required path, while no part of the manipulator should cross any part of the obstacle, defined by the black marked rectangle.*

In the main cycle of data generation, approximately thousands of possible combinations are generated for each of the tiles. A maximum amount per tile can be set. The data generation is stopped when some preset amount of data cannot be stored to the appropriate tiles, because they are full. The default value is a 1000 abandoned settings.

3.5 Fitness function

When enough data are generated individual generated combinations in each of the tiles are sorted using the required fitness function. In the test described here, the condition of the minimal sum of x-values of all the joints (3) was used.

$$f = \sum_{i=1}^{6} x_i. \tag{3}$$

The end point coordinates were not included.

3.6 Smoothing condition

To maintain the possibility of physical realization, the neighbor solution should represent only small changes of angles of the respective joints (the shape of the manipulator arm should change only very little when moveing a small distance).

The same condition is required if the feed-forward neural network is used for identification. A value of 0.5 radians for any joint was chosen as acceptable. This value is disputable and different values should be checked.

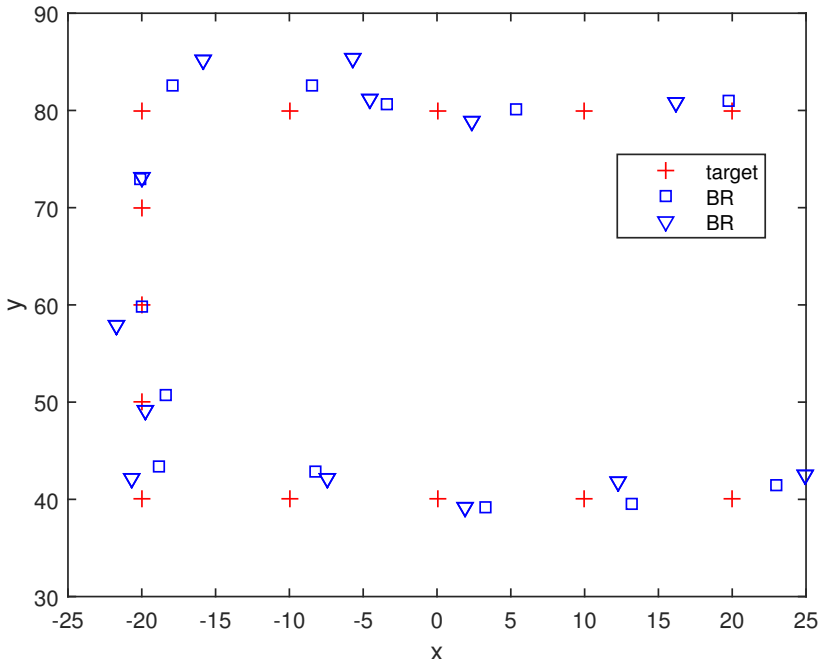

Fig. 5 *Results for new data, generated after definition of the obstacle. Only relevant part of the working area is on the graph. Two BR series (blue) represent two independent trainings of the neural network with the same data.*

First, some tile in the middle of the target area was chosen. In this tile, all combinations which differed in any angle by more than 0.5 radians from the best solution in the tile were deleted.

The next step was to propagate this condition horizontally. Each combination in any tile which differed by more than the chosen 0.5 radians from the best combination of the neighbor tiles was deleted. The content of the tile was then sorted. After sorting, the process was repeated till the end of the line.

After values in the selected row were restricted using the described method, the same process was repeated for each of the vertical columns, starting from the processed row.

During tests, 1 to 3 % of randomly generated solutions survived this process, more of them in the tiles near the starting tile.

3.7 Neural network training

At this point, a standard set of training data was prepared and the standard Matlab nftool toolbox was used. This toolbox guides the user through import data and

parameter selection and offers three training methods (scaled conjugate gradient, Bayesian regularization and Levenberg-Marquardt). The neural network used here is the MLP (multilayer perceptron), with one hidden layer (see Fig. 6). The input layer represents data assignment only. Each neuron in the hidden and output layers executes the function (4):

$$y = f\left(b + \sum_{i=1}^{n} x_i w_i\right), \tag{4}$$

where b is the bias, x_i are the individual inputs (values from the previous layers), w_i are settable weights (trained by the selected algorithm) and f is the transfer function. The hidden layer uses the tansig function Eq. (5), while the output layer uses the so-called "purelin" function (the presence of no transfer function at all, can be represented as multiplication by 1).

$$tansig(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{5}$$

The output layer uses no transfer function (defined in Matlab as the "purelin" function). The general structure of MLP is shown in Fig. 6. This type of neural network is described, for example, in [10], chapter 2.

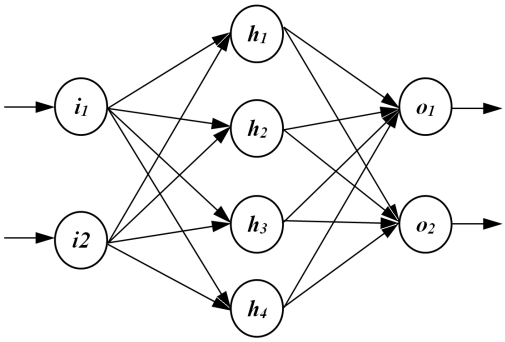

Fig. 6 *Simple MLP (multilayer perceptron) neural network. Neurons are marked* ***i*** *as input,* ***h*** *as hidden and* ***o*** *as output.*

For the described experiment, the output layer had seven nodes (angles to be set to obtain a required position) and two input nodes (required position). The corresponding minimum number of hidden neurons recommended by the Matlab neural network toolbox was 10. In the test, 19 hidden neurons were used. In this configuration, the used neural network has $19 \times (2+1) + 7 \times (19+1) = 197$ settable parameters, including biases.

3.8 Results of training

In this first test of the proposed algorithm, up to 5000 samples per tile were prepared. After all the steps (see points 3.3 to 3.6), the 3 best samples from each of the tiles were used for neural network training. For these prepared data, each method

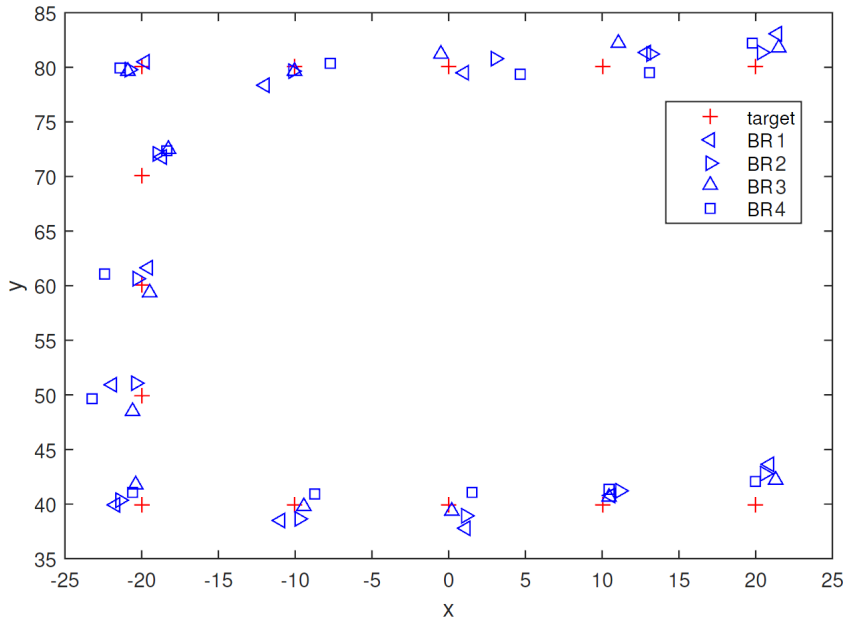

Fig. 7 *Halving the tile size. Up to 5000 samples per tile were generated in the original set and the 7 best selected. Four training experiments (Bayesian regularization used for training the neural network); red crosses represent the required values.*

available in Matlab, was used for neural network training three times. In Fig. 3, the results for Levenberg–Marquardt (LM) and Bayesian regularization (BR) are displayed. For testing the trained results, a simple path (see Fig. 2) was proposed. On this path, 13 points were chosen. The results are shown in Fig. 3. Red crosses represent the required points. These values were used as the neural network input, and angles generated by the neural network were used to calculate the simulated end point value. The respective results are denoted BR and LM (Fig. 3). The third method available in the Matlab nftool, scale conjugate gradient, did not return usable results.

4. Obstacle

In this simple case, a redundant manipulator is a good model for testing this method, but a simpler solution for inverted kinematics can be used [2]. For example, the rectangular coordinates of the destination could be converted to polar, $\mathbf{r}$ and ϕ. To reach exactly this position ($\mathbf{r}$ and ϕ), the length of the manipulator arm has to be shortened by forming a regular arc from all of the links. To achieve this, all of the link angles except the first must be assumed to be the same. Using simple single-value optimization, this angle can be evaluated. The first angle (joint position at the robot base, α_1) will be

$$\alpha_1 = \phi - 6 \cdot \alpha_2. \tag{6}$$

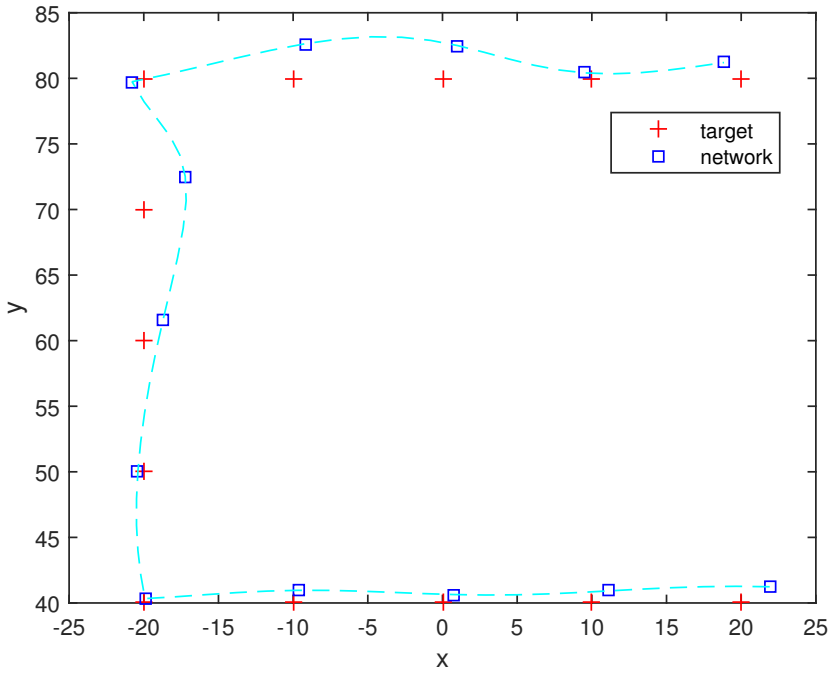

Fig. 8 *One of the series from the previous picture, including a more detailed path. The cyan line is generated using the trained network for points in steps of two units. The neural network can generate positions in any step, but an interpolation is typical for this type of manipulator.*

If the coordinates of the end-point of the manipulator are in the left side of the working area, the negative solution is better. For a more complicated proposition (different lengths of the links or a range of settings for the joints), a pseudo-Jacobian method is often used [7].

None of these simple solutions can be used if there are obstacles in the working area. A possible solution is described in [8]. The method proposed in the previous chapter works, but the feasibility functions have to be changed, and the fitness function should be reformulated.

4.1 A trace with an obstacle

In Fig. 4, the new proposition is defined.

In this second example, data were prepared exactly for the given trace (path). In Fig. 4, the obstacle is in black. The data were prepared for the grey area only. During random data generation, most of the randomly generated combinations for the setting of the joints were abandoned because they fell outside the grey area.

4.2 Feasibility functions

Additional feasibility functions were defined:

Neither any joint of the manipulator nor any part of the arm can be in the obstacle area. The second condition was simplified by assuming the width of the arm to be zero (this simplification can be compensated by extending the obstacle dimensions). For targets above the obstacle, the last joint, connecting the last two segments should have a higher y-coordinate, than the end point.

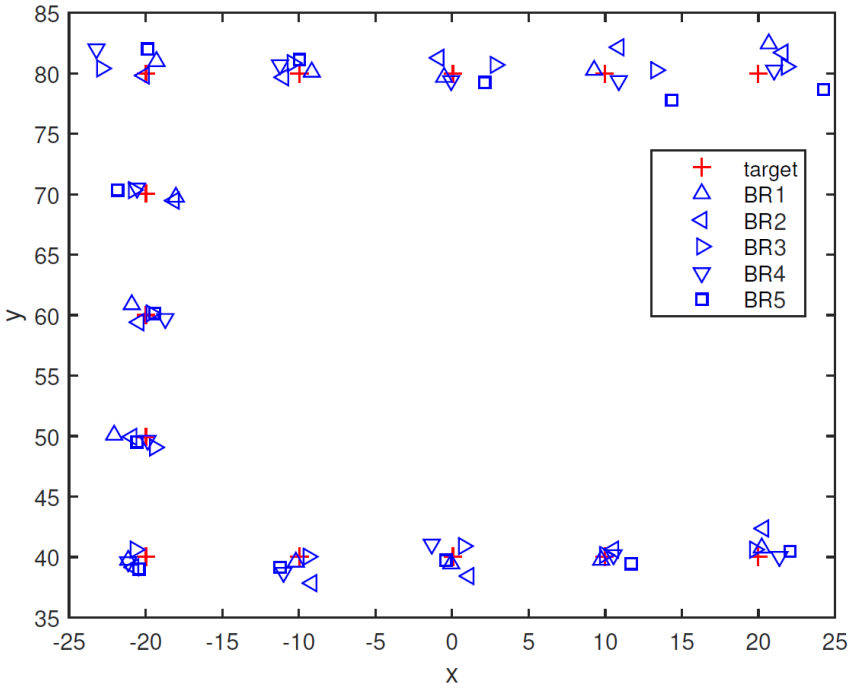

Fig. 9 *Generating even more points did not yield any improvement*

4.3 Fitness function

The original proposition was that the fitness function was evaluated as the minimum distance from the obstacle to any of the manipulator joints. This function ignores the position of the other joints. When using data generated in this way, the training of the neural network failed, because the target area was not continuous. After reformulation, the fitness function sums the Euclidean distances of all joints from the obstacle. Data generated in this way were useable with respect to training the network, although this definition was shown not to be perfect (Fig. 5; an example of visualization in Fig. 12). For this test, ten best points for each of the selected tiles (in the gray area only) were used, i.e. $(8 \times 9 - 12) \times 10 = 600$ samples.

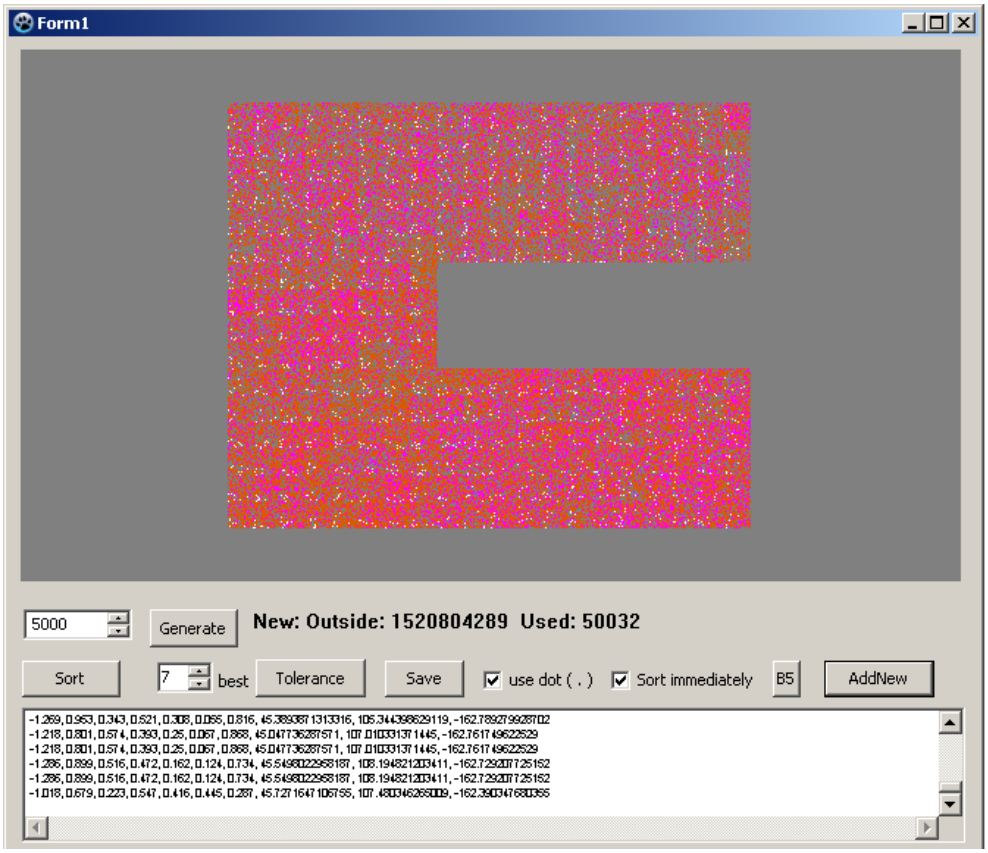

Fig. 10 *Generator interface. Pink points persisted after smoothing, orange points were later generated using additional feasibility functions. Finally, white points were selected for neural network training.*

4.4 Finer division

The first modification was to reduce the tile size to 5, half of the original. This gave more points to the neural network training algorithm (seven best from each of the tiles selected; $(16 \times 18 - 4 \times 12) \times 7 = 1680$). This better responded to the number of parameters of the network, which was 197 (for 19 neurons in the hidden layer and 7 in the output layer: $19 \times (2+1) + 7 \times (19+1) = 197$, including biases). The result is presented in Fig. 7. The generation of the source data took three times as long (about 30 minutes; up to 5000 samples per each of the tiles before selection; data for the tiles outside the gray area (Fig. 4) were abandoned using the modified feasibility function).

4.5 Improved data generation

The smoothing condition, defined in section 3.6, keeps the change of any joint angle between two neighboring tiles to some value (0.5 radians in this case). After

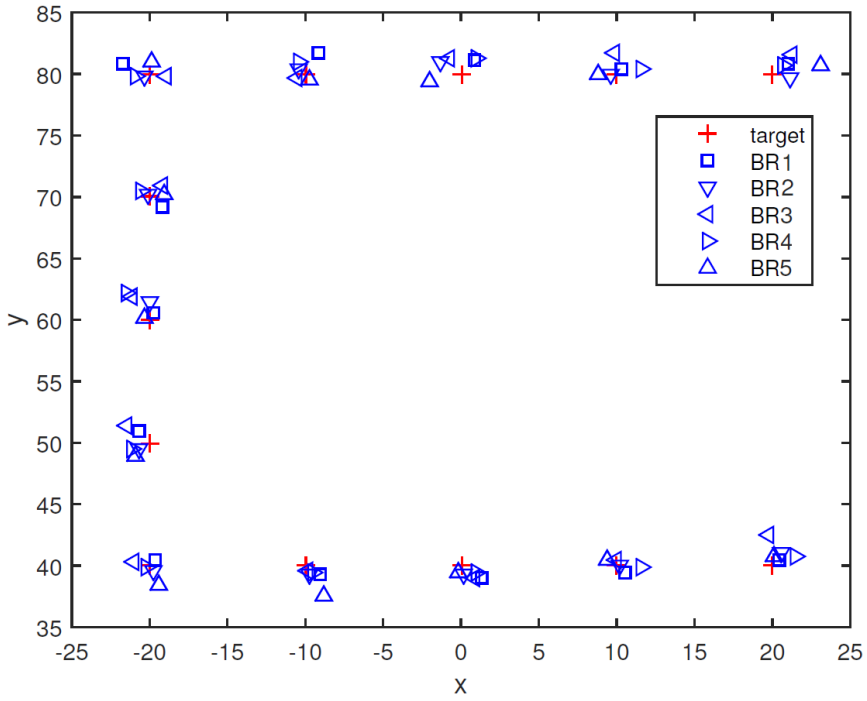

Fig. 11 *Results of neural network after generation of additional points. Generation was interrupted and data were sorted; then the best was selected, the smoothing condition was implemented, and the generation of new solutions continued, but with the smoothing condition as another (additional) feasibility function.*

application of this condition, the possible range for each of the angles recorded to this tile is known, so this new condition can be used during the generation of new points.

Because a better solution than the current best can be found, it is reasonable to repeat the whole sequence (including sorting and new smoothing). The result after new data generation and new network training is shown in Fig. 11. The variability of the training of neural networks is lower and the precision of the position is better in comparison with Fig. 7.

In the next step, the process of adding new data was repeated. The generator interface window after this step is shown in Fig. 10. The new set of data was used for neural network training. The results of the newly trained networks are shown in Fig. 9. In comparison with the previous result, only a small improvement is evident, while the most problematic point, $(-20; 70)$, is even worse. Further improvement by additional data generation would probably be very small, as the method reached the limits of the current setting. Possible further improvement can be achieved by changing the fitness function.

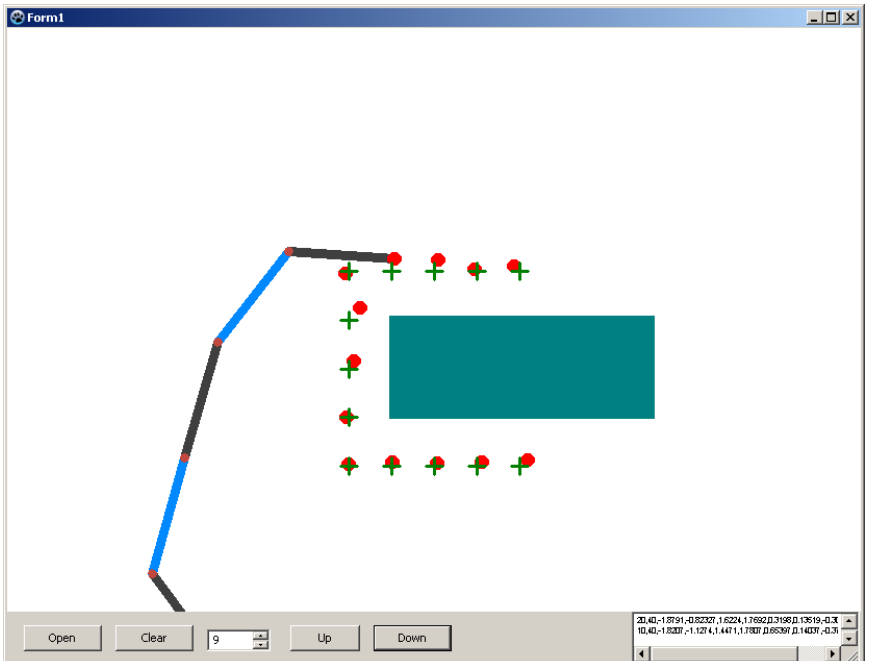

Fig. 12 *Tool for visualization. The thick line represents the arm of the manipulator; segments are in black and blue color. The program prepared using Lazarus IDE provides visualization of the movement along the required trace (path), marked by the green crosses. For each of positions, red dots are placed. The screenshot was taken after a few cycles of movement; at the start there was no red dot.*

5. Conclusion

5.1 Redundant manipulator

Positioning accuracy can be measured in terms of the error of the end point position relative to the required position. In Tab. I, this relative error is evaluated as an absolute error divided by the trained range (90 horizontally, 80 vertically). Compared to the overall range of the manipulator, which was 175 at the highest vertical approach, twice more than the horizontal, the error would appear even smaller. If further improvement is required, the employed feed forward neural network can be further trained using standard back propagation. In a real application, only the points near to the required path (the tiles surrounding the required path) can be used for this additional training. If a limited number of cycles of the back-propagation algorithm are used, then there is a high probability that a manipulator which is controlled by a neural network will maintain obstacle avoidance. After additional training, this feasibility condition should be checked again. The method described here can be a good alternative to other methods of neural network control used for the control of manipulators [3, 9].

target x	target y	nn x	nn y	error x	error y	error %
−10	80	−9.115	81.660	0.885	1.660	2.30
−20	80	−21.753	80.858	−1.753	0.858	2.22
0	40	1.273	38.972	1.273	−1.028	1.91
0	80	0.811	81.077	0.811	1.077	1.62
20	80	21.020	80.852	1.020	0.852	1.56

Target values were applied as inputs for the tested neural network. Angles (the output of the neural network) were inserted into a geometric model. Resulting end point coordinates are in the next two columns. Errors were counted as the difference between required and resulting positions. Relative error was counted as the Euclidean distance between the required and resulting positions relative to the size of the diagonal of the positive half of the working area (Fig. 2).

Tab. I *Example of errors of the training.*

5.2 Neural network with data preselection

This example proves that if an inverted function of some problematic function is required, its values can be generated and substituted by a neural network. If the function is redundant, another condition for selecting a desirable solution can be (and must be) added. If there are more conditions, they can be aggregated to one fitness function. Even a highly non-linear function and its adjoined feasibility and fitness functions can be used.

In many cases, as in the example of a redundant manipulator, data for training a neural network can be prepared using simulation, while executing a pre-trained neural network can be very fast. After theoretical evolution of the target position, the required exact positioning can be achieved by a small change of two angles. In real conditions camera monitoring should be used, because the stiffness of this type of manipulator is typically low and requires feedback for an accurate setting. This feedback (readjustment) can also comprise the error in this method.

References

[1] GHANBARI A., VAGHEI Y., NOORANI S. Reinforcement Learning in Neural Networks: a Survey. in: *International journal of Advanced Biological and Biomedical Research*, 2(5), 2014, pp. 1398–1416.

[2] JIN L., LI S., YU J., HE J. Robot manipulator control using neural networks: A survey. In: *Neurocomputing*, 2018, 285, pp. 23–34.

[3] LI S., ZHANG Y., JIN L. Kinematic Control of Redundant Manipulators Using Neural Networks. In: *IEEE Trans. on Neural Networks and Learning Systems*, 2017, 28(1), pp. 2243–2254.

[4] RUBINSTEIN R.Y., KROESE D.P. Simulation and the Monte Carlo Method (2nd ed.). New York: John Wiley & Sons. ISBN 978-0-470-17793-8.

[5] GOLASOWSKI M., LITSCHMANNOVA M., KUCHAR S., PODHORANYI M., MARTINOVIC J. Uncertainly modeling in rainfall-runoff simulations based on parallel Monte Carlo method. In: *Neural Network World*, 3/15, pp. 267–286.

[6] BRANDEJSKY T.: Influence of (p)RNGs onto GPA-ES behaviors. In: *Neural Network World*, 2017, 27(6), pp. 593–605. ISSN 1210-0552, doi: 10.14311/NNW.2017.27.033

[7] HOLLERBACH J., SUH K. Redundancy resolution of manipulators through torque optimization. In: *IEEE J. Robot. Automat.*, 1987, 3(4), pp. 308–316.

[8] DUNCOMBE J.U. Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. In: *Robotics and Computer-Integrated Manufacturing*, 2012, 28(2), pp. 132–146.

[9] RAHMANI A., GHANBARI A., MAHBOUBKHAH M. Kinematics analysis and numerical simulation of hybrid serial-parallel manipulator based on neural network. *Neural Network World*, 4/15, pp. 427–442.

[10] BENNETT K. MATLAB: Applications for the Practical Engineer. IntechOpen, 2014. ISBN 978-9535117193.