

Inverted Kinematics of a Redundant Manipulator with a MLP Neural Network

Vladimir Hlavac

Faculty of Mechanical Engineering
Czech Technical University in Prague
Prague, Czech Republic

hlavac@fs.cvut.cz, <https://orcid.org/0000-0002-8794-3444>

Abstract— The article describes the solution of the inverse kinematics of a serial redundant manipulator. Reachable endpoint positions are generated randomly based on forward kinematics, described by Denavit–Hartenberg notation. If the randomly generated position is part of the area in which the desired movement is to be solved, it is recorded in a special structure where each cell corresponds to a small range of the endpoint coordinates. Up to thousands of possible combinations can be recorded in each of the cells. Based on this data, inverse kinematics cannot be solved for a redundant manipulator because the same point can be reached by infinitely many combinations of arm settings. Therefore, the prepared angle settings for reaching an individual cell are first evaluated with a suitable additional fitness function. Additionally, solutions that do not represent continuous movement are filtered. After this process, described in this article, the few best solutions are then selected from each of the cells and used to train a simple MLP (multilayer perceptron) neural network. Based on data from forward kinematics, the network is trained to obtain an inverse kinematics solution. The result is a smooth motion whose accuracy is limited by the cell size used and the amount of samples generated.

Keywords— Neural network, Redundant manipulator, Inverse kinematics, Serial manipulator, Articulated manipulator, Multilayer perceptron neural network

I. INTRODUCTION

Inverse kinematics of serial manipulators is a challenging, but often solved problem [1]. The current state of the arts is described in [2]. An interesting alternative for solving systems of equations is the use of an MLP neural network to solve inversion [3]. An inverse function exists if one desired location can be reached by only one possible combination of angles. Fig. 1 shows the situation of a simple planar manipulator with two links that reaches the same endpoint with two different combinations of angles. Redundant manipulators have more degrees of freedom than

needed to reach the required position and angle of the end-effector in the required part of the workspace. In this case, there are infinitely many possible combinations, and the inverse function cannot be directly determined.

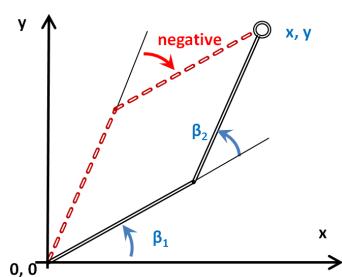


Fig. 1. Two variants with the same end-effector coordinates of a non-redundant manipulator

The first article of this series [4] describes a solution where random joint angle settings are generated and these combinations are recorded in a data structure, where a single cell records the possible combinations that can achieve a given range of the end-effector coordinates. Since [4] solves a planar manipulator, the individual cells represent tiles in the region of the required range of motion. Solutions randomly generated outside the desired range of motion are not stored. Each cell can record at least thousands of possible solutions for a given range of end-effector positions.

Generation is terminated if the capacities of some cells are overflowing. Then, in each cell, the solutions are sorted according to some additional fitness function and the best ones are used.

In order to train the neural network, the generated motion must be continuous. This is achieved by applying a smoothing function. The solution of this problem is described in [4]. As the first step, all solutions in the least accessible cell (usually at the end of the desired motion) are sorted according to the fitness function. Then, the best one is selected, and all solutions whose joint angles differ from this solution by more than the allowed value are erased from the cell. In the next step, all the solutions in the neighboring cells, which differ more than some allowed value (in this case, a little bigger), are erased too. This process is repeated in the next cell in the trace of the required motion. If more data is required for the neural network training, this process can be repeated with surrounding cells (near the required trace).

The resulting data is used to train an MLP (multilayer perceptron) neural network with one hidden layer (compared to a deep neural network, training is very fast, on the order of tens of seconds). A follow-up paper [5] describes how the fitness function can simultaneously solve the obstacle avoidance problem.

The accuracy of the described solution depends on the amount of data generated for each cell. Refinement can also be achieved by reducing the size of individual cells. Both quickly increase the amount of data to be processed. For [5], data preparation took ten times longer than training the neural network itself. A possible solution to this problem is described in [6]. Random settings are again generated, and possible solutions are recorded, but only if the solution belongs to a larger area around the last point of the required move. In this case, all data is recorded in a single cell. After capturing a sufficient number of solutions, the data is sorted by the fitness function. Subsequently, only one single solution is selected and a cell structure with very fine

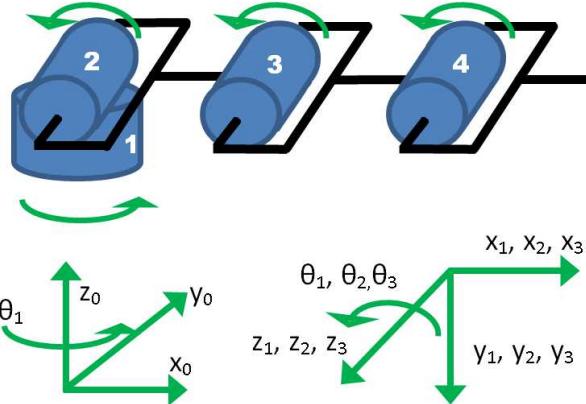


Fig. 2. Manipulator with 4 DOF.

subdivision is generated. The only selected solution is recorded into the appropriate cell and on the base of this solution, by making small changes to the joint angle settings, new solutions are generated. According to their end-effector coordinates, most of them are saved into the neighboring cells of the prepared structure. A similar procedure is repeated in each cell, first towards the required end point of the motion (the starting point may be generated off the required trace of motion) and then in the opposite direction to the required motion.

In this paper, an extension of the solution for serial 3D manipulators is described. Here, the cell structure must be three-dimensional, and the data generation other than that described in the previous paragraph [6] is not practically possible.

To describe the movement of a simple redundant serial 3D manipulator, it is best to use the standard Denavit-Hartenberg convention [7] [8], where the movement is described using suitably chosen partial reference coordinate systems (of each of the nodes) in such a way that rotation only takes place around the x and z axis and displacement again only in the x and z direction, which allows the use of pre-solved rotation matrices (1) [1].

$${}^{n-1}T_n = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The entire calculation of the position of the end effector is thus divided into repeated multiplication of matrices, each

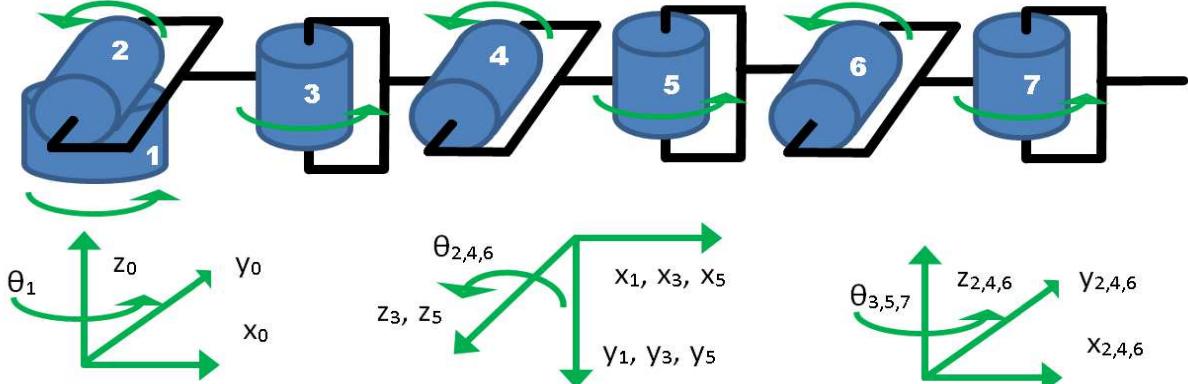


Fig. 3. Manipulator with seven degrees of freedom (7DOF)

TABLE I. THE DH PARAMETERS OF THE 4DOF CASE (FIG. 1)

segment	θ	α	r	d
1	θ_1+0	$\pi/2$	0	0
2	θ_2+0	0	8	0
3	θ_3+0	0	7	0
4	θ_4+0	0	6	0

TABLE II. THE DH PARAMETERS OF THE 7DOF CASE (FIG. 2)

segment	θ	α	r	d
1	θ_1+0	$\pi/2$	0	0
2	θ_2+0	$\pi/2$	8	0
3	θ_3+0	$-\pi/2$	7	0
4	θ_4+0	$\pi/2$	6.1	0
5	θ_5+0	$-\pi/2$	5.3	0
6	θ_6+0	$\pi/2$	4.6	0
7	θ_7+0	$-\pi/2$	4.1	0

corresponding to one adjustable parameter, most often the angle of rotation at the given node. During data generation itself, the solution space is then divided into cells (3D equivalent of tiles from [4]), each representing a certain range of x, y, and z coordinates. The solution is written into the appropriate cell, corresponding to the randomly reached x, y and z coordinates of the end-effector. A finer division here leads to a more precise achievement of the desired position, but as the cell size decreases, the amount of data which must be evaluated increases very quickly.

In the first example described in the next chapter, only the coordinates of the end-effector are solved. In the second one, end-effector orientation in space (direction) is also solved. Using the described methodology, it would require the construction of a six-dimensional cell structure. Here, an innovative solution is proposed where the end-effector direction is included in the modified fitness function.

II. SOLVED PROBLEMS

A. Endpoint movement of 4 DOF manipulator

The first example is the manipulator with a rotating base. The other segments actually represent a planar manipulator with three segments, with which it is rotated in space (Fig. 2).

Similar kinematics has a simple excavator [9]. Note, that this system has only four degrees of freedom. It can reach any point in its working area, but it cannot set most of the angles. If we solve only the end point coordinates, then from this point of view, this manipulator is redundant.

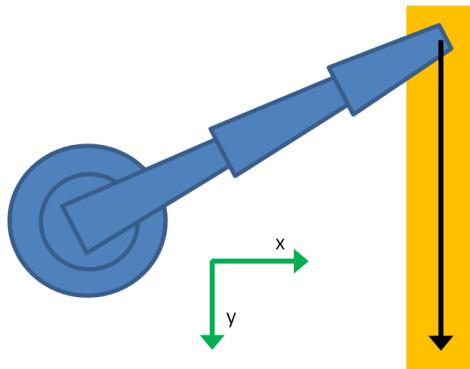


Fig. 4. Required trace (black arrow) and area of solution (yellow)

The base point is selected as a common point of rotation of the base and the first link. All other rotations are selected to be on the x axis (for the first one z axis) of the previous element, so all displacements are zero (last column of Table I.).

B. End effector movement and position of 7 DOF manipulator

The second example represents a more general 3D manipulator. The schematic in Fig. 3 shows the arm in the position at zero theta angles. From this position, it is most obvious that such a manipulator lacks one degree of freedom, the rotation of the end effector around the x-axis. It is therefore twice redundant with seven degrees of freedom.

The base point is again selected as a common point of rotation of the base point and the first link. All other rotations are selected to be on the x axis (for the first z axis) of the previous element, so all displacements are zero (last column of Table II.).

C. The task

The easiest possible task was chosen for both manipulators, the movement of the end point was along the y-axis in the specified range. In the second task (manipulator with 7DOF), the additional requirement was added: the direction of the last link should be in the direction of the x axis (of the basement coordinates). This corresponds, for example, to the position of the spray gun for applying paint when moving along the painted object. The schematic top view is on Fig. 4.

The black arrow on Fig. 4 indicates the intended path of

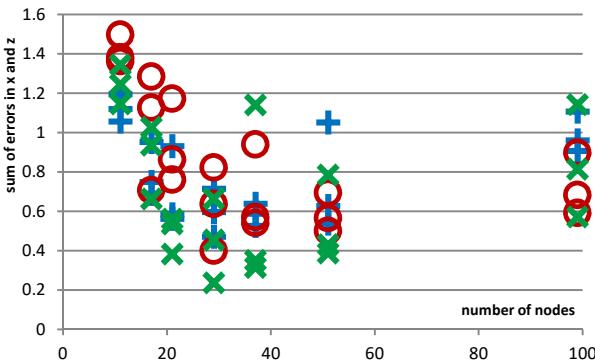


Fig. 5. Errors evaluated after training networks for the 4DOF case.
Sum of errors in the x axis (+ symbol), y axis (o) and z axis (x)

movement. The yellow rectangle represents the projection of the space for which the data is generated for the individual arm end-effector positions. The span of space for which the data is generated in the z-coordinate (not shown) is the same as in the horizontal y-coordinate, and contains two additional layers of cells in addition to the central cell. Data are therefore stored for which the coordinates after rounding to integers are in the range $x=[16 .. 20]$, $y=[-11 .. 11]$ and $z=[-2 .. 2]$ for the first (4DOF) manipulator, respectively $x=[19 .. 23]$, $y=[-11 .. 11]$ and $z=[-2 .. 2]$ for the second case (the size of a cell was selected to be 1 in all axes). The movement is from the point $[18, -10, 0]$ to the point $[18, +10, 0]$, or from the point $[21, -10, 0]$ to the point $[21, +10, 0]$ in the second case. All thetas (the joint angles) were generated in the range $\pm\pi/2$.

III. SOLUTION OF THE PROBLEM

A. Description of used methods

The procedure described in [4] is possible, but it would represent too large number of randomly generated manipulator positions. If a thousand possible solutions are recorded for each cell in the part of the workspace where we are looking for a solution, that is only $5 * 5 * 23 = 575,000$ solutions, but since this is only a small slice of the workspace, hundreds of billions of solutions would have to be generated. Therefore, the variant described in [6] is used.

The algorithm starts by randomly generating several billion solutions until several solutions are found in the motion start region. It can even be an area that includes multiple neighboring cells (ref. [6] even uses a method where a suitable solution is selected and then the mesh of cells is refined). Based on this cell, other solutions are then generated as small changes to the individual angles of this solution. With appropriate settings, solutions are obtained that the algorithm places in neighboring cells. The solutions are always immediately sorted according to the fitness function, so the solution for the next cell is always based on the few best solutions from previous cells. It is then continued with other cells in the direction of the planned movement of the endpoint, if necessary, in a wider area. Even in this case, it is not possible to avoid filtering the solution according to the continuity condition, because after several cycles, a solution may be found that is better according to the fitness function, but does not meet the condition of continuity (we repeatedly generate new solutions on the basis of already filtered cells; the newly generated solutions are saved to neighboring cell, some of

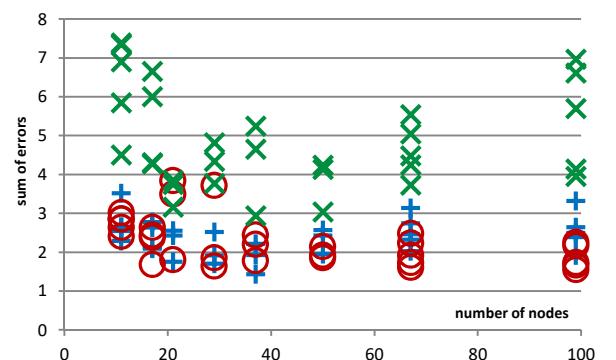


Fig. 6. Errors evaluated after training networks for the 7DOF case.
Sum of errors in the x axis (+ symbol), y axis (o) and z axis (x)

TABLE III. SUM OF ERROR OF SELECTED NETWORK FOR SOLUTION OF THE 4DOF MANIPULATOR. PRESENTED RESULTS FOR THE NETWORK FOR WHICH THE ERROR IN THE X+Z AXIS (3) IS THE MEDIAN OF ALL THE TRAININGS WITH PRESENTED NUMBER OF THE HIDDEN NODES.

nodes	e_x	e_y	e_z	e_{xz}
11	1.119	1.382	1.244	1.673
17	0.749	0.706	0.936	1.199
21	0.562	0.760	0.560	0.794
29	0.597	0.400	0.455	0.750
37	0.638	0.540	0.350	0.728
51	0.626	0.566	0.430	0.760
99	0.961	0.681	0.814	1.259

TABLE IV. SUM OF ERROR OF SELECTED NETWORK FOR SOLUTION OF THE 7DOF MANIPULATOR. PRESENTED RESULTS FOR THE NETWORK FOR WHICH THE ERROR IN THE X+Z AXIS (3) IS THE MEDIAN OF ALL THE TRAININGS WITH PRESENTED NUMBER OF THE HIDDEN NODES.

nodes	e_x	e_y	e_z	e_{xz}
11	2.641	3.022	7.345	7.805
17	2.637	2.385	6.655	7.159
21	2.433	3.846	3.858	4.561
29	1.917	3.718	3.769	4.229
37	1.928	2.206	2.938	3.514
50	2.432	1.919	4.243	4.890
67	2.741	2.237	4.484	5.256
99	2.644	1.670	6.965	7.450

them in the next cell on the path of the required movement, so it could be used in the next cycle). Successive repetition generates solutions that are close to the maximum achievable value of the fitness function for the given cells (provided that the continuity condition is met).

For the 7DOF manipulator, where the direction of the last link is a part of the fitness function, a better fitting of this requirement can be achieved by repeating the process of generating and filtering new solutions. In this case, it has been proven, that if the direction of the processing of the cells is changed for each of this outer cycle, than convergence to the best achievable solution (mainly for the cell at the starting point of the procedure) is faster.

B. Results of testing the proposed procedure

The data was generated by a program created using the Lazarus IDE [10]. The source code of the program is available at <http://users.fs.cvut.cz/hlavac/DH/>. The program was supplemented with a simplified visualization, including the resulting movement according to the prepared script. Matlab was used to train the neural network. The resulting data was inserted back into the original program, where the angles were again converted to coordinates with the forward kinematics and the accuracy of the achieved position was evaluated. The prepared program is single-purpose. To change any parameters, a change of the source code is required. The advantage of the solution is the high speed of data generation, due to the use of FPC [10] as a compiler.

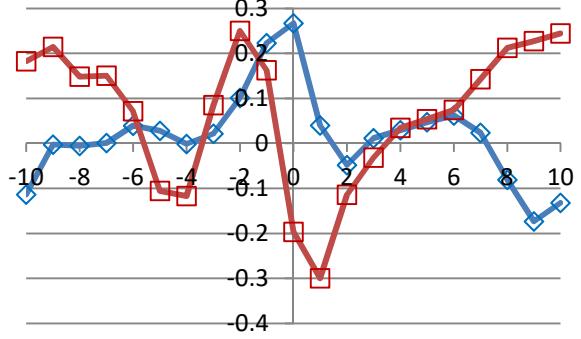


Fig. 7. Errors in trajectory when controlled by the trained neural network. Horizontally y-position, blue line and diamond shape markers show the x-axis deviations, red line and squares z-axis deviations (4DOF).

The error was defined as the sum of the absolute values of all deviations in a given coordinate for the simulated 23 points, e.g. for the x-axis:

$$e = \sum_{i=0}^{23} |x_i - x_{i \text{ required}}| \quad (2)$$

The results for several network trainings are in Fig. 5 and Fig. 6. Multiple attempts of training were made for the neural network representing the inverse kinematics of the seven-DOF manipulator.

$$e_{xz} = \sum_{i=0}^{23} \sqrt{(x_i - x_{i \text{ req}})^2 + (z_i - z_{i \text{ req}})^2} \quad (3)$$

For a given movement trajectory, there are significant deviations in the x and z axes. The Table III and Table IV show selected values calculated according to the formula (3) for different trained networks. Median values from several trained neural networks are selected (the table row represents the neural network whose total error was the median of multiple networks - average is not a suitable function when selecting a single result of training to be presented).

The average relative setting error can be calculated by dividing this sum by the number of set points (23) and then by the total length of the manipulator arm (21 or 35.1) and converting it to percentages. The average is around 0.2% for the simpler task and 0.6% for the 7 DOF manipulator. When the best interpolating neural network is evaluated, the error is approximately half of the presented value.

IV. RESULTS

The accuracy of the adjustment of each position relative to the arm length is on average 0.2% for the simpler manipulator and 0.6% for the DOF7 manipulator. In the latter case, probably more data should have been generated, or a finer subdivision cell structure could have been used.

The actual data generation for general manipulators may be more time-consuming. However, the method itself uses simulation, and the preparation of the neural network takes place independently of the real manipulator. Thus, the control program can be prepared in advance, so the time requirement itself is not significant. In the simplified cases presented, the time consumed was on the order of minutes. It takes a comparable amount of time to train the neural network.

The use of neural networks for solving inverse kinematics represents a useful alternative to analytical

solutions for the motion of these manipulators, not requiring the solution of difficult mathematical relations. Together with the appropriate simulation, they reduce the possibility of human error in programming these devices.

Redundant serial 3D manipulators are not common; to solve problems analogous to [5], such as the obstacle avoidance, they would have to have too many links, causing worse arm stiffness and decreasing the setup accuracy. However, a redundant manipulator is, for example, a model of a human arm [11], or robot arms in cases where redundancy serves as a backup in case of failure, such as in reconnaissance robots (military, rescue). In this case, the [6] algorithm is advantageous, as it allows to quickly compute a suitable path from a given position to the next desired position. The best solutions found along the desired path can then be used even without training the neural network, if the discontinuous and unsmoothed motion of the arm is not a problem.

V. CONCLUSION

In this paper, the possibility of solving inverse kinematics using a neural network even for 3D manipulators has been demonstrated. At the same time, a method has been proposed to provide the desired direction of the end-effector in space by modifying the fitness function, so that the three-dimensional cell structure is still sufficient for data generation.

VI. REFERENCES

- [1] F. Xiao, G. Li, D. Jiang, Y. Xie, J. Yun, Y. Liu, L. Huang and Z. Fang, "An effective and unified method to derive the inverse kinematics formulas of general six-DOF manipulator with simple geometry," *Mechanism and Machine Theory*, vol. 159, 2021.
- [2] I. Agustian, N. Daratha, R. Faurina, A. Suandi and Sulistyaningsih, "Robot Manipulator Control with Inverse Kinematics PD-Pseudoinverse Jacobian and Forward Kinematics Denavit Hartenberg," *Jurnal Elektronika dan Telekomunikasi (JET)*, vol. 21, no. 1, pp. 8-18, August 2021.
- [3] S. Li, Y. Zhang and L. Jin, "Kinematic Control of Redundant Manipulators Using Neural Networks," *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*, vol. 28, no. 10, pp. 2243-2254, 2017.
- [4] V. Hlavac, "Neural Network for the identification of a functional dependence using data preselection," *Neural Network World*, vol. 2, p. 109–124, 2021.
- [5] V. Hlavac, "MLP Neural Network for a Kinematic Control of a Redundant Planar Manipulator," in *Advances in Mechanism Design III*, Springer Nature, 2022 (in print).
- [6] V. Hlavac, "Kinematics Control of a Redundant Planar Manipulator with a MLP Neural Network," in *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)* doi: 10.1109/ICECCME52200.2021.959108, 2021.
- [7] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 22 (2): 215–221. doi:10.1115/1.4011045., 1955.
- [8] R. S. Hartenberg and J. Denavit, "Kinematic synthesis of linkages," *McGraw-Hill series in mechanical engineering. New York: McGraw-Hill.*, p. 435, 1965.
- [9] B. Patel and J. M. Prajapati, "Kinematics of mini hydraulic backhoe excavator - part II," *International Journal of Mechanisms and Robotic Systems*, vol. 1, no. 4, pp. 261 - 282, 2013.
- [10] "Lazarus," [Online]. Available: <https://www.lazarus-ide.org/>. [Accessed 17 7 2022].
- [11] W. Dewandhana, K. I. Apriandy, B. S. B. Dewantara and D. Pramadihanto, "Forward Kinematics with Full-Arm Analysis on "T-FLoW" 3.0 Humanoid Robot," in *2021 International Electronics Symposium (IES)*, Surabaya, Indonesia , 2021.