A NAO robot playing tic-tac-toe

Comparing alternative methods for Inverse Kinematics

Renzo Poddighe Nico Roos

Department of Knowledge Engineering, Maastricht University

Abstract

In this paper, two alternative methods to the Inverse Kinematics problem are compared to traditional methods regarding computation time, accuracy, and convergence rate. The test domain is the arm of the NAO humanoid robot. The results show that FABRIK, a heuristic iterative approximation algorithm outperforms the two traditional methods, which are both based on the Jacobian inverse technique, on all aspects. An artificial neural network vastly outperforms all algorithms regarding computation time, but lacks accuracy. To demonstrate the practical applicability, FABRIK has been used enable a NAO humanoid robot to play tic-tac-toe.

Keywords: Humanoid robot, inverse kinematics, machine learning.

1 Introduction

The Inverse Kinematics (IK) problem is a non-linear optimization problem which tries to optimize the position of the end effector of a robot's kinematic chain with respect to a certain target position, by manipulating the intermediate joint configurations in the chain [7]. These joint configurations have to be calculated in a backwards manner from the Cartesian coordinates. Since not all points in Cartesian space map to a joint configuration, there is not always a solution. It is very exceptional for a kinematic chain to have a complete analytically derivable solution. Therefore, Inverse Kinematics solvers rely on numerical approaches.

Traditional methods The most commonly used method is using the inverse of the Jacobian matrix, a matrix of first-order partial derivatives of the joint system, to make a linear approximation of the non-linear function that describes the Inverse Kinematics. Since the Jacobian is not always nonsingular, the inverse can be approximated by a number of methods [3], each of which has its drawbacks. The transpose of the Jacobian is proven to be a good replacement for the inverse [12]. However, the joint configurations are often unpredictable and many iterations are needed for convergence. The Moore-Penrose pseudoinverse of the Jacobian is a better estimate, but often performs poorly because of instability for configurations near singularities. The Damped Least Squares (DLS) method avoids the use of pseudoinverses and provides a more stable solution near singularities. A damping constant has to be chosen carefully: a larger constant makes the algorithm more numerically stable, but also lowers the convergence rate. Pseudoinverse DLS attempts to overcome this problem by applying Singular Value Decomposition. Pseudo-inverse DLS performs similar to regular DLS away from singularities, and smooths out the performance of regular DLS near singularities. Selectively Damped Least Squares (SDLS) is an extension of Pseudoinverse DLS that also takes into account the relative positions of the end effector and the target position when choosing the damping constraint, resulting in fewer iterations needed for convergence, but a slower performance time. A general problem with all these methods is applying constraints, which is not at all straightforward to do using any Jacobian method. The existing methods do not guarantee optimal solutions and slow down performance times [4, 11]. When coping with computational limitations on a robot, it is therefore appealing to search for effective alternatives that have low on-line computational cost and are robust against singularities. In

this paper, two alternative approaches are proposed and compared on several criteria, such as speed and precision. The standard algorithm to which we compare the two alternative techniques, Jacobian transpose and the Jacobian pseudoinverse.

Alternatives In this paper, two alternatives are proposed.

FABRIK [1] (short for Forward And Backward Reaching Inverse Kinematics) is a heuristic iterative method that tries to solve the IK problem by treating the joint coordinates as being points on a line. It uses the previously calculated positions of the joints to find the updates in a forward and backward iterative manner. The algorithm has low computational cost and converges quickly. Furthermore, FABRIK does not suffer from singularity problems, since the use of matrix inverses is completely avoided.

A Neural Network is a supervised learning method that is inspired by the interconnections between the neurons in the brain [10]. It allows for *non-linear* function approximation and is therefore natural candidate for tackling a problem such as Inverse Kinematics[8].

In this paper we investigate whether FABRIK or a Neural Network could be used as alternative for traditional approaches for Inverse Kinematics.

Test and application domain As a test domain we use the arm of the NAO humanoid robot, developed by the French company Aldebaran Robotics. Moreover, we applied the research results in an application that enables the NAO to play the game of tic-tac-toe against a human. The NAO has to recognize the game state on a white board, determine its action, and draw a cross or a circle depending on whether it had the first move. The research results reported in this paper were used in the drawing of crosses and circles.

Outline The remainder of the paper has the following outline. In the next section we will give a brief introduction in Kinematics and Inverse Kinematics. Moreover, we will describe the traditional Jacobian inverse technique for Inverse Kinematics. Section 3 and 4 describe the alternative approaches that we investigated, FABRIK and Neural Networks respectively. Section 5 describes the experiments and Section 6 describes the application we implemented in which we applied the reported research of the preceding section. Section 7 presents the conclusions that can be drawn from the paper and points out directions for further research.

2 Preliminaries

Forward Kinematics Forward Kinematics can be described as the problem of calculating the position of the end effector (or any other joint) of a kinematic chain from the current joint angles of that chain. In other words, Forward Kinematics is the problem of mapping the joint space of a kinematic chain to the Cartesian space.

A kinematic chain consists of links connected by joints. With each link we associate an coordinate system where the one of the axes is normally chosen in the direction of the link. The *kinematics equations* [7] describe transformation of the coordinate system of one link into the coordinate system of a connected link. These equations describe *affine transformations*; i.e., transformations in which the ratios of distances between every pair of points are preserved. To represent affine transformations, so-called *homogeneous* coordinates must be used. This means describing an n-vector as an $(n+1)$-vector, by adding a 1. For example, when applying it to the case of the NAO, a joint coordinate in three dimensions (x, y, z) is represented by the vector $(x, y, z, 1)$. This is necessary in order to describe translations using matrix multiplication.

The translation matrix and the rotation matrixes for the rotation around the x-, y- and z-axis, are:

$$Tr(\boldsymbol{t}) = \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Knowing the degrees of freedom of the NAO's arm (i.e. shoulder pitch and roll, elbow pitch and yaw[5]), the position of the NAO's hand (the end effector) can now be described as:

$$T = R_x(\theta_1)R_z(\theta_2)Tr(\boldsymbol{l}_1)R_y(\theta_3)R_z(\theta_4)Tr(\boldsymbol{l}_2) \tag{3}$$

$\boldsymbol{l}_1 = (0, l_1, 0, 1)^{\mathrm{T}}$ is the translation along the first link of the chain, and $\boldsymbol{l}_2$ is the translation along the second link. Equation 3 can be broken down into smaller pieces and solved sequentially, in order to calculate the coordinates of the intermediate joints.

Inverse Kinematics Inverse Kinematics (IK) is the exact opposite of Forward Kinematics: the problem of calculating the joint configurations of a kinematic chain corresponding to the desired position of the end effector, or in other words, mapping the desired joint coordinate in Cartesian space back to the corresponding configurations in the joint space [7].

Let $\boldsymbol{\theta} = \theta_1, \theta_2, ..., \theta_n$ be the n joint configurations in the kinematic chain. Then let $\boldsymbol{s}$ be the end effector position, which can be described as a function of the joint configurations $\boldsymbol{s} = f(\boldsymbol{\theta})$, and $\boldsymbol{t}$ the target position. The Inverse Kinematics problem is to find values for $\boldsymbol{\theta}$ such that $\boldsymbol{s} = \boldsymbol{t}$. Since not all points in Cartesian space map to a joint configuration, there is no straightforward inverse function $f^{-1}(\boldsymbol{t}) = \boldsymbol{\theta}$ for Inverse Kinematics, as opposed to Forward Kinematics, for which a completely analytically derivable solution exists. Therefore, Inverse Kinematics solvers rely on numerical approaches.

The *Jacobian inverse technique* is a solution to the Inverse Kinematics problem that linearly approximates the inverse function $f^{-1}(\boldsymbol{t})$ using the Jacobian matrix [7]. The Jacobian is a function of the $\boldsymbol{\theta}$ values: $J(\theta)_{ij} = (\frac{\delta f_i(\theta_j)}{\delta \theta_j})_{ij}$ In the case of a single end effector, $i = 1$. So J will be a 1-by-n matrix for n values of θ, whose entries are vectors in $\mathbb{R}^3$. Since the entries in the Jacobian are first order partial derivatives of the joint system, the relative movement of the end effector $\Delta\boldsymbol{s}$ can be estimated as: $\Delta\boldsymbol{s} \approx J\Delta\boldsymbol{\theta}$.

Since the Inverse Kinematics problem is about finding the right joint configurations corresponding to a certain target position, the IK problem can be formulated as: $\Delta\theta = J^{-1}\Delta\boldsymbol{s}$. Unfortunately, the Jacobian is not always invertible. There are various ways to approximate the Jacobian inverse. It is possible to take the transpose instead, which is proven to be a good approximation when scaled by some small scalar α [3]. Another technique is taking the Moore-Penrose pseudoinverse, which is defined for all m-by-n matrices [2]. This is a better approximation, and generally converges to a solution more quickly. The Jacobian pseudoinverse, denoted by $J^{\dagger}$, can be calculated using one of the two following equations, depending on the number of rows and columns:

$$J^{\dagger} = J^T(JJ^T)^{-1} \text{ if } m < n, \text{ and } J^{\dagger} = (J^TJ)^{-1}J^T \text{ if } m > n$$

The Jacobian pseudoinverse can be used to approximate the joint configurations:

$$\Delta\theta = J^{\dagger}\Delta\boldsymbol{s} \tag{4}$$

When J is full row rank, (JJ^T) and (J^TJ) are guaranteed to be invertible. A general formula for the pseudoinverse for J not of full row rank can be found in [2]. Equation 4 can be applied iteratively until the error drops down to a certain threshold. The algorithm is easily implemented and is computationally fast. The big downside is its instability for configurations near singularities.

The Jacobian transpose as well as the Jacobian pseudo-inverse method are used in this paper for comparison purposes. These are natural choices for baseline algorithms, since the Jacobian inverse methods have been the standard in solving Inverse Kinematics for a very long time [7].

3 FABRIK

FABRIK (short for Forward And Backward Reaching Inverse Kinematics) is a heuristic method, developed by Aristidou and Lasenby [1], that tackles the Inverse Kinematics problem described in Section 2. Unlike traditional methods, FABRIK does not make use of calculations involving matrices or rotational angles. Instead, the IK problem is solved by finding the joint coordinates as being points on a line. These points are iteratively adjusted one at a time, until the end effector has reached the target position, or the error is sufficiently small. FABRIK starts at the end effector of the chain and works forwards, adjusting each joint along the way. Thereafter, it works backwards in the same way, in order to complete a full iteration. Since the use of rotational angles and matrices is avoided, the algorithm has low computational cost, converges quickly, and does not suffer from singularity problems. Furthermore, the algorithm produces realistic human-like poses and is easily implemented.

In Figure 1, a visualization of the algorithm is shown. The various steps of the algorithm, indicated with the letters (a) through (f) in Figure 1, are described in words below.

Since homogeneous coordinates are only used in Forward Kinematics, the n joint positions of the kinematic chain can be represented by the triplets $\boldsymbol{p}_i = (x_i, y_i, z_i)$ for $i = 1, 2, ..., n$, where $\boldsymbol{p}_1$ is the root joint and $\boldsymbol{p}_n$ the end effector (a). The target position is named $\boldsymbol{t}$ and the initial root position is named $\boldsymbol{b}$. The target position is reachable if the distance between the root joint and the target position, denoted as $dist$, is smaller than or equal to the sum of the distances between the joints $d_i = |\boldsymbol{p}_{i+1} - \boldsymbol{p}_i|$ for $i = 1, 2, ..., n-1$. If the target is reachable, the first stage of the algorithm starts. In this stage, named 'forward reaching', the joint positions are estimated by positioning the end effector on the target position $\boldsymbol{t}$ (b). The new position of the $n-1^{\text{th}}$ joint, $\boldsymbol{p}'_{n-1}$, lies on the line l_{n-1}, which passes through the point $\boldsymbol{p}_{n-1}$ and the new end effector position $\boldsymbol{p}'_n$, and has distance d_{n-1} from $\boldsymbol{p}'_n$ (c). Subsequently, the new joint position $\boldsymbol{p}'_{n-2}$ can be calculated by taking the point on the line l_{n-1} with distance d_{n-2} from $\boldsymbol{p}'_{n-1}$. The first stage of the algorithm is completed when all new joint positions have been calculated (d). The current estimate is not a feasible one, though, since the position of the root has changed. Therefore, a second stage of the algorithm is necessary to achieve a solution. This stage, named 'backward reaching', is similar to the first stage of the algorithm, only the operations are carried out the other way around: from the root to the end effector. The new root position $\boldsymbol{p}''_1$ is the initial root position $\boldsymbol{b}$ (e). The next joint position $\boldsymbol{p}''_2$ is then determined by taking the point on the line l_1, that passes through the points $\boldsymbol{p}''_1$ and $\boldsymbol{p}'_2$, with distance d_1 from $\boldsymbol{p}''_1$. This procedure is repeated for all other joints, and a full iteration is completed (f). The end effector is now closer to its target position. The algorithm is repeated until the end effector has reached its target, or the distance to the target is smaller than a user-defined threshold.

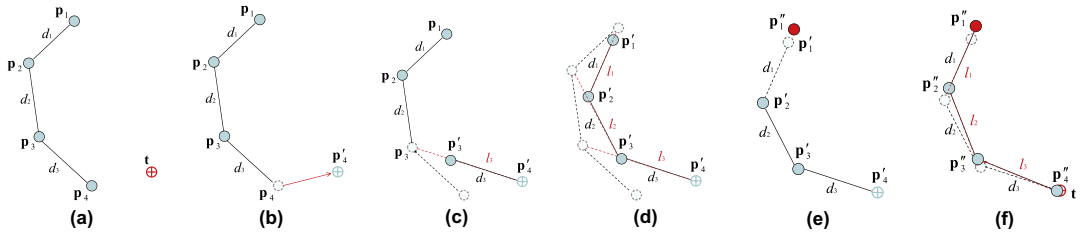

Figure 1: A visualization of one iteration of the FABRIK algorithm [1].

Calculating back to joint coordinates The FABRIK algorithm provides a solution to the inverse kinematics of the arm, by giving the Cartesian coordinates of each joint relative to the root joint. However, the NAO needs to know the joint configurations corresponding to these coordinates. A mapping from the Cartesian coordinates to joint configurations is therefore necessary in order to make the NAO move its arm.

The FABRIK algorithm outputs the three joint coordinates $(0, 0, 0, 1), (x_1, y_1, z_1, 1), (x_2, y_2, z_2, 1)$ to be the solution to the inverse kinematics problem for a target position $\boldsymbol{t} = (x_2, y_2, z_2, 1)$. Equation 5 describes the forward kinematics equation for mapping from the first two (currently unknown) rotations θ_1 and θ_2 to

the second joint coordinate $\boldsymbol{p}_1$:

$$\boldsymbol{p}_1 = R_x(\theta_1)R_z(\theta_2)Tr(l_1)\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^{\mathrm{T}} \tag{5}$$

which simply means taking the last column of the resulting transformation matrix. Because $\boldsymbol{p}_1$ is known, the joint angles can be derived from this equation. The expressions for the coordinates of the second joint can be found by rewriting Equation 5 as follows:

$$x_1 = -l_1\sin(\theta_2) \quad y_1 = l_1\cos(\theta_2)\sin(\theta_1) \quad z_1 = -l_1\cos(\theta_2)\sin(\theta_1) \tag{6}$$

Since the second rotation (the one around the x-axis) does not affect the x-coordinate itself, the following expression for the first rotation θ_2 can be derived. Next,the other rotation θ_1 can also be derived.

$$\theta_2 = \frac{-\arcsin(x_1)}{l_1} \quad \theta_1 = \frac{-\arcsin(z_1)}{l_1\cos(\theta_2)} \tag{7}$$

When expressing the end effector coordinates in the same manner (i.e. expressing the coordinates relative to the root joint), the expressions are not that simple anymore and the joint angles are not easily derivable anymore. So, the end effector coordinates have to be expressed relative to the second joint in the chain. Because the first joint angles are calculated, the orientation and position of the second joint can be captured in the following transformation matrix:

$$T = R_x(\theta_1)R_z(\theta_2)Tr(\boldsymbol{l}_1) \tag{8}$$

The end effector can be expressed relative to the second joint by multiplying its coordinates by the inverse of T:

$$\boldsymbol{p}_2' = T^{-1}\boldsymbol{p}_2 \tag{9}$$

Equation 10 describes the forward kinematics equation for mapping from the last two rotations (to be calculated) θ_3 and θ_4 to the relative end effector coordinate $\boldsymbol{p}_2'$:

$$\boldsymbol{p}_2' = R_y(\theta_3)R_z(\theta_4)Tr(\boldsymbol{l}_2)\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^{\mathrm{T}} \tag{10}$$

which can be rewritten in the same manner as in Equation 6:

$$\begin{aligned} x_2' &= -l_2\cos(\theta_3)\sin(\theta_4) & \theta_4 &= \frac{-\arccos(y_2')}{l_2} \\ y_2' &= l_2\cos(\theta_4) & & \\ z_2' &= l_2\sin(\theta_3)\sin(\theta_4) & \theta_3 &= \frac{-\arcsin(z_2')}{l_2\sin(\theta_4)} \end{aligned} \tag{11}$$

4 Neural networks

A neural network[10] is a computational architecture used in the field of machine learning, inspired by the highly interconnected structure of the brain, aiming to benefit from properties such as parallelism, generalization, fault tolerance, and adaptivity. It is a learning method that learns a mapping from input to output by being fed training examples and minimizing the error between the network's output and the desired output using a learning algorithm. Such a mapping is called a neural network *model*. Neural networks have previously been applied to Inverse Kinematics[8].

A neural network can be trained in several ways, of *backpropagation* is one of the most popular [10]. Backpropagation tries to find the minimum of the error function by following using *gradient descent*: it minimizes the function by taking steps towards the negative gradient of the error function. Therefore, in order to be able to use backpropagation, the error function has to be differentiable. This is achieved by choosing a continuous activation function, since the network function is a composition of these activation

functions and is therefore by definition also continuous. This makes the error function with respect to the network weights continuous as well.

The advantage of backpropagation is that it is straightforward to understand and implement. However, convergence of the algorithm is not guaranteed and is typically very slow. It may also converge to a local minimum instead of a global minimum. To overcome these problems, an adaptive gradient search-based method has been developed. Resilient backpropagation[9] does only take into account the sign of the gradient and not the magnitude, and scales it by a weight-specific update value. This allows for faster convergence speed and provides more robustness against local minima (although they may still occur).

The design choices of the neural network used in this paper are the number of input and output nodes, the number of hidden layers and nodes, the activation function, and the learning algorithm used to train the neural networks. The network consists of three input nodes, one for each target coordinate, and four output nodes, one for each of the joint configurations that have to be adjusted. If the function we are trying to fit is linear, there would be no need for a hidden layer at all. Since Inverse Kinematics is a non-linear optimization problem, hidden layers are needed. In practice, one or two layers is sufficient to approximate any function, if there are enough hidden nodes in the layers. Experiments with different configurations, see Section 5, showed that one hidden layer containing 40 nodes gives the best results. The training algorithm of choice is the RPROP algorithm as described in [9], since this proposed modification of the backpropagation algorithm has shown to be a great improvement in the training speed. The activation function has to be a continuous function, since RPROP relies on the fact that the error function has to be differentiable. Experiments showed that the sigmoid activation function gives the best results.

5 Experiments

The traditional IK algorithms and their alternatives have been evaluated in a series of experiments. All four algorithms are compared to each other regarding computation time. Moreover, the Jacobian transpose, Jacobian pseudoinverse and FABRIK are compared w.r.t. the number of iterations and computation time different error tolerances. The neural network is evaluate for different configurations. Due to space limitations only the main results concerning the number of nodes will be presented.

Algorithm	**Time** (ms)	μ **iterations**	σ **iterations**	**unsolved**
FABRIK	0.78	7.26	7.38	0
J. pseudoinverse	2.21	9.99	10.02	185
J. transpose	27.65	141.8	176.4	0
Neural network	0.002771	n/a	n/a	0

Table 1: Comparing the algorithms regarding computation time, the average number of iterations and its standard deviation.

Table 1 shows the experimental results of the four algorithms regarding computation time, the average number of iterations and the standard deviation of the number of iterations. The methods are evaluated using a dataset consisting of 10,000 uniformly distributed random samples from the set of reachable target positions. The training and evaluating of the networks was performed on separate validation sets of identical size to avoid overfitting. The sets were generated by applying Forward Kinematics to all possible combinations of allowed joint angles. The error tolerance was set to 0.1cm. Since the neural network is not an iterative method and computes the outputs instantly, no iteration results are listed for the neural network. The percentage of unsolved instances of the Jacobian pseudoinverse method is due to the unstable behavior near singularities. If a joint configuration is close to a singularity, the pseudoinverse method will lead to large changes in joint angles, even for small movements of the target position [3].

Figure 2 shows the computation time of the three iterative algorithms for different error tolerances. The Jacobian transpose method clearly performs worse for a lower error tolerance, whereas the Jacobian pseudoinverse shows only a slight increase in computation time and FABRIK remains constant. FABRIK was the only algorithm that yielded results when the error tolerance was set to zero, taking the same computation

time as it needed for other error tolerances. Figure 2 does not include the neural network, since it is not an iterative method and the error is not a variable that can be set but a given number measured from the output.

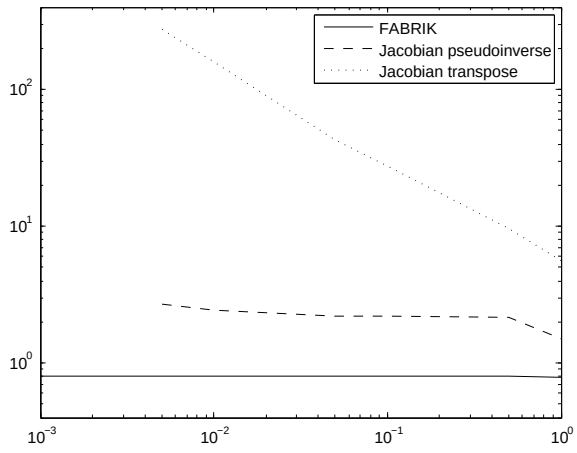

Figure 2: Comparing the change in computation time when decreasing the error tolerance.

The Neural Network was evaluated for 5 to 100 hidden nodes. Table 2 shows some of the results. All networks were trained until the error rates did not decrease anymore. Again, the neural networks were trained on a training set with 10,000 uniformly distributed random samples from the set of reachable target positions, and their corresponding joint configurations. The entries were normalized so that their values lie between 0 and 1. The trained networks were tested on a validation set of identical size to avoid overfitting. Since the error rate does not decrease any more when more than 40 nodes are used, this number of nodes was used in the experiments described above. An error of 2.38% for a kinematic chain of length 21.87 centimeters corresponds to a maximum error of 0.52 centimeters at the end effector.

# of hidden nodes	Training error	Validation error
30	2.37%	2.40%
40	2.35%	2.38%
50	2.35%	2.38%

Table 2: Experimentally determining the number of nodes in the hidden layer.

6 A NAO robot paying tic-tac-toe

As a proof of concepts we used FABRIK to enable a NAO humanoid robot to play tic-tac-toe. The NAO identifies the games state using its camera. The camera image is analyzed using algorithms from the OpenCV library, and mapped to a game state. A simple mini-max algorithm is used to determine the next move. Finally, FABRIK is used to coordinate the hand movements necessary to draw a cross or a circle. Figure 3 gives an illustration.

7 Conclusion

FABRIK outperforms the Jacobian transpose method as well as the Jacobian pseudoinverse method in terms of calculation time and number of iterations. Furthermore, it is the only algorithm that could always yield results with the error tolerance set to zero, and is well-behaved near singularities. The Jacobian pseudoinverse performs slightly slower than FABRIK, but is still decent. However, it does not always yield a solution

Figure 3: A NAO playing tic-tc-toe.

near singularities, or when the error tolerance is set to zero. The Jacobian transpose method does not suffer from singularity problems, but its computation time explodes when the error tolerance is reduced.

If, however, computation time is of the highest priority, and precision is secondary, neural networks might be a more suitable approach. It greatly outperforms all the other methods regarding speed, but is the least accurate method of all, with a possible error of 2.38%. This translates to 0.52 centimeters on the NAO, but when scaled to bigger applications, this might be a more significant number. Therefore, this neural network is suited for applications which require a fast response time and do not need to be that accurate.

It is safe to say that FABRIK is a better alternative to the Jacobian pseudo-inverse method and the Jacobian transpose method when applied to the NAO humanoid robot, since it performs better, is more accurate, and converges more quickly. Moreover, it does not suffer from singularity problems and is therefore a well-behaved approximation in each possible case. A general conclusion about FABRIK and neural networks, however, cannot be drawn from this paper. This would require a comparison with other traditional methods such as DLS/SDLS described in Section 1, or techniques that were not discussed in this paper such as Cyclic Coordinate Descent (CCD) [11] and Elman networks [6]. Finally, FABRIK should be compared with other algorithms for kinematic chains with more than two joints.

References

[1] Andreas Aristidou and Joan Lasenby. Fabrik: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 2011.

[2] Samuel R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, 2003.

[3] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. 2009.

[4] Martin Fedor. Application of inverse kinematics for skeleton manipulation in real-time. *Proceedings of the 19th spring conference on Computer graphics*, 2003.

[5] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, Jerome Monceaux, P. Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of nao humanoid. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 769–774, 2009.

[6] Rait Kker. A neuro-genetic approach to the inverse kinematics solution of robotic manipulators. *Scientific Research and Essays*, 2011.

[7] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, 1981.

[8] Juan Pereda, Javier Lope, and Daro Maravall. Comparative analysis of artificial neural network training methods for inverse kinematics learning. In Roque Marn, Eva Onainda, Alberto Bugarn, and Jos Santos, editors, *Current Topics in Artificial Intelligence*, volume 4177 of *Lecture Notes in Computer Science*, pages 171–179. Springer Berlin Heidelberg, 2006.

[9] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE International Conference on Neural Networks*, 1993.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2009.

[11] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993.

[12] W. Wolovich and H. Elliott. A computational technique for inverse kinematics. *IEEE Conference on Decision and Control*, 2009.